UNIVERSIDAD
DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicaciones

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Algoritmos para la aproximación de un conjunto a partir de otros. Caracterización matemática del problema y estudio experimental.

Presentado por:
Laura Lázaro Soraluce

Curso académico 2024-2025

Algoritmos para la aproximación de un conjunto a partir de otros. Caracterización matemática del problema y estudio experimental.

Laura Lázaro Soraluce

Laura Lázaro Soraluce *Algoritmos para la aproximación de un conjunto a partir de otros. Caracterización matemática del problema y estudio experimental.*.
Trabajo de fin de Grado. Curso académico 2024-2025.

Responsable de tutorización

Nicolás Marín Ruiz
Ciencias de la Computación e Inteligencia Artificial

Daniel Sánchez Fernández
Ciencias de la Computación e Inteligencia Artificial

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones

Universidad de Granada

Declaración de originalidad

D./Dña. Laura Lázaro Soraluce

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 4 de septiembre de 2025

Fdo: Laura Lázaro Soraluce

Dedicatoria (opcional)
Ver archivo `preliminares/dedicatoria.tex`

Agradecimientos

Agradecimientos (opcional, ver archivo `preliminares/agradecimiento.tex`).

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Resumen

Índice general

Introducción

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.
Ver archivo `preliminares/introduccion.tex`

1. Definición del problema

En este trabajo abordamos el problema de aproximar un conjunto objetivo de elementos partiendo de una colección de subconjuntos disponibles. La idea principal consiste en seleccionar y combinar algunos de estos subconjuntos mediante operaciones sobre conjuntos, respetando una serie de restricciones impuestas por el usuario y optimizando ciertas medidas internas, para posteriormente evaluar los algoritmos con medidas externas comunes. Entre las restricciones podemos incluir un número máximo de operaciones utilizadas o un tamaño mínimo requerido para la solución. En cuanto a las medidas internas, estas pueden estar relacionadas con la sobrecobertura, la infracobertura o la redundancia entre conjuntos, entre otras.

No siempre nos interesa obtener una coincidencia exacta con el conjunto objetivo. En muchos casos puede ser suficiente, o incluso preferible, obtener una aproximación que cumpla con los requisitos. Por ello, formulamos el problema de forma flexible y lo iremos particularizando a lo largo del trabajo, estudiando distintas variantes y aspectos concretos según el contexto.

1.1. Contexto y Motivación

Para ponernos en contexto de nuestro problema, debemos hablar de una variación o particularización suya mucho más antigua: el problema de recubrimiento de conjuntos. Esta surge en el campo de las matemáticas en el siglo XIX, cuando se empieza a formalizar el concepto en teoría de conjuntos, incluyendo nociones como álgebra de conjuntos y anillos de conjuntos, en las que profundizaremos más adelante. En el siglo XX aparece también en el ámbito topológico, cuando se habla de recubrimientos abiertos de espacios topológicos, que guarda una cierta similitud con el concepto de recubrimiento de conjuntos.

Más recientemente, en la década de 1970, tienen lugar grandes avances en el estudio de la complejidad computacional, con el Teorema de Cook y el desarrollo del concepto NP-Completo. Más adelante, la publicación de la lista de 21 problemas NP-Completos de Richard Karp, nos presenta el problema del conjunto de cobertura (Set Cover Problem), popularizando así en el ámbito de la informática el concepto de recubrimiento de conjuntos. Este consiste en: dado un conjunto de elementos, U, y una familia F de subconjuntos de U, ¿se puede encontrar un subconjunto $F' \subseteq F$ de k o menos conjuntos con unión U? Este problema de decisión también tiene una versión de optimización: en las mismas condiciones, minimizar el número de subconjuntos que usa $F' \subseteq F$. [Muno6]

A partir del Set Cover Problem han surgido otras variantes como el Exact Cover, que consiste en: dado un conjunto U y una familia F de subconjuntos de U, encontrar un subconjunto $F' \subseteq F$ donde cada elemento de U aparece en un y sólo un subconjuntos de F'. Otra variante es el Exact Cover By 3-Sets (X3C), que consiste en: dado un conjunto U, con cardinal múltiplo de tres, y una familia F de subconjuntos de U de tres elementos, encontrar un subconjunto

$F' \subseteq F$ donde cada elemento de U aparece en un y sólo un elemento de F'. [LP14]

Aunque el problema que abordamos en este trabajo está estrechamente relacionado con estas formulaciones clásicas, es importante recalcar que no se trata exactamente del mismo problema. Como veremos más a fondo, en nuestro caso permitimos también el uso de otras operaciones a parte de la unión, con el objetivo de aproximar G lo mejor posible según medidas específicas, que no siempre se traduce en hallar una partición suya o una cobertura exacta.

1.2. Definición general del problema

Una vez que conocemos el contexto histórico, el origen y la importancia del problema que nos ocupa, nos centramos en formularlo de forma general. Queremos definir un marco amplio que englobe todas las variantes posibles que analizaremos a lo largo del trabajo.

A continuación, enumeramos los elementos principales de nuestro problema:

1. **Universo** U: conjunto finito de elementos.
2. **Familia de subconjuntos** $F \subseteq \mathcal{P}(U)$: subconjuntos de U que podemos usar para construir la aproximación.
3. **Conjunto objetivo** $G \subseteq U$: conjunto que queremos aproximar utilizando combinaciones de elementos de F.
4. **Operaciones permitidas**: operaciones de conjuntos correctamente definidas sobre los elementos de U, que se permiten para construir las expresiones.
5. **Espacio de expresiones** $\mathcal{E}$: todas las expresiones formadas combinando elementos de F mediante las operaciones permitidas. Se permite anidar operaciones y repetir subconjuntos.
6. **Conjuntos resultantes** $\mathcal{R}$: todos los conjuntos resultado que se obtienen de evaluar las expresiones del espacio $\mathcal{E}$. Estos conjuntos resultantes pueden verse como la imagen por una función $eval : \mathcal{E} \to \mathcal{R}$ que a cada expresión le asigna su evaluación.
7. **Espacio de soluciones válidas** $\mathcal{S}$: subconjunto de $\mathcal{R}$ que cumplen todas las restricciones del problema. Sólo los elementos de $\mathcal{S}$ se consideran soluciones válidas. Será o no igual a $\mathcal{R}$ dependiendo de las restricciones concretas que tenga la variante.
8. **Medidas de evaluación**: funciones que asignan un valor numérico a una expresión o a una solución, permitiendo guiar el proceso de construcción o evaluar su calidad final. Podemos clasificar estas medidas según el tipo de propiedad que evalúan. Según el algoritmo utilizado, una misma medida puede usarse durante la construcción de la solución o después, para comparar y seleccionar soluciones. Estos son algunos ejemplos de medidas, clasificadas según el tipo de propiedad que evalúan, que estudiaremos más adelante:
   - Medidas de asociación
   - Medidas direccionales

- Medidas de cobertura
- Medidas de redundancia
- Medidas de ruido
- Medidas de error global
- Medidas de tamaño
- Medidas ponderadas

9. **Restricciones**: condiciones adicionales que podemos imponer sobre los resultados para que sean válidos, el espacio de expresiones o sobre las operaciones permitidas. Estas definen el subconjunto de expresiones o soluciones que se consideran admisibles, reduciendo así el espacio de expresiones o soluciones. Pueden ser restricciones del usuario, que dependen del problema concreto o la aplicación práctica y se imponen antes de ejecutar cualquier algoritmo; o restricciones del propio algoritmo, que surgen de cómo funciona el algoritmo elegido. Por ejemplo, límite de subconjuntos utilizados, operaciones o tamaño máximo del conjunto aproximado.

