Demostraciones de NP-Dureza y NP-Completitud

Ariel González Gómez

December 22, 2024

1 Introducción

La teoría de la complejidad computacional clasifica problemas de decisión en función de su dificultad relativa y los recursos necesarios para resolverlos. Dentro de esta clasificación, los problemas NP-Completos representan un desafío central, ya que capturan la esencia de los problemas más difíciles en la clase NP. Paralelamente, los problemas NP-Hard, que pueden no pertenecer a NP, son al menos tan difíciles como los problemas NP-Completos.

En este artículo, abordamos un conjunto representativo de problemas clásicos y demostramos su pertenencia a estas categorías a través de reducciones polinomiales desde problemas ya conocidos como NP-Completos. Estas demostraciones no solo corroboran su complejidad inherente, sino que también destacan sus implicaciones en áreas prácticas como la optimización, la teoría de grafos, y la lógica computacional.

La estructura del artículo se organiza como sigue: en la Sección 2, se presenta el marco teórico y las definiciones clave. En las secciones siguientes se desarrolla cada demostración de forma detallada, abordando tanto la construcción de las reducciones como las verificaciones necesarias.

2 Marco Teórico y Definiciones Clave

En esta sección, establecemos las bases teóricas necesarias para las demostraciones de NP-completitud y NP-dureza presentadas en este trabajo. Revisamos los conceptos fundamentales de las clases de complejidad P, NP, NP-Completo y NP-Duro, y describimos los métodos de reducción polinomial, que son el núcleo de las pruebas de complejidad.

2.1 Clases de Complejidad: P, NP, NP-Completo y NP-Duro

- **P**: Es la clase de problemas de decisión que pueden ser resueltos en tiempo polinomial por una máquina determinista. Formalmente, un problema L pertenece a P si existe un algoritmo determinista que decide L en tiempo $O(n^k)$, donde n es el tamaño de la entrada y k es una constante.

- **NP**: Es la clase de problemas de decisión para los cuales, si la respuesta es afirmativa, existe un certificado verificable en tiempo polinomial por una máquina determinista. En otras palabras, un problema L está en NP si para cada $x \in L$, existe una prueba y tal que una máquina determinista puede verificar que (x, y) satisface las condiciones de L en tiempo polinomial.

- **NP-Completo**: Un problema es NP-Completo si cumple dos condiciones: (1) pertenece a NP, y (2) todo problema en NP es reducible a él en tiempo polinomial. Los problemas NP-Completos son considerados los más difíciles dentro de NP, ya que resolver cualquiera de ellos en tiempo polinomial implicaría que P = NP.

- **NP-Duro**: Un problema es NP-Duro si todo problema en NP es reducible a él en tiempo polinomial. A diferencia de los problemas NP-Completos, los NP-Duros no necesitan pertenecer a NP; pueden ser más generales, como problemas de optimización.

2.2 Reducción Polinomial

La reducción polinomial es una herramienta fundamental para demostrar que un problema es NP-Completo o NP-Duro. Dado un problema A y un problema B, decimos que A se reduce polinomialmente a B (denotado $A \leq_P B$) si existe una función computable en tiempo polinomial f tal que, para toda entrada x, $x \in A$ si y solo si $f(x) \in B$.

La reducción polinomial establece que B es al menos tan difícil como A. Si A es conocido como NP-Completo y se demuestra que $A \leq_P B$, entonces B también es NP-Completo, siempre y cuando $B \in \text{NP}$.

2.3 Problemas Base

En este trabajo, utilizamos problemas ya conocidos como NP-Completos para construir reducciones hacia los problemas que analizamos. Los problemas base utilizados incluyen:

- **SAT (Satisfiabilidad):** Determinar si una fórmula booleana tiene una asignación de valores que la haga verdadera.
- **3-SAT:** Una variante de SAT donde cada cláusula contiene exactamente tres literales.
- **k - Clique:** Encontrar un subgrafo completo de tamaño k en un grafo.
- **k - Conjunto Independiente:** Determinar si existe un conjunto de tamaño k de vértices de un grafo tal que ninguno de ellos esté conectado por una arista.
- **Vertex Cover:** Encontrar un conjunto mínimo de vértices que cubra todas las aristas de un grafo.
- **Subset Sum:** Determinar si existe un subconjunto de un conjunto de números enteros que sume un valor objetivo.
- **Mochila (Knapsack):** Resolver el problema de optimización donde se seleccionan elementos con peso y valor para maximizar el valor total sin exceder una capacidad dada.
- **Hamilton (Dirigido):** Determinar si existe un ciclo hamiltoniano en un grafo dirigido.
- **Viajante (TSP):** Encontrar el recorrido de menor costo que visite cada ciudad exactamente una vez y regrese a la ciudad de origen.