Ahora que hemos nombrado los elementos principales de nuestro problema, podemos dar una definición formal que resulte un poco más intuitiva: dados un conjunto universo U y una colección F de subconjuntos de U, queremos encontrar la mejor aproximación de un subconjunto objetivo $G \subseteq U$ utilizando diferentes operaciones sobre elementos de F.

Como ya hemos definido, cada combinación de subconjuntos y operaciones nos da una expresión dentro de un espacio de expresiones, $\mathcal{E}$, y al evaluarlas obtenemos el espacio de conjuntos resultantes, $\mathcal{R}$. Es importante tener en cuenta que, debido a propiedades del Álgebra de Boole, distintas expresiones pueden dar lugar al mismo resultado ($eval(e_m) = eval(e_n) = r_i \in \mathcal{R}; \quad m, n \in \mathbb{N}, \quad e_m, e_n \in \mathcal{E}; \quad e_m \neq e_n$), por lo que no podemos asegurar que la aplicación *eval* sea inyectiva [Unia].

En el proceso de construcción de la aproximación, algunos algoritmos como el Greedy hacen uso de una medida evaluadora interna para seleccionar en cada paso el par {Operación, Subconjunto} más prometedor. En otros casos, como la Búsqueda Exhaustiva, se generan múltiples soluciones sin una guía local, y se selecciona la mejor según una medida evaluadora externa. Estas nos permiten valorar la calidad de la solución obtenida por cualquier algoritmo y compararlas entre sí.

Planteamos el siguiente ejemplo sencillo al que haremos referencia a lo largo del trabajo, para mostrar diferentes conceptos de forma intuitiva.

Ejemplo 1.1. Consideramos los siguientes elementos:

- $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- $F = \{\{1, 2\}, \{2, 3, 4\}, \{2, 5\}, \{4, 7\}, \{4, 5, 6\}, \{6, 7, 8\}, \{7, 10\}, \{8, 9\}, \{10\}\}$
- $G = \{2, 4, 5, 7, 10\}$
- $\mathcal{E}$: es el conjunto de todas las expresiones válidas que se pueden formar combinando subconjuntos de F (con posible repetición) mediante operaciones permitidas como la unión, intersección, diferencia... Por ejemplo: $\{1, 2\} \cup \{2, 3, 4\}$ o $\{2, 5\} \setminus \{8, 9\}$.

- $\mathcal{R}$: es el conjunto de todos los subconjuntos de U que se pueden obtener evaluando alguna expresión de $\mathcal{E}$. Por ejemplo, de las dos expresiones anteriores: $\{1,2,3,4\}$ y $\{2,5\}$.

1.1
Con este ejemplo vemos que, mientras que $\mathcal{E}$ describe cómo se construyen combinaciones de los conjuntos de F y potenciales soluciones, $\mathcal{R}$ describe los conjuntos finales que se obtienen al evaluarlas. Como no imponemos restricciones sobre la repetición de subconjuntos ni sobre el tipo de operaciones permitidas, el tamaño de $\mathcal{E}$ en este ejemplo es infinito. El tamaño de $\mathcal{R}$, sin embargo, está superiormente acotado por $\mathcal{P}(U)$ como veremos al presentar el concepto de conjunto potencia.

Es conveniente aclarar que, además de estos elementos principales de definen el problema general, al introducir restricciones específicas y formular nuevas variantes más concretas, surge la necesidad de añadir elementos derivados, que presentaremos en la siguiente sección, y que facilitarán la comprensión.

1.3. Variantes del problema

A lo largo del trabajo, consideramos distintas variantes del problema general que acabamos de formular. Estas consisten en aplicar restricciones específicas sobre alguno de los elementos del problema, comúnmente el espacio de expresiones, el uso de subconjuntos o las operaciones aplicables. Cada variante define un subconjunto del espacio general de soluciones, y permite adaptar el problema a estrategias de resolución prácticas que no podríamos aplicar sobre el problema general.

1.3.1. Ejemplos de variantes

Por un lado tenemos las variantes algorítmicas que surgen al imponer restricciones sobre el proceso de construcción de soluciones, es decir, sobre la manera en la que se explora el espacio de expresiones $\mathcal{E}$.

- **Variante búsqueda exhaustiva con profundidad** k: Consideramos todas las expresiones posibles de hasta k operaciones sobre elementos de F, construidas libremente, y permitiendo repetición de conjuntos. Las operaciones permitidas son: unión, intersección y diferencia. Esta variante representa el espacio completo explorado por una búsqueda exhaustiva acotada por profundidad, que veremos más adelante.

- **Variante Greedy clásico**: Limitamos las expresiones en $\mathcal{E}$ a aquellas que se pueden construir añadiendo iterativamente un par {Operación, Conjunto}. Permitimos la repetición de elementos de F, e imponemos un límite superior de k operaciones. Esto lo hacemos porque, como veremos más adelante, en el algoritmo Greedy clásico, la solución se construye de forma secuencial a partir de una aproximación inicial vacía. En cada iteración, aplicamos una operación entre el conjunto construido hasta el momento y un subconjunto F_i que el algoritmo selecciona según una medida evaluadora

interna. El orden en el que aplicamos las operaciones es fijo y secuencial, de la forma: $(((A \text{ op } B) \text{ op } C) \text{ op } D) \ldots$.

Por otro lado las tenemos variantes estructurales, que se definen al restringir las operaciones permitidas o las condiciones que deben cumplir las soluciones.

- **Variante sin repetición**: El problema es esencialmente el mismo que el general, limitando el las expresiones de $\mathcal{E}$ a aquellas en las que cada subconjunto de F puede aparecer como mucho una vez en cada expresión. Es decir, no permitimos repetición de elementos de F.

- **Variante Recubrimiento de** G: La unica operación permitida es la unión. Se impone como restricción que la solución debe ser recubrimiento de G, definición que veremos más adelante.

- **Variante Set Cover clásico**: Partiendo de la variante anterior, añadimos la condición de que el recubrimiento debe ser exacto. Además, el objetivo pasa a ser minimizar el número de subconjuntos seleccionados para la solución. Por tanto, la medida evaluadora que conviene usar es el número de subconjuntos usados. Esta es la formulación clásica del Set Cover problem.

- **Variante Exact Cover**: Esta variante introduce una nueva restricción adicional sobre la solución: los subconjuntos utilizados deben ser disjuntos dos a dos. Así, no sólo buscamos cubrir G exactamente, sino hacerlo mediante una partición. La única operación permitida sigue siendo la unión.

- **Variante sin conjunto vacío en** F: En el planteamiento general del problema permitimos que el conjunto vacío pertenezca a la familia F, lo cual es necesario para poder trabajar con determinadas estructuras algebraicas (como anillos o álgebras de conjuntos). Sin embargo, en un contexto práctico, puede ser útil excluir $\emptyset$ de F, ya que no contribuye a mejorar la aproximación al conjunto objetivo G y puede generar soluciones triviales. Esta variante tiene una restricción, en la que se elimina del espacio de búsqueda el conjunto vacío como subconjunto disponible para construir expresiones.

- **Variante Semianillo**: las únicas operaciones permitidas son la intersección y la diferencia.

- **Variante Anillo**: las únicas operaciones permitidas son la unión y la diferencia.

- **Variante Álgebra de Conjuntos**: las únicas operaciones permitidas son la unión y el complemento con respecto a U.

1.3.2. Elementos derivados

De la definición de las distintas variantes, surge la necesidad de introducir elementos adicionales a la definición de nuestro problema:

- **Espacio de expresiones restringido** $\mathcal{E}'$: surge de restricciones sintácticas, es decir, sobre las expresiones permitidas. Por ejemplo:
  - Limitar el número de operaciones que pueden aparecer en una expresión, como en la variante de búsqueda exhaustiva con profundidad k.
  - Restringir la forma en la que se construyen las expresiones, como en la variante Greedy clásico, donde sólo permitimos expresiones secuenciales de la forma $((A \text{ op } B) \text{ op } C) \text{ op } \ldots$.
  - No permitir la repetición de un mismo subconjunto $F_i \in F$ más de una vez en cada expresión, como en la variante sin repetición.
- **Conjuntos resultantes restringidos** $\mathcal{R}' = eval(\mathcal{E}')$: surgen como consecuencia de $\mathcal{E}'$. Por ejemplo, si $\mathcal{E}'$ sólo admite uniones, entonces $\mathcal{R}'$ contendrá únicamente conjuntos que puedan obtenerse mediante uniones de elementos de F.
- **Espacio de soluciones válidas** $\mathcal{S} \subseteq \mathcal{R}$: surge de restricciones sobre los resultados para que se consideren soluciones. Por ejemplo:
  - Exigir que los subconjuntos utilizados sean disjuntos, como en la variante de Exact Cover. Esta restricción se puede entender también como restricción sintáctica sobre $\mathcal{E}$, pero en la literatura sobre Set Partition y Set Cover se formula habitualmente como restricción sobre las soluciones [KV12].
  - Requerir que la solución cubra al menos un porcentaje x % de los elementos de G, como en la variante de recubrimiento de G, donde se exige cubrir el 100 %.

  Aclaramos que no siempre coincide con $\mathcal{R}$, ya que en $\mathcal{R}$ aparecen también los conjuntos resultantes de evaluaciones intermedias que todavía no constituyen soluciones válidas. Sólo aquellos que cumplen todas las restricciones forman parte de $\mathcal{S}$.

1.4. Aplicaciones e impacto

El problema de aproximación de conjuntos tiene aplicaciones en distintas áreas, especialmente en la toma de decisiones y la optimización de recursos, aplicada también a recubrimiento de áreas físicas. Veamos algunos ejemplos relevantes:

- En marketing: colocar anuncios de forma que cubran un público objetivo, sin redundancias (sin que las personas lo vean varias veces) y sin desperdiciar recursos (sin que lo vean personas a las que no va dirigido).
- En logística: minimizar el número de camiones utilizados en la distribución de mercancías [MMW24]. Tendríamos un conjunto de todas las ubicaciones de entrega, un conjunto de ubicaciones que pueden ser cubiertas por cada camión y un conjunto de ubicaciones a cubrir. En este caso añadiríamos una restricción adicional de escoger el menor número posible de camiones, es decir, minimizar el número de subconjuntos escogidos para cubrir.

- En problemas de clasificación en informática: sacar un conjunto representativo de las características para construir un modelo eficiente en términos de predicción y de complejidad. Analizamos un caso concreto de una práctica de la asignatura de Inteligencia de Negocio del 4º curso del grado en Ingeniería Informática, que hemos cursado recientemente: predicción de aprobación de créditos. Tendríamos un conjunto de todas las instancias, todos los solicitantes de crédito con sus características; un subconjunto de características que los modelos usan para predecir la aprobación de crédito; y un conjunto de solicitantes con una etiqueta determinada (aprobado/rechazado).
- En redes de comunicación: colocar antenas o puntos de acceso para asegurar cobertura en todo el área [RTF76]. Tendríamos un conjunto con todos los posibles puntos de ubicación en un área determinada, un conjunto de áreas de cobertura de cada antena, y un conjunto de ubicaciones específicas que queremos cubrir con la red.
- En biología computacional: seleccionar un conjunto de genes que sea representativo de un grupo de pacientes específico (sano/enfermo). Tendríamos un conjunto con todos los genes presentes en el conjunto de muestras biológicas, un conjunto de genes que están activos cuando se tiene una cierta enfermedad, y un conjunto de genes esenciales para diferenciar entre grupos de interés.

Estas aplicaciones muestran la importancia del problema de aproximación de conjuntos y sus distintas variantes, en la toma de decisiones eficientes y la optimización de recursos, lo que lo convierte en un problema de gran interés no sólo en el ámbito computacional.

1.5. Objetivos del trabajo y metodología

Este trabajo está dividido en dos bloques principales: uno más teórico, matemático; y otro más práctico, computacional. En la primera parte, haremos un estudio matemático exhaustivo del problema de aproximación de conjuntos, que hemos formalizado de forma rigurosa.

Para este análisis, nos apoyamos en diferentes áreas de las matemáticas, como el álgebra de conjuntos y las estructuras algebraicas del álgebra; la teoría de la medida del análisis; las medidas de asociación y tablas de contingencia de la estadística; el cálculo del número de expresiones y de particiones desde la combinatoria; y la NP-completitud y el tamaño del espacio de búsqueda desde la teoría de la complejidad computacional.

La idea es dar un contexto histórico y teórico general de dichas áreas, profundizando sólo en los conceptos que son directamente relevantes en nuestro problema. En este sentido, se trata más bien de un estudio transversal por distintas ramas matemáticas, en lugar de un análisis en profundidad de una sola de ellas. Este enfoque nos permite conectar distintos conceptos trabajados a lo largo del Grado en Matemáticas, integrándolos en un problema práctico.

En la segunda parte, vamos a abordar el problema desde un punto de vista práctico, en el que implementaremos y estudiaremos el comportamiento de varios algoritmos sobre distintas variantes ya definidas del problema. Utilizaremos en cada caso las medidas adecuadas correspondientes para guiar el proceso de construcción de la solución, y asegurar que se cumplen las restricciones teóricas impuestas. En particular, los algoritmos que incluiremos serán:

búsqueda exhaustiva con profundidad limitada, que explora secuencialmente el espacio de soluciones completo; y Greedy, que construye una solución de forma iterativa tomando la mejor decisión basada en una medida evaluadora interna.

Finalmente, evaluaremos los resultados utilizando medidas para comparar de forma objetiva la calidad de las soluciones obtenidas, y analizar el rendimiento de cada algoritmo en este problema.