2.4 Metodología de las Demostraciones

Cada demostración presentada en este trabajo sigue los siguientes pasos:

1. **Selección del problema base:** Se elige un problema conocido como NP-Completo o NP-Duro para realizar la reducción.
2. **Construcción de la instancia:** Se diseña una transformación polinomial que mapea instancias del problema base hacia instancias del problema objetivo.
3. **Prueba de equivalencia:** Se demuestra que la instancia transformada es equivalente en términos de solución, es decir, que resolver el problema objetivo implica resolver el problema base y viceversa.
4. **Clasificación final:** Con base en la pertenencia del problema a NP y la reducción, se concluye si el problema es NP-Completo o NP-Duro.

Esta sección proporciona el marco necesario para entender y contextualizar las demostraciones que se presentan en las secciones posteriores.

3 *Exact Cover*

3.1 Definición del problema *Exact Cover*

El problema de *Exact Cover* puede describirse como sigue: Dado un conjunto X y una colección S de subconjuntos de X , el problema consiste en determinar si existe un subcolector $S' \in S$ tal que cada elemento de X aparezca exactamente una vez en los subconjuntos de S'.

3.2 Pertenencia a NP

Para demostrar que el problema *Exact Cover* pertenece a la clase NP, debemos mostrar que, dado un conjunto X, una colección S de subconjuntos de X, y una solución candidata $S' \subseteq S$, es posible verificar en tiempo polinomial si S' constituye una solución válida al problema. En otras palabras, debemos verificar si cada elemento de X aparece exactamente una vez en los subconjuntos de S'.

Sea $|X| = n$ y $|S| = m$, la verificación puede realizarse mediante el siguiente procedimiento:

1. Inicializar un contador $count(x)$ para cada elemento $x \in X$ en 0.
2. Recorrer cada subconjunto $S_i \in S'$ y para cada elemento $x \in S_i$, incrementar $count(x)$ en 1.
3. Verificar que $count(x) = 1$ para todos los $x \in X$. Si esta condición se cumple, entonces S' es una solución válida; de lo contrario, no lo es.

El costo computacional de este procedimiento es como sigue:

- Inicializar los contadores para todos los elementos en X tiene un costo de $O(n)$.
- Recorrer todos los subconjuntos en S' y todos sus elementos requiere un tiempo proporcional al tamaño total de los subconjuntos seleccionados, que es $O(n \cdot m)$ en el peor caso.
- Verificar los valores de los contadores tiene un costo de $O(n)$.

Por lo tanto, el tiempo total de verificación es $O(n \cdot m)$, lo cual es polinomial en el tamaño de la entrada.

Dado que se puede verificar en tiempo polinomial si una solución candidata S' es válida, concluimos que el problema *Exact Cover* pertenece a la clase NP.

3.3 NP-Dureza

Este problema es NP-Duro, y haremos la demostración de esto mediante la reducción desde el problema de satisfacibilidad (SAT).

La demostración de que *Exact Cover* es NP-completo mediante la reducción desde el problema de satisfacibilidad (SAT) se organiza en dos pasos principales:

1. **Construcción de la instancia del problema Exact Cover ($\tau(F)$)** a partir de una fórmula de SAT.
2. **Prueba de equivalencia**: demostrar que la fórmula SAT es satisfacible si y sólo si $\tau(F)$ tiene una solución de cobertura exacta.

A continuación, desarrollamos esta demostración:

1. Construcción de la instancia de Exact Cover

Fórmula de entrada:

Dada una fórmula de SAT:

$$F = \{C_1, C_2, \ldots, C_\ell\},$$

donde cada cláusula $C_j = (L_{j1} \vee L_{j2} \vee \cdots \vee L_{jm_j})$ está formada por m_j literales L_{jk}, construimos una instancia de *Exact Cover* como sigue:

Universo U:

El universo U para el problema *Exact Cover* incluye:

1. **Variables de SAT**:
$$\{x_1, x_2, \ldots, x_n\},$$
donde x_i representa cada variable de la fórmula.

2. **Cláusulas de SAT**:
$$\{C_1, C_2, \ldots, C_\ell\},$$
donde C_j representa cada cláusula de la fórmula.

3. **Literales individuales**: Para cada literal en cada cláusula C_j, agregamos elementos únicos:
$$\{p_{jk} \mid 1 \leq j \leq \ell, 1 \leq k \leq m_j\},$$
donde p_{jk} corresponde al literal L_{jk} en la cláusula C_j.