2. Fundamentos matemáticos

2.1. Convenciones y notación

A lo largo de este trabajo, utilizaremos las siguientes convenciones de notación:

- U: conjunto universo sobre el que trabajamos, de tamaño $n = |U|$.
- F: familia finita de subconjuntos de U, de tamaño $m = |F|$.
- $F_1, F_2, ..., F_m$: elementos de F. Cada F_i tiene m_i elementos.
- $G \subseteq U$: conjunto objetivo que queremos aproximar, de tamaño $g = |G|$.
- $\tilde{G} \in \mathcal{S} \subseteq \mathcal{R}$: mejor aproximación final de G, obtenida mediante una combinación de subconjuntos de F, con $\tilde{g} = |\tilde{G}|$.
- $\tilde{G}_i \in \mathcal{R}$: aproximación parcial de G tras la i-ésima iteración de un algoritmo (no necesariamente válida).
- $\mathcal{E}$: espacio de expresiones construidas a partir de subconjuntos de F y operaciones permitidas.
- $\mathcal{R}$: familia de conjuntos resultantes que se obtienen evaluando expresiones de $\mathcal{E}$. Cada elemento $R_i = eval(e_i) \in \mathcal{R}$ es el resultado de evaluar una expresión $e_i \in \mathcal{E}$. Es decir, cada R_i es un único conjunto de elementos donde el orden no importa. Cada $r_j \in R_i \in \mathcal{R}$ representa un elemento de un conjunto de $\mathcal{R}$.
- $\mathcal{S} \subseteq \mathcal{R}$: espacio de soluciones válidas, formado por aquellos conjuntos $s_i = r_i \in \mathcal{R}$ que cumplen las restricciones del problema.
- $F' \subseteq F$: familia de subconjuntos de F utilizados en la expresión que genera la aproximación final $\tilde{G}$, de tamaño $m' = |F'|$.

Salvo que se indique lo contrario, mantendremos estas notaciones a lo largo del trabajo para evitar ambigüedades y facilitar la lectura.

Convenimos sin pérdida de generalidad que $U \neq \emptyset$, $F \neq \emptyset$ ($m \geq 1$), y que $G \neq \emptyset$, pues en dichos casos el problema no tendría interés práctico o teórico.

2.2. Estructuras algebraicas

Nuestro problema puede abordarse desde distintas áreas de las matemáticas, cada una aportando perspectivas e información diferente. En primer lugar, nos enfocamos en el álgebra y en las estructuras algebricas que aparecen en nuestro problema, analizando tanto sus propiedades como la interacción entre ellas y las posibles estructuras derivadas. Más adelante

introduciremos medidas que nos permitirán comparar subconjuntos y asignarles valores en función de ciertas propiedades. Por último, una vez entendido y formalizado el marco del problema que nos ocupa, estudiaremos su complejidad teórica.

Históricamente, la teoría de conjuntos comienza a desarrollarse a finales del siglo XIX con los trabajos de George Cantor en la rama del análisis, donde introdujo los cardinales infinitos entre otros conceptos. En esos momentos existían muchas paradojas alrededor de dicha teoría, como la de Russell, que motivaron la búsqueda de axiomas más rigurosos. Gottlob Frege propuso una primera formalización en 1903, pero Russell evidenció sus inconsistencias. En 1908, Ernst Zermelo propone una axiomatización que termina Abraham Fraenkel en 1922, dando lugar a la teoría de conjuntos moderna. Además, a Zermelo se le atribute el conocido axioma de elección. Hoy en día, la mayor parte del álgebra se sigue basando dicha teoría de conjuntos. Aunque no se puede demostrar si la teoría de conjuntos moderna es consistente de acuerdo con el segundo teorema de incompletitud de Gödel, no se han encontrado contradicciones en el último siglo, y por tanto se sigue basando la mayor parte de álgebra en ella [Set20].

2.2.1. Conjunto potencia

Esta teoría de conjuntos moderna y su formulación axiomática, nos da el marco formal que necesitamos para trabajar con estructuras y operaciones que conocemos hoy en día sobre bases sólidas. Cuando trabajamos con subconjuntos en el contexto de estructuras algebraicas aparecen conceptos básicos que es fundamental introducir. Es el caso del conjunto potencia, que en nuestro problema es especialmente relevante por distintas razones. Por un lado, nos permite comprender cómo está $\mathcal{R}$ acotado superiormente. Esto se debe a que cualquier operación entre subconjuntos de U siempre produce otro subconjunto de U. Es decir, no se generan elementos ajenos a U, y por tanto, ningún conjunto fuera de $\mathcal{P}(U)$. Por otro lado, el conjunto potencia es la base de estructuras matemáticas más complejas que nos permiten añadir propiedades a nuestro problema, así como enriquecer su análisis.

Definición 2.1. [Cru22] El conjunto potencia de U, denotado $\mathcal{P}(U)$, es el conjunto cuyos elementos son todos los subconjuntos de U. Se escribe

$$\mathcal{P}(U) = \{X | X \subseteq U\}$$

Es interesante conocer este concepto fundamental, pues para cualquier conjunto F que consideremos, siempre se va a dar $F \subseteq \mathcal{P}(U)$. Esto nos va a permitir utilizar algunas definiciones que veremos más adelante.

El conjunto potencia de un conjunto de cardinal n contiene 2^n conjuntos [Lip98]. Utilizando el ejemplo 1.1, el conjunto potencia de U, $\mathcal{P}(U)$, está formado por los $2^{10} = 1024$ subconjuntos posibles de U, entre los que están:

- el conjunto vacío: $\emptyset$
- todos los conjuntos unitarios: $\{1\}, \ldots, \{10\}$
- todas las combinaciones de hasta 9 elementos: $\{1,2\}, \{2,5,7\}, \ldots$

- el conjunto total: U

En este caso, la familia de conjuntos resultantes $\mathcal{R}$ tendrá a lo sumo 1024 elementos. Esto implica que, por muchas expresiones distintas que se construyan en $\mathcal{E}$, al evaluarlas nunca obtendremos más de 1024 conjuntos diferentes. Esto es de gran utilidad en nuestro problema, pues nos permite detener el proceso de construcción de expresiones una vez alcanzado el número máximo de conjuntos resultantes posibles.

2.2.2. Retículo y álgebra de Boole

Una vez que hemos presentado el conjunto $\mathcal{P}(U)$, sobre el que trabajaremos, es natural introducir el concepto de retículo. Antes de hacerlo, vemos algunos conceptos subyacentes como el conjunto ordenado:

Definición 2.2 (Relación de orden)**.** [Mir] Una operación binaria $\leq$ es una relación de orden si es asociativa, antisimétrica y transitiva.

En nuestro caso, $\subseteq$ es una relación de orden sobre $\mathcal{P}(U)$, pues cumple:

- Reflexividad: para todo $A \in \mathcal{P}(U)$ se tiene $A \subseteq A$.
- Antisimetría: si $A \subseteq B$ y $B \subseteq A$, entonces $A = B$.
- Transitividad: si $A \subseteq B$ y $B \subseteq C$, entonces $A \subseteq C$.

Como tenemos una relación de orden sobre $\mathcal{P}(U)$, decimos que $(\mathcal{P}(U), \subseteq)$ es un conjunto ordenado. Además, como la inclusión sólo puede ir en un sentido, estamos ante un conjunto totalmente ordenado.