El universo queda definido como:

$$U = \{x_1, x_2, \ldots, x_n\} \cup \{C_1, C_2, \ldots, C_\ell\} \cup \{p_{jk} \mid 1 \leq j \leq \ell, 1 \leq k \leq m_j\}.$$

Subconjuntos S:

Definimos los subconjuntos de U como sigue:

1. Para cada literal p_{jk}:
$$S_{p_{jk}} = \{p_{jk}\}.$$

2. Para cada variable x_i:
   - Si x_i es **verdadero** (T):
   $$S_{x_i,T} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = \neg x_i\}.$$
   (Contiene x_i y todas las ocurrencias negativas de x_i).
   - Si x_i es **falso** (F):
   $$S_{x_i,F} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = x_i\}.$$
   (Contiene x_i y todas las ocurrencias positivas de x_i).

3. Para cada cláusula C_j:
$$S_{C_{jk}} = \{C_j, p_{jk}\},$$
para $1 \leq k \leq m_j$, asociando C_j con cada literal de la cláusula.

Con esta construcción, S contiene:

- n pares de conjuntos ($S_{x_i,T}$ y $S_{x_i,F}$) para las variables.
- m conjuntos individuales $S_{p_{jk}}$ para los literales.
- $\ell \cdot m_j$ conjuntos $S_{C_{jk}}$ para las cláusulas.

2. Prueba de equivalencia

Dirección 1: Si F es satisfacible, entonces $\tau(F)$ tiene una cobertura exacta.

Si existe una asignación de verdad v que satisface F:

1. Para cada variable x_i, seleccionamos:
   - $S_{x_i,T}$ si $v(x_i) = \text{true}$.
   - $S_{x_i,F}$ si $v(x_i) = \text{false}$.
2. Para cada cláusula C_j, seleccionamos un subconjunto $S_{C_{jk}}$ que cubra C_j, correspondiente a un literal L_{jk} que sea verdadero bajo v.
3. Finalmente, para cada literal p_{jk} que no haya sido cubierto por las selecciones anteriores, seleccionamos el conjunto $S_{p_{jk}}$.

Esta selección cubre exactamente todos los elementos de U, cumpliendo con las condiciones del *Exact Cover*.

Dirección 2: Si $\tau(F)$ tiene una cobertura exacta, entonces F es satisfacible.

Si existe una cobertura exacta C para $\tau(F)$:

1. Para cada variable x_i, el conjunto seleccionado en C será $S_{x_i,T}$ o $S_{x_i,F}$. Esto determina una asignación de verdad para x_i.
2. Para cada cláusula C_j, existe un conjunto $S_{C_{jk}}$ en C que cubre C_j. Esto garantiza que al menos un literal de cada cláusula sea verdadero bajo la asignación construida.

Por lo tanto, F es satisfacible.

Conclusión

Hemos demostrado que F es satisfacible si y sólo si $\tau(F)$ tiene una solución de cobertura exacta. Como *SAT* es NP-Duro y la reducción se realiza en tiempo polinomial, concluimos que **Exact Cover** es NP-Duro y, al ser además NP, es NP-Completo.

4 *Max Clique*

4.1 Definición del problema *Max Clique*

Dado un grafo $G = (V, E)$, el problema *Max Clique* consiste en encontrar el clique de mayor tamaño, es decir, el conjunto máximo de vértices $C \subseteq V$ tal que para todo par de vértices $u, v \in C$, se cumple que $(u, v) \in E$. Este problema busca maximizar el tamaño del clique.

4.2 Relación con el problema *k-Clique*

Sabemos que el problema *k-Clique* es NP-Completo. En este problema, dado un grafo G y un entero k, la tarea es decidir si existe un clique de tamaño al menos k en G.

Resolver el problema *Max Clique* implica resolver el problema *k-Clique* de la siguiente manera:

1. Si disponemos de un algoritmo para *Max Clique*, calculamos el tamaño del clique máximo $k_{\max}$ en G.
2. Comparamos $k_{\max}$ con k:
   - Si $k_{\max} \geq k$, entonces G contiene un clique de tamaño al menos k.
   - En caso contrario, no existe tal clique.

Por lo tanto, resolver *Max Clique* en tiempo polinomial permitiría resolver *k-Clique* en tiempo polinomial.

4.3 Reducción de *k-Clique* a Max *Clique*

Dado que el problema *k-Clique* es NP-Completo, y hemos demostrado que una solución para *Max Clique* permite resolver *k-Clique*, concluimos que *Max Clique* es al menos tan difícil como *k-Clique*.