Para representar cualquier conjunto ordenado, se puede utilizar un diagrama de Hasse. En nuestro caso, como en el ejemplo base 1.1, $\mathcal{P}(U)$ tiene demasiados elementos lo que dificulta la visualización, lo mostramos para un nuevo ejemplo:

Ejemplo 2.3. $U = \{1, 2, 3, 4\}$

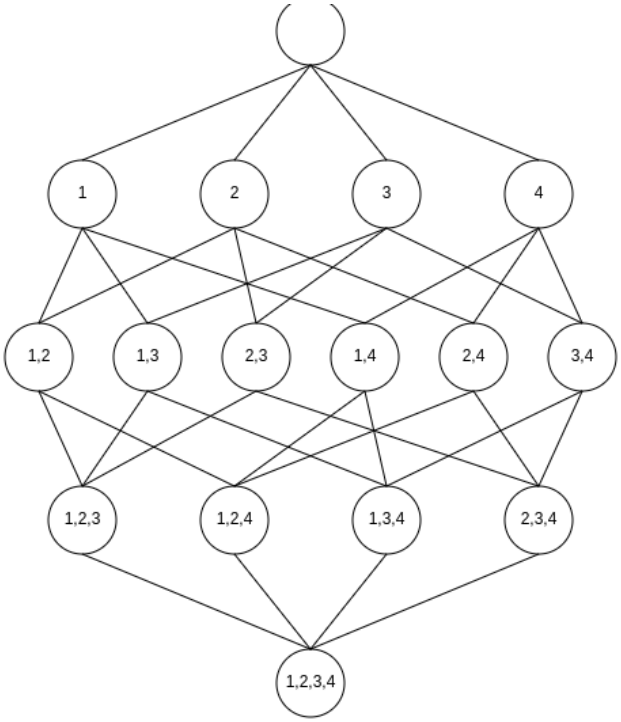

Figura 2.1.: Diagrama de Hasse del conjunto potencia de $U = \{1, 2, 3, 4\}$. Elaborada con la herramienta online "Draw the Hasse Diagram of a Powerset", disponible en: https://www.philippe-fournier-viger.com/tools/draw_hasse.php

Por último, antes de dar la definición formal de retículo, recordamos que las cotas superiores de un subconjunto de $\mathcal{P}(U)$ son todos aquellos elementos de $\mathcal{P}(U)$ que lo contienen, y que el supremo es la mínima de dichas cotas superiores. De manera análoga, las cotas inferiores son los subconjuntos contenidos en todos los elementos del conjunto dado, y el ínfimo es la máxima de ellas.

Definición 2.4 (Retículo)**.** [Mir] Un retículo es un conjunto ordenado, $(\mathcal{P}(U), \subseteq)$ en el que cualquier familia finita de subconjuntos tiene supremo e ínfimo.

En nuestro caso, para cualquier familia finita de subconjuntos de $\mathcal{P}(U)$, $\mathcal{Y}$, se cumple que:

$$\sup(\mathcal{Y}) = \bigcup_{Y \in \mathcal{Y}} Y, \qquad \inf(\mathcal{Y}) = \bigcap_{Y \in \mathcal{Y}} Y.$$

Es decir, la unión actúa como supremo y la intersección como ínfimo en $\mathcal{P}(U)$. Esto es fácil de ver en un diagrama de Hasse. Para encontrar el supremo de una familia de conjuntos basta con bajar en el diagrama hasta encontrar el menor conjunto que contiene a todos ellos. De la misma manera, para determinar el ínfimo, ascendemos en el diagrama hasta encontrar el mayor conjunto que está contenido en todos los considerados. Siguiendo con nuestro ejemplo 2.3 y utiliziando nuestro diagrama 2.1, si consideramos $\mathcal{Y} = \{\{3\}, \{1, 2\}\}$, siguiendo hacia abajo, encontramos el conjunto $\{1, 2, 3\}$, que es el menor conjunto que contiene a todos

los de $\mathcal{Y}$. Este será el supremo. Para el ínfimo, vemos que el único conjunto contenido en ambos de $\mathcal{Y}$ es $\emptyset$.

Por lo tanto, concluimos que cualquier familia de subconjuntos de $\mathcal{P}(U)$ siempre va a tener ínfimo y supremo, que como mucho serán el conjunto vacío y el total respectivamente, probando así que $\mathcal{P}(U)$ es un retículo.

Si además un retículo es distributivo y todos sus elementos tienen complemento, decimos que se trata de un Álgebra de Boole [Mir]. Este es justo el caso de $\mathcal{P}(U)$, pues la unión y la intersección son distributivas la una con respecto de la otra; y para todo $A \in \mathcal{P}(U)$, se cumple que $A \cup U \setminus A = U$ y $A \cap (U \setminus A) = \emptyset$.

2.2.3. Relación con la lógica proposicional

El hecho de que $\mathcal{P}(U)$ sea álgebra de Boole, nos permite conectar directamente con la lógica proposicional, pues a cada elemento de $\mathcal{P}(U)$, es decir, a cada subconjunto de U, se le puede asociar una proposición lógica. Además, las operaciones de unión, intersección y complemento corresponden a la disyunción, la conjunción y la negación, respectivamente [Unic].

En la siguiente tabla mostramos a qué operación lógica corresponde cada operación de conjuntos:

Operación en conjuntos	**Símbolo**	**Operación lógica**
Unión	$A \cup B$	Disyunción ($p \vee q$)
Intersección	$A \cap B$	Conjunción ($p \wedge q$)
Complemento	$U \setminus A$	Negación ($\neg p$)
Conjunto vacío	$\emptyset$	Falso ($\bot$)
Conjunto total	U	Verdadero ($\top$)

Tabla 2.1.: Correspondencia entre álgebra de Boole y lógica proposicional.

Por ejemplo, cada subconjunto de U corresponde a una proposición lógica p_A que indica si un elemento u pertenece o no a A. Para el ejemplo 2.3, el subconjunto $A = \{1,3\}$, se asocia con la proposición p_A, que vale 0 si $u \notin \{1,3\}$ y 1 en caso contrario. Sea $B = \{2,3\}$ otro subconjunto de U, tenemos que

$$A \cup B = \{1,2,3\}, \qquad p_{A \cup B}(u) = p_A(u) \vee p_B(u).$$

Con la intersección:

$$A \cap B = \{3\}, \qquad p_{A \cap B}(u) = p_A(u) \wedge p_B(u).$$

Y con el complemento:

$$U \setminus A = \{2,4\}, \qquad p_{U \setminus A}(u) = \neg p_A(u).$$

Así, tenemos un marco equivalente a la lógica proposicional, de manera que trabajar con familias de subconjuntos equivale a razonar con fórmulas proposicionales. Esto hace que

nuestro problema esté estrechamente relacionado con la lógica matemática y la teoría de la computación.

2.2.4. Particiones y recubrimientos

Sobre este marco algebraico, podemos definir objetivos concretos de algunas variantes como la de Recubrimiento de G, de Set Cover clásico, y de Exact Cover, en las que intervienen varios conceptos matemáticos no mencionados hasta ahora. En la variante de Exact Cover en concreto, ser una partición del conjunto objetivo es un requisito para ser solución.

Definición 2.5 (Partición)**.** [Lip98] Dada una familia $F' = F'_1, \ldots, F'_{m'}$ de subconjuntos de F, F' es una partición de G si:

- $\bigcup_{i \in \{1,\ldots,m'\}} F'_i = G$,
- $\forall F'_i, F'_j$, o bien $F'_i = F'_j$, o bien $F'_i \cap F'_j = \emptyset$

Dicho de otra forma, una partición descompone G en subconjuntos disjuntos que lo recubren completamente, sin que haya elementos sin cubrir.

Queremos que los $F'_1, \ldots, F'_{m'}$ cumplan las siguientes condiciones:

- Forman un recubrimiento exacto de G:

$$G = \bigcup_{i=1}^{m'} F'_i$$

- Son disjuntos dos a dos:

$$F'_i \cap F'_j = \emptyset, \quad \forall i \neq j$$

- No hay redundancia, es decir, no podemos eliminar ningún F'_i sin dejar de cubrir G.

El concepto de recubrimiento exacto de G que vemos en esta definición de partición, aparece también en la variante del Set Cover clásico, donde es un requisito para constituir una solución.

Sin embargo, esto no siempre va a ser de interés, necesario o incluso posible, pues puede que haya elementos de G que no quedan cubiertos por ningún elemento de F, que algunos de los subconjuntos de F escogidos se solapen entre sí, o bien que se tengan en cuenta elementos que se salen del conjunto G.

Además, el número de particiones posibles para un conjunto finito depende de su cardinal, n, y viene dado por el número de Bell [Alo]:

- $B(0) = 1$
- $B(n) = \sum_{k=0}^{n-1} \binom{n-1}{k} B(k)$

Es claro que este número crece rápidamente con n, lo cuál nos da una idea de la complejidad del problema de trabajar con particiones incluso en casos sencillos, y justifica la utilidad y necesidad de otras variantes y criterios. Por ello, en este trabajo nos centramos en encontrar la mejor aproximación posible priorizando diferentes objetivos en lugar de limitándonos a sólo uno.

En muchos casos, como en la variante de Recubrimiento de G, puede ser más útil y eficiente encontrar un recubrimiento de G aunque haya solapamientos, condición más relajada que la partición.

Definición 2.6 (Recubrimiento)**.** [Llu16] La familia $F' \neq \{\emptyset\}$ es un recubrimiento del conjunto G si su unión contiene a G, es decir, si $G \subset \bigcup_{i \in \{1,\ldots,m'\}} F'_i$.

Volviendo a nuestro Ejemplo 1.1, donde $G = \{2, 4, 5, 7, 10\}$, podemos encontrar:

- Recubrimieto de G: $F'_1 = \{\{2, 3, 4\}, \{4, 5, 6\}, \{6, 7, 8\}, \{10\}\}$
- Recubrimiento exacto de G: $F'_2 = \{\{2, 5\}, \{4, 7\}, \{7, 10\}\}$
- Partición exacta de G: $F'_3 = \{\{2, 5\}, \{4, 7\}, \{10\}\}$

2.2.5. Estructuras inducidas

En el ámbito del álgebra, una estructura surge cuando dotamos a los conjuntos de ciertas leyes de composición [Pal], y por lo tanto se definen por:

- Un conjunto subyacente.
- Una o varias leyes de composición definidas sobre el conjunto.
- Unos axiomas que deben verificar dichas leyes y explican cómo se comportan los elementos entre ellos.

En nuestro caso, a partir de F, podemos considerar varias estructuras algebraicas de conjuntos que surgen de cerrar F bajo distintas operaciones. Estas estructuras generadas por F definen subconjuntos de $\mathcal{P}(U)$, y por tanto también subconjuntos de $\mathcal{R}$, la familia de conjuntos resultantes de evaluar las expresiones en $\mathcal{E}$. Para la construcción de estas estructuras, hacemos todas las posibles combinaciones de elementos de F mediante las operaciones permitidas en cada caso, incluyendo también combinaciones previas.

Si consideramos la variante en la que sólo permitimos las operaciones de unión y diferencia entre subconjuntos de F, llegamos naturalmente a la estructura algebraica conocida como anillo de conjuntos:

Definición 2.7 (Anillo)**.** [Hal74] Un anillo de conjuntos es una colección no vacía $\mathcal{R}$ de conjuntos, tal que si $R_i, R_j \in \mathcal{R}$ para $i, j \in I = \{1, \ldots, \#\mathcal{R}\}$, entonces $R_i \cup R_j \in \mathcal{R}$ y $R_i \setminus R_j \in \mathcal{R}$.

En particular, como en nuestro caso $\mathcal{R}$ es finito, deducimos que si $R_i, R_j \in \mathcal{R}$, entonces $\bigcup_{i \in I} R_i \in \mathcal{R}$ y $\bigcap_{i \in I} R_i \in \mathcal{R}$.

Teniendo esto en cuenta, el conjunto vacío y el conjunto $\mathcal{P}(U)$ también son ejemplos de anillos de conjuntos.

Ejemplo sencillo:

- $U = \{1, 2, 3, 4\}$
- $F = \{\{1\}, \{1, 2\}, \{3, 4\}\}$
- Operaciones permitidas: unión y diferencia

El espacio $\mathcal{E}$ de expresiones que se genera es: $\mathcal{E} = \{\{1\}, \{1, 2\}, \{3, 4\}, \{1\} \cup \{1, 2\}, \{1\} \cup \{3, 4\}, \{1, 2\} \cup \{3, 4\}, \{1\} \setminus \{3, 4\}, \{1\} \setminus \{1, 2\}, \{1, 2\} \setminus \{3, 4\}, \dots\}$

En este caso, $\mathcal{E}$ tiene infinitos elementos, pues no hemos impuesto restricciones sobre el número máximo de operaciones o elementos de F que podemos utilizar.

La familia de conjuntos resultantes que se genera es:

$$\mathcal{R} = \{\emptyset, \{1\}, \{2\}, \{1, 2\}, \{3, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$$

Esto es, el anillo de conjuntos generado por F.

Si consideramos ahora la variante en la que sólo permitimos las operaciones de intersección y diferencia entre subconjuntos de F, llegamos naturalmente a la estructura algebraica conocida como subanillo de conjuntos:

Definición 2.8 (Semianillo)**.** [Hal74] Un semianillo es una colección no vacía $\mathcal{R}$ de conjuntos tal que:

- si $R_i, R_j \in \mathcal{R}$ entonces $R_i \cap R_j \in \mathcal{R}$
- si $R_i, R_j \in \mathcal{R}$ y $R_i \subset R_j$, entonces $\exists\{C_0, \dots, C_n\}$ una colección finita de conjuntos de $\mathcal{R}$, tal que $R_i = C_0 \subset C_1 \subset \cdots \subset C_n = R_j$ y $D_i = C_i \setminus C_{i-1} \in \mathcal{R} \quad \forall i \in I = \#\mathcal{R}$.

De nuevo, todos los semianillos siempre contienen al conjunto vacío.

Ejemplo sencillo:

- $U = \{1, 2, 3, 4\}$
- $F = \{\{1\}, \{2, 3\}, \{1, 2, 3\}\}$
- Operaciones permitidas: intersección y diferencia.

La familia de conjuntos resultantes que se genera es:

$$\mathcal{R} = \{\emptyset, \{1\}, \{2, 3\}\}$$

Esto es, el subanillo generado por F.

Si pasamos a considerar la variante en la que permitimos las operaciones de unión, y complemento con respecto a U, llegamos al concepto de álgebra de conjuntos:

Definición 2.9 (Álgebra de conjuntos)**.** [Hal74] Un álgebra de conjuntos (también a veces conocido como campo de conjuntos) es una colección no vacía $\mathcal{R}$ de conjuntos tal que:

- si $R_i \in \mathcal{R}$ y $R_j \in \mathcal{R}$, entonces $R_i \cup R_j \in \mathcal{R}$
- si $R_i \in \mathcal{R}$, entonces $\overline{R_i} \in \mathcal{R}$.

Ejemplo sencillo:

- $U = \{1, 2, 3, 4\}$
- $F = \{\{2, 3\}, \{1, 2, 3\}, U\}$
- Operaciones permitidas: complemento y unión.

La familia de conjuntos resultantes que se genera es:

$$\mathcal{R} = \{\emptyset, \{4\}, \{1, 4\}, \{2, 3\}, \{1, 2, 3\}, \{1, 2, 3, 4\}\}$$

Esto es, el álgebra de conjuntos generado por F.