En otras palabras, existe una reducción polinomial desde *k-Clique* a *Max Clique*. Esto implica que *Max Clique* es un problema NP-Hard.

4.4 Pertenencia a NP

Para que un problema sea NP-Completo, debe cumplir dos condiciones:

1. Pertenecer a NP: su solución puede ser verificada en tiempo polinomial dado un certificado.
2. Ser NP-Hard: cualquier problema en NP puede ser reducido a él en tiempo polinomial.

Aunque hemos demostrado que *Max Clique* es NP-Hard, el problema no pertenece a NP, ya que encontrar el clique máximo no tiene una versión de decisión natural que permita verificar su solución en tiempo polinomial.

Conclusión

El problema *Max Clique* es NP-Hard, ya que resolverlo permite resolver cualquier instancia del problema NP-Completo *k-Clique* en tiempo polinomial. Sin embargo, *Max Clique* no es NP-Completo, pues no pertenece a NP.

5 *3-Colouring*

5.1 Definición del problema *3-Colouring*

El problema de *3-Colouring* consiste en determinar si, dado un grafo $G = (V, E)$, es posible asignar a los vértices de G un conjunto de tres colores de manera que ningún par de vértices adyacentes compartan el mismo color. Formalmente, el problema se define como:

> **3-Colouring:** Dado un grafo $G = (V, E)$, devolver *1* si y solo si existe una asignación de colores $c : V \to \{1, 2, 3\}$ tal que $\forall (u, v) \in E,\ c(u) \neq c(v)$.

5.2 Pertenencia a NP

El problema *3-Colouring* pertenece a la clase NP, ya que una solución candidata puede ser verificada en tiempo polinomial. Dada una asignación de colores c para los vértices de G, el verificador puede recorrer todas las aristas de G y comprobar que los extremos de cada arista tienen colores diferentes. Este proceso requiere tiempo $O(|E|)$, donde $|E|$ es el número de aristas en el grafo.

5.3 NP-Dureza

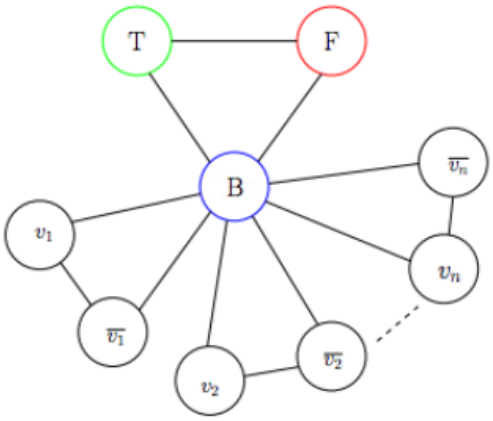
Para demostrar que *3-Colouring* es NP-Hard, realizamos una reducción polinomial desde el problema *3-SAT*, que es NP-Completo.

5.3.1 Construcción de la reducción

La construcción del grafo $G = (V, E)$ busca capturar dos aspectos clave:

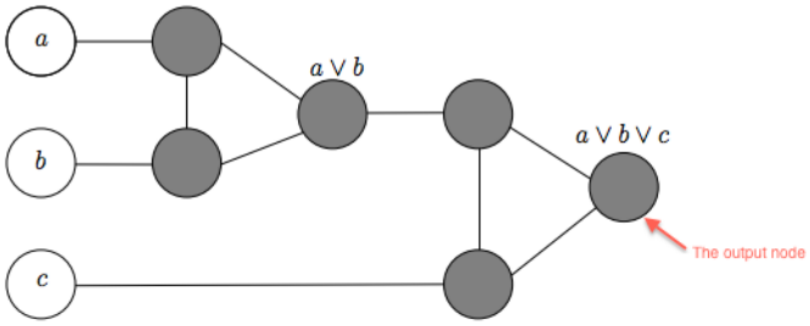
1. Establecer una asignación de verdad para las variables $x_1, x_2, \ldots, x_n$ mediante los colores de los vértices de G.
2. Capturar la satisfacibilidad de cada cláusula C_i en φ.

Para lograr el primero de estos objetivos, creamos un triángulo en G con tres vértices etiquetados como T, F y B, que representan los colores *True*, *False* y *Base*. Este triángulo garantiza que el grafo requiera tres colores para ser coloreado correctamente. A continuación, para cada variable x_i, agregamos dos vértices v_i y $\overline{v}_i$, que representan la variable y su negación, respectivamente. Estos vértices se conectan al vértice B para formar un triángulo $\{B, v_i, \overline{v}_i\}$:

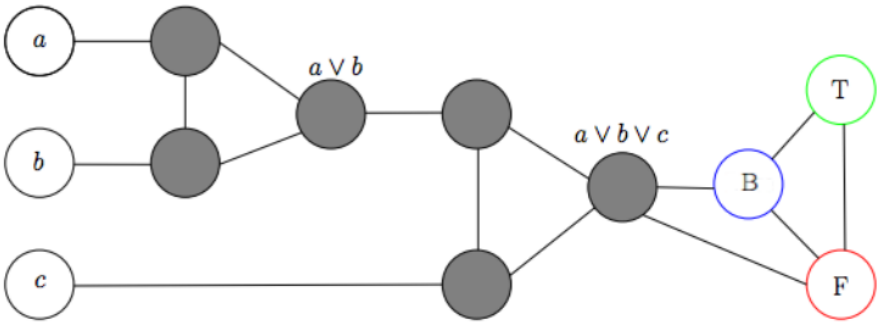


Esta estructura asegura que v_i y $\overline{v}_i$ no puedan compartir el mismo color; en particular, si v_i es T, entonces $\overline{v}_i$ debe ser F, y viceversa.

Para capturar la satisfacibilidad de cada cláusula $C_j = (a \vee b \vee c)$, introducimos un *gadget* llamado *OR-Gadget*. Este gadget incluye un vértice de salida etiquetado como $a \vee b \vee c$, que representa la salida de la cláusula. Este vértice se conecta a los literales a, b y c, y se diseña de tal manera que si al menos uno de los literales tiene el color T, el vértice de salida también puede colorearse como T. Si todos los literales tienen el color F, el vértice de salida debe colorearse como F, lo que captura la insatisfacibilidad de la cláusula.



Además, el vértice de salida de cada gadget se conecta a los vértices B y F del triángulo inicial, lo que asegura que solo pueda ser coloreado como T.



5.3.2 Correctitud de la reducción

La correctitud de la reducción se comprueba demostrando que: φ es satisfacible $\Leftrightarrow$ G es 3-colorable.

($\Rightarrow$) Si φ es satisfacible, entonces G es 3-colorable. Para cada variable x_i, asignamos T a v_i y F a $\overline{v}_i$ (o viceversa) según la asignación que satisface φ. Para cada gadget de cláusula, el vértice de salida puede colorearse como T si al menos uno de los literales está coloreado como T, lo cual es posible porque la cláusula es satisfacible.

($\Leftarrow$) Si G es 3-colorable, entonces φ es satisfacible. Para cada variable x_i, asignamos True si v_i está coloreado como T y False si $\overline{v}_i$ está coloreado como T. Dado que el gadget de cada cláusula tiene un vértice de salida coloreado como T, al menos uno de los literales en la cláusula debe ser verdadero, lo que garantiza que φ sea satisfacible.

5.4 Conclusión

Hemos demostrado que *3-Colouring* pertenece a NP y que es NP-Hard mediante una reducción polinomial desde *3-SAT*. Por lo tanto, concluimos que *3-Colouring* es NP-Completo.

6 Número Cromático

6.1 Definición del problema de Número Cromático

El número cromático de un grafo es el número mínimo de colores necesarios para colorear los vértices del grafo de manera que dos vértices adyacentes no compartan el mismo color. El problema consiste en determinar el número cromático de un grafo dado. Formalmente, se define como:

> **Número Cromático:** Dado un grafo $G = (V, E)$, determinar el mínimo entero k tal que existe una asignación de colores $c : V \to \{1, 2, \ldots, k\}$ tal que $\forall (u, v) \in E$, $c(u) \neq c(v)$.

6.2 Demostración de NP-Dureza

Para demostrar que el problema de Número Cromático es NP-Hard, realizamos una reducción desde el problema *3-Colouring*, que ya se ha demostrado como NP-Completo.

6.2.1 Reducción desde 3-Colouring

Sea $G = (V, E)$ una instancia del problema *3-Colouring*. Queremos construir una instancia del problema de Número Cromático tal que resolver esta nueva instancia permita resolver el problema *3-Colouring*.

Dado que en *3-Colouring* se pregunta si es posible colorear G con a lo sumo 3 colores, la solución del problema de Número Cromático en G nos dará el mínimo número de colores necesarios para colorear G. Si el número cromático de G menor o igual que 3, entonces G es 3-colorable, y por lo tanto la instancia de *3-Colouring* tiene una respuesta afirmativa. Si el número cromático de G es mayor que 3, entonces G no es 3-colorable.