La siguiente proposición es un resultado clásico en teoría de conjuntos:

Teorema 2.10 (Relación álgebra de conjuntos y anillo)**.** *[Lip98] Toda álgebra de conjuntos es también un anillo de conjuntos, pero el recíproco no es cierto.*

Demostración. Sea F un álgebra de conjuntos sobre U. Por definición, F está cerrada bajo complemento y unión finita, y contiene al conjunto vacío. Se tiene también que F es cerrada bajo intersección y diferencia, ya que:

$$A \cap B = \overline{(\overline{A} \cup \overline{B})}, \quad A \setminus B = A \cap \overline{B}$$

Por tanto, toda álgebra es también un anillo, ya que cumple las propiedades necesarias.
Sin embargo, no todo anillo de conjuntos es un álgebra, ya que un anillo no exige que $U \in F$.
Por ejemplo, sea:

$$F = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} \subseteq \mathcal{P}(\{1, 2, 3\})$$

Esta familia es un anillo (cerrada bajo diferencia y unión), pero no es un álgebra, ya que $\{1, 2, 3\} \notin F$. Por tanto, la implicación contraria no se cumple. □

Analizamos también el concepto de σ-álgebra, que va a ser esencial cuando queramos aplicar el concepto de medida clásico.

Definición 2.11 (σ-álgebra)**.** [Pal99] Una σ-álgebra es un álgebra $F \subset \mathcal{P}(U)$ que además verifica:

$$F_i \in F \quad \forall n \in \mathbb{N} \implies \bigcup_{n \in \mathbb{N}} F_i \in F$$

Es decir, esta estructura es cerrada bajo uniones numerables. Además, tenemos en cuenta el siguiente teorema:

Teorema 2.12 (Relación σ-álgebra de conjuntos y anillo)**.** *[Lip98] Todo σ-álgebra de conjuntos es también un anillo, pero el recíproco no es cierto.*

Demostración. Sea $F \subseteq \mathcal{P}(U)$ una σ-álgebra. Por definición, cumple:

- $U \in F$
- Si $A \in F$, entonces $\overline{A} \in F$
- Si $\{A_n\}_{n\in\mathbb{N}} \subseteq F$, entonces $\bigcup_{n=1}^{\infty} A_n \in F$

Como F está cerrada bajo complementos y uniones numerables, también lo está bajo intersecciones numerables (por las leyes de De Morgan), y en particular bajo intersección finita y unión finita (son un caso particular de las numerables). Además, está cerrada bajo diferencia, ya que:

$$A \setminus B = A \cap \overline{B}$$

Por tanto, F cumple las propiedades necesarias para ser un anillo de conjuntos.

Para demostrar que el recíproco no es cierto, consideramos el siguiente contraejemplo. Sea $U = \mathbb{N}$, definimos:

$$F = \{A \subseteq \mathbb{N} \mid A \text{ es finito o } \overline{A} \text{ es finito}\}$$

Esta familia es un anillo, ya que está cerrada bajo diferencia y unión finita:

- La unión de dos conjuntos finitos es finita.
- La diferencia de un conjunto finito y otro finito es finita.
- La diferencia de un conjunto cofinito (su complemento es un conjunto finito) y otro cualquiera sigue siendo finita o cofinita.

Sin embargo, F no es una σ-álgebra, ya que no está cerrada bajo uniones numerables. Por ejemplo, sea $A_n = \{2n\}$ para cada $n \in \mathbb{N}$. Entonces $A_n \in F$ porque cada conjunto es finito, pero:

$$\bigcup_{n=1}^{\infty} A_n = \{2, 4, 6, 8, \dots\}$$

es un conjunto infinito con complemento también infinito, por tanto:

$$\bigcup_{n=1}^{\infty} A_n \notin F$$

Queda demostrado que F no es una σ-álgebra. □

En particular, tenemos la relación: σ-álgebra $\subset$ Álgebra $\subset$ Anillo $\subset$ Semianillo. [Nie] [Zum]

2.3. Medidas de evaluación

Una vez que hemos analizado las estructuras algebraicas de los distintos conjuntos presentes en la definición de nuestro problema, nos interesa introducir las medidas de evaluación. Estas se dividen en tres categorías principales: medidas de asociación, direccionales, y otras; y las analizamos en función del punto del proceso en el que intervienen y de los subconjuntos que

evalúan. Por un lado, pueden actuar como criterio interno a un algoritmo concreto para guiar el proceso de construcción de las soluciones candidatas y de las aproximaciones intermedias. Por otro lado, sirven como criterio externo al algoritmo, aplicandose tras su ejecución para comparar diferentes soluciones candidatas y seleccionar la más adecuada según el criterio escogido. Finalmente, distinguimos entre medidas de optimización, cuyo valor buscamos maximizar o minimizar según el objetivo concreto, y medidas de restricción, donde imponemos un umbral que ha de cumplirse para constituir una solución válida.

2.3.1. Medidas de asociación

Las medidas de asociación son ampliamente utilizadas en el ámbito de la estadística y el análisis de datos para cuantificar la relación enntre variables categóricas. Cuando tenemos una población clasificada según dos o más variables, es decir, clasificación cruzada, surgen preguntas sobre el grado de asociación o dependencia que hay entre ellas, y en qué grado conocer el valor de una variable nos da información sobre la otra. La elección de la medida de asociación debe hacerse en función del contexto y de los objetivos específicos del problema y no de manera generalizada [BL21]. De hecho, no siempre es posible definir un único objetivo para una investigación. Muchas veces buscamos simplemente resumir de forma compacta una gran cantidad de datos o identificar patrones.

En términos de probabilidad, si X e Y son variables categóricas, una medida de asociación suele expresar cuanto se desvía la distribución conjuntos $P(X, Y)$ del producto de las marginales $P(X)P(Y)$, lo cual equivale a la independencia. Una opción común es construir modelos probabilísticos de actividad predictiva, y definir la medida de asociación como una probabilidad derivada de dicho modelo [GK79]. Esto mide la reducción relativa en la probabilidad de error al predecir una variable conociendo la otra.

Trasladando esto a nuestro problema de aproximación de conjuntos, las clasificaciones se corresponden con la pertenencia de los elementos al conjunto objetivo G y a la aproximación final $\tilde{G}$. Es decir, medir hasta qué punto ambas clasificaciones están asociadas, sin asumir prioridad de una sobre la otra. Para ello, no necesitamos construir explícitamente los modelos probabilísticos que hemos mencionado, sino que nos centraremos en métricas más sencillas que pueden interpretarse como medidas de asociación derivadas de las tablas de contingencia 2×2 entre el conjunto objetivo G y la aproximación $\tilde{G}$:

	En G	**No en** G	**Total**
En $\tilde{G}$	Verdaderos Positivos (TP)	Falsos Positivos (FP)	$\lvert\tilde{G}\rvert$
No en $\tilde{G}$	Falsos Negativos (FN)	Verdaderos Negativos (TN)	$\lvert U \setminus \tilde{G}\rvert$
Total	$\lvert G\rvert$	$\lvert U \setminus G\rvert$	$\lvert U\rvert$

Tabla 2.2.: Tabla de contingencia 2×2 entre el conjunto objetivo G y la aproximación $\tilde{G}$.