6.2.2 Correctitud de la Reducción

La reducción es válida porque:

- Resolver el problema de Número Cromático implica conocer el número mínimo de colores necesarios para colorear G.
- Si el número cromático es menor o igual que 3, entonces G admite una coloración con a lo sumo 3 colores, resolviendo la instancia de *3-Colouring*.
- Si el número cromático es mayor que 3, G no puede ser 3-colorable, lo que también resuelve la instancia de *3-Colouring*.

6.3 Conclusión

Hemos demostrado que resolver el problema de Número Cromático permite resolver *3-Colouring*, lo que implica que el problema de Número Cromático es al menos tan difícil como *3-Colouring*. Como *3-Colouring* es NP-Completo, concluimos que el problema de Número Cromático es NP-Hard.

7 Cobertura de Clique

7.1 Definición del problema de Cobertura de Clique

Dado un grafo $G = (V, E)$, una *cobertura de cliques* es un conjunto de cliques $\{C_1, C_2, \ldots, C_k\}$ tal que cada arista $(u, v) \in E$ pertenece a al menos uno de estos cliques. El objetivo del problema es determinar el número mínimo de cliques necesarios para cubrir todas las aristas del grafo, conocido como el *número de cobertura de cliques*.

7.2 Relación con la Coloración del Complemento

El problema de Cobertura de Clique tiene una relación directa con el problema de coloración de grafos aplicada al complemento del grafo G. Recordemos que:

- Un conjunto de vértices es una *clique* en G si y solo si es un conjunto independiente (o estable) en el grafo complemento $\overline{G}$.
- Una cobertura de cliques de G corresponde a una partición de los vértices de G en cliques, que es equivalente a una partición de los vértices de $\overline{G}$ en conjuntos independientes.
- Como las coberturas de cliques, las coloraciones de grafos son particiones del conjunto de vértices, pero en subconjuntos sin adyacencias (conjuntos independientes), en vez de cliques. Un subconjunto de vértices es un clique en G si y solo si es un conjunto independiente en el complemento de G, por tanto una partición de los vértices de G es una cobertura de cliques de G si y solo si es una coloración del complemento de G.

Por lo tanto, encontrar el número mínimo de cliques necesarios para cubrir las aristas de G es equivalente a encontrar el número cromático (mínimo número de colores) necesario para colorear $\overline{G}$.

7.3 Pertenencia a NP

El problema de Cobertura de Clique pertenece a NP porque, dado un número k y una partición de los vértices en k cliques, podemos verificar en tiempo polinomial que cada arista en E está cubierta por al menos uno de los cliques de la partición.

7.4 NP-Dureza

Para demostrar que el problema es NP-Hard, reducimos el problema de *Coloración* al problema de Cobertura de Clique. Sabemos que el problema de Coloración es NP-Completo; dado un grafo H, queremos determinar si se puede colorear con k colores.

Construimos el complemento de H, $\overline{H}$, como una instancia del problema de Cobertura de Clique. Resolver el problema de Cobertura de Clique en $\overline{H}$ (encontrando el número mínimo de cliques necesarios para cubrir $\overline{H}$) equivale a resolver el problema de Coloración en H, ya que el número de colores necesarios para colorear H es igual al número de cliques necesarios para cubrir $\overline{H}$.

7.5 Conclusión

Hemos demostrado que el problema de Cobertura de Clique es equivalente al problema de Coloración aplicado al complemento del grafo. Como el problema de Coloración es NP-Completo, y la reducción entre ambos problemas es polinomial, concluimos que el problema de Cobertura de Clique es NP-Completo.

8 *Dominant Set*

8.1 Definición del problema *Dominant Set*

En un grafo $G = (V, E)$, un conjunto $D \subseteq V$ es un *conjunto dominante* si cada vértice $v \in V$
D es adyacente a al menos un vértice en D. El *número dominante* es la cardinalidad del menor conjunto dominante de G. El problema consiste en determinar el número dominante de un grafo.

8.2 Pertenencia a NP

El problema pertenece a NP porque, dado un conjunto candidato D, podemos verificar en tiempo polinomial si D es un conjunto dominante. Para ello, verificamos que cada vértice $v \in V$
D tiene al menos un vecino en D. Esto puede realizarse recorriendo todas las aristas del grafo en tiempo $O(|E|)$.

8.3 NP-Dureza

Para demostrar que el problema Dominant Set es NP-Hard, realizamos una reducción desde el problema *Vertex Cover*, que es conocido como NP-Completo. La construcción de la reducción se realiza como sigue:

8.3.1 Construcción de la Reducción

Primero, notemos que un conjunto dominante debe incluir todos los vértices aislados (aquellos que no tienen vértices adyecentes). Por tanto, asumimos que nuestro grafo de entrada no posee vértices aislados.

Dado un grafo $G = (V, E)$ como instancia del problema *Vertex Cover*, construimos un nuevo grafo $G' = (V', E')$ como sigue:

1. Mantener todos los vértices y aristas de G en G'.
2. Para cada arista $\{u, v\} \in E$, agregar un nuevo vértice w en V' y dos nuevas aristas $\{u, w\}$ y $\{w, v\}$ en E'.
3. Mantener las aristas originales $\{u, v\} \in E$ en G'.

Este procedimiento asegura que cualquier arista en G tiene un vértice intermedio w que crea un camino alternativo entre u y v en G'.

8.3.2 Correctitud de la Reducción

($\Rightarrow$) Si G tiene un Vertex Cover de tamaño k, entonces ese mismo conjunto de vértices forma un conjunto dominante de tamaño k en G'. Como S cubre todas las aristas de G, en G' cada vértice adicional w agregado para una arista $\{u, v\}$ es adyacente a un vértice de S. Por lo tanto, S domina todos los vértices de G'.

($\Leftarrow$) Si G' tiene un conjunto dominante D de tamaño k, entonces D puede ajustarse para que no incluya vértices intermedios w. Cada vértice w es adyacente únicamente a los extremos u y v de una arista original en G, lo que significa que D puede modificarse reemplazando w por uno de sus vecinos. De esta manera, D cubre todas las aristas de G, y por lo tanto corresponde a un Vertex Cover de tamaño k en G.

8.4 Conclusión

Hemos demostrado que existe un conjunto dominante de tamaño k en G' si y solo si existe un Vertex Cover de tamaño k en G. Como el problema *Vertex Cover* es NP-Completo, y la reducción es polinomial, concluimos que el problema Dominant Set es NP-Completo.

9 *Domatic Number*

9.1 Definición del problema *Domatic Number*

Dado un grafo $G = (V, E)$, el número domático (*Domatic Number*) de G es el número máximo k tal que los vértices de G pueden particionarse en k conjuntos disjuntos $D_1, D_2, \ldots, D_k$, donde cada conjunto D_i es dominante. Un conjunto D_i es dominante si cada vértice $v \in V \setminus D_i$ es adyacente a al menos un vértice de D_i. El problema consiste en determinar el número domático de un grafo.

9.2 NP-Dureza

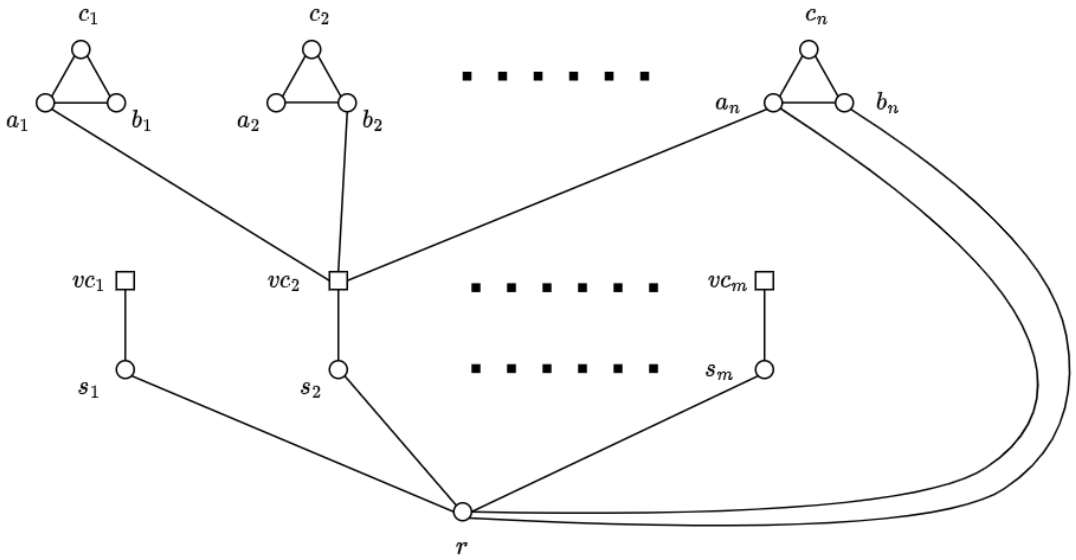
Para demostrar que el problema Domatic Number es NP-Hard, realizamos una reducción desde el problema *3-SAT*, conocido por ser NP-Completo.