Hemos tomado como clase "positiva" la pertenencia a G, pues es nuestra realidad. Por lo tanto, los Verdaderos Positivos (o True Positives) son los elementos que están tanto en la aproximación como en el conjunto objetivo; los Falsos Positivos (o False Positives) son los

elementos que están en la aproximación pero no en G; los Falsos Negativos (o False Negatives) son los elementos de G que no están en la aproximación; y por último, los Verdaderos Negativos (o True Negatives) son los elementos que no están ni en $\tilde{G}$ ni en G.

De esta tabla, podemos sacar algunas de las medidas de asociación más comunes:

- Índice de Jaccard [RV96]: mide la proporción de elementos que pertenecen tanto a G y $\tilde{G}$, sobre los que pertenecen al menos a uno de los dos. Es una de las más utilizadas en el ámbito de cobertura de conjuntos y del análisis de datos. Su fórmula es:

$$\mathcal{J}(G, \tilde{G}) = \frac{|G \cap \tilde{G}|}{|G \cup \tilde{G}|} = \frac{TP}{TP + FN + FP}$$

- Coeficiente de Sørensen-Dice: es muy similar al índice de Jaccard pero pondera más los valores TP. Su fórmula es:

$$D(G, \tilde{G}) = \frac{2|G \cap \tilde{G}|}{|G| + |\tilde{G}|} = \frac{2TP}{2TP + FP + FN}$$

- Coeficiente de correlación ϕ [BB01]: mide la calidad de clasificaciones binarias. Es muy utilizado en el ámbito de la bioinformática y del Machine Learning, donde es más conocido como MCC (Matthews correlation coefficient). Utiliza toda la tabla, ya que también considera los TN. Toma el valor $+1$ cuando la predicción es perfecta, 0 cuando es una predición aleatoria, y -1 cuando es una predicción inversa. Es especialmente útil cuando existe un desbalanceo de clases. En nuestro problema, puede servir como medida asociativa más robusta cuanddo G es mucho más pequeño que U. Su fórmula es la siguiente:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- Test χ^2 de independencia [Urd05]: permite determinar si los elementos de una muestra caen en las categorías de forma aleatoria, es decir, si la división en categorías de una de las variables es independiente de la otra variable categórica. A pesar de ser de los tests de independencia más conocidos en la estadística, no lo utilizaremos en nuestro problema, pues nos centramos en métricas cuantitativas, más que en contrastes de hipótesis.

En la práctica se ha prestado mucha atención al equilibrio entre la precisión y el recall, especialmente en conjuntos de datos desbalanceados y en problemas como la detección de fraude [Rob21]. Aunque este análisis es más común en machine learning, donde se usan curvas precisión-recall para ajustar el threshold en cada iteración de entrenamiento, nos muestra su importancia en nuestro problema. No conviene despreciar una de estas dos medidas por completo y sólo focalizarnos en la otra. Si tenemos una precisión alta con un recall bajo, los elementos incluidos en la aproximación realmente pertenecen a G, pero se pueden estar ignorando muchos otros elementos de G. Por el contrario, si tenemos un recall alto con una precisión baja, aproximamos la mayoría de elementos de G, pero estamos incluyendo también muchos elementos de fuera de G. Esto nos muestra lo necesario que es considerar ambas

medidas conjuntamente, según nuestro objetivo.

Todas estas medidas asociativas son externas al algoritmo utilizado para obtener los candidatos a solución [HM15]. Esto se debe a que únicamente utilizan el conjunto objetivo y los elementos de $\mathcal{S}$, sin recurrir a pasos intermedios del proceso. Por ello, se aplican con posterioridad a la ejecución del algoritmo. Por un lado sirven para evaluar y comparar el rendimiento de diferentes algoritmos en nuestro problema. Es decir, la medida de evaluación se encarga de determinar el mejor algoritmo para aproximar el conjunto G. Por otro lado, pueden emplearse como criterio discriminador para seleccionar la solución óptima de entre todas las candidatas generadas por el algoritmo durante el proceso de construcción. Sin embargo, este tipo de medidas no son útiles como criterios internos de construcción, ya que requieren comparar conjuntos completos.

- Convención con signo: La medida toma valores entre -1 y $+1$, alcanzando estos extremos en caso de asociación completa, y siendo el 0 el caso de independencia. Es el caso del coeficiente de correlación ϕ. Esta convención es útil cuando existe un orden natural en las categorías que nos permita asignar un sentido a la relación.
- Convención sin signo: La medida toma valores entre 0 y +1, alcanzando el +1 en caso de asociación completa, y el 0 en caso de independencia. Es el caso de índice de Jaccard. Esta convención se utiliza cuando no hay un orden natural entre las categorías y no tiene sentido hablar de direcciones positivas o negativas.

Es importante destacar que la idea de "asociación completa" puede ser ambigua en contextos complejos. En cambio, el concepto de independencia sí es claro: por ejemplo, si la inclusión de un elemento en G es estadísticamente independiente de su aparición en $\tilde{G}$.

Finalmente, conviene y es interesante recordar que una vez definida una medida de asociación dentro de uno de estos rangos, es posible generar otras medidas cumpliendo la misma convención mediante transformaciones (como elevarla a una potencia o aplicarle funciones monótonas), sin perder su interpretación general.

2.3.2. Medidas direccionales

A diferencia de las medidas de asociación, las medidas direccionales no son simétricas, sino que evalúan una variable respecto de otra. Toman una como "realidad" y otra como su "predicción", por lo que son especialmente útiles cuando existe un criterio de referencia, como es nuestro caso, donde queremos aproximar G.

De la tabla 2.2, podemos extraer algunas de las medidas direccionales más utilizadas:

- Precisión: es la proporción de elementos predichos que son correctos. En nuestro problema, es la proporción de elementos de $\tilde{G}$ que pertenecen efectivamente a G. Su fórmula es:

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Está directamente relacionada con la sobrecobertura, pues cuanto mayor es la precisión, menor es la sobrecobertura de G, es decir, el número de elementos de $U \setminus G$ incluidos

en la aproximación. La diferencia es que la sobrecobertura se expresa en términos absolutos ($|\tilde{G} \setminus G| = FP$), mientras que la precisión es relativa, pues normaliza respecto al tamaño de $\tilde{G}$:

$$1 - \text{Precision} = \frac{FP}{TP + FP} = \frac{|\tilde{G} \setminus G|}{|\tilde{G}|}$$

.

- Recall o Sensibilidad: es la proporción de elementos reales que han sido correctamente predichos. En nuestro problema, cuántos de los elementos de G han sido cubiertos en $\tilde{G}$. Su fórmula es:

  $$\text{Recall} = \frac{TP}{TP + FN}$$

  Está directamente relacionada con la infracobertura, pues cuanto mayor es el recall, menor es la infracobertura de G, es decir, el número de elementos de G que quedan sin cubrir por la aproximación. Al igual que en el caso anterior, la infracobertura es absoluta ($|G \setminus \tilde{G}| = FN$), mientras que el recall es relativo, al normalizar respecto al tamaño de G:

  $$1 - \text{Recall} = \frac{FN}{TP + FN} = \frac{|G \setminus \tilde{G}|}{|G|}$$

  .

- Especificidad: es la proporción de negativos correctamente identificados. En nuestro problema, cuántos de los elementos de $U \setminus G$ están en $U \setminus \tilde{G}$. Su fórmula es:

  $$\text{Specificity} = \frac{TN}{TN + FP}$$

  Esta medida se centra únicamente en la calidad de la clasificación de los elementos negativos. Por ello, no siempre resulta de interés por sí sola, ya que ignora lo que ocurre con los elementos positivos. En