9.2.1 Construcción de la Reducción

Sea $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ una instancia de *3-SAT* con n variables $x_1, \ldots, x_n$. Construimos un grafo $G = (V, E)$ como sigue:

1. Para cada cláusula C_i en ϕ, agregamos un vértice vc_i al conjunto de vértices V.
2. Para cada variable x_j, agregamos tres vértices a_j, b_j y c_j a V, formando un triángulo completo (es decir, $\{a_j, b_j, c_j\}$ son adyacentes entre sí).
3. Si x_j aparece en C_i, agregamos una arista entre vc_i y a_j. Si $\neg x_j$ aparece en C_i, agregamos una arista entre vc_i y b_j.
4. Agregamos un vértice r conectado a todos los vértices a_j y b_j ($j = 1, \ldots, n$).
5. Para cada cláusula C_i, conectamos vc_i a un vértice intermedio s_i, y s_i está conectado al vértice r.

Este grafo asegura que las conexiones reflejan la estructura de la fórmula ϕ.



9.2.2 Correctitud de la Reducción

Afirmamos que ϕ es satisfacible si y sólo si el número domático de G es al menos 3.

(⇒) Supongamos que ϕ tiene una asignación satisfactoria. Esto implica que podemos seleccionar un subconjunto de vértices de G correspondiente a los literales verdaderos de ϕ. Dividimos los vértices de G en los siguientes conjuntos:

1. $V_1 = \{a_1, \ldots, a_k, b_{k+1}, \ldots, b_n\} \cup \{r\}$: Este conjunto corresponde a una asignación satisfactoria de las variables $x_1, \ldots, x_n$.
2. $V_2 = \{b_1, \ldots, b_k, a_{k+1}, \ldots, a_n\} \cup \{vc_1, \ldots, vc_m\}$.
3. $V_3 = \{c_1, \ldots, c_n, s_1, \ldots, s_m\}$.

Cada uno de estos conjuntos es dominante en G, y por lo tanto el número domático es al menos 3.

(⇐) Supongamos que el número domático de G es al menos 3. Entonces, podemos dividir los vértices de G en tres conjuntos dominantes V_1, V_2, V_3. Esto implica que cada vértice s_i y cada cláusula vc_i están dominados por al menos uno de estos conjuntos.

(⇐) Supongamos que el número domático de G es al menos 3. Esto implica que podemos dividir los vértices de G en tres conjuntos disjuntos V_1, V_2, V_3, donde cada conjunto es dominante en G.

Sin pérdida de generalidad, asumamos que $r \in V_1$. Entonces, para cada $i \in \{1, \ldots, m\}$ (las cláusulas de la fórmula ϕ), hay solo dos posibilidades para la asignación de vc_i y s_i entre los conjuntos V_2 y V_3, como se describe a continuación:

1. $vc_i \in V_2$ y $s_i \in V_3$: En este caso, el vértice s_i está dominado por $r \in V_1$, y vc_i está dominado por $s_i \in V_3$.
2. $vc_i \in V_3$ y $s_i \in V_2$: Similarmente, s_i está dominado por $r \in V_1$, y vc_i está dominado por $s_i \in V_2$.

En ambos casos, cada cláusula vc_i está dominada por los vértices en su correspondiente conjunto dominante. Esto asegura que cada cláusula de ϕ puede satisfacerse en el grafo.

Ahora consideremos las variables x_j. Para cada $j \in \{1, \ldots, n\}$, recordemos que los vértices a_j, b_j, c_j forman un triángulo completo. Dado que $r \in V_1$, el vértice r domina a a_j y b_j, pero no a c_j. Por lo tanto, uno de los vértices a_j o b_j debe estar en V_2 y el otro en V_3 para dominar completamente a c_j. Esto genera dos posibles asignaciones para cada variable:

1. Si $a_j \in V_2$ y $b_j \in V_3$, esto corresponde a asignar el valor `True` a la variable x_j.
2. Si $a_j \in V_3$ y $b_j \in V_2$, esto corresponde a asignar el valor `False` a la variable x_j.

Dado que cada cláusula vc_i está conectada a los vértices de las variables que la satisfacen, y dado que los conjuntos dominantes aseguran la conectividad entre los vértices, podemos deducir que cada cláusula C_i tiene al menos un literal verdadero. Esto significa que la fórmula ϕ es satisfacible.

9.3 Conclusión

Hemos demostrado que si el número domático de G es al menos 3, entonces la fórmula ϕ es satisfacible. Esto completa la demostración de que el problema **Domatic Number** es NP-Hard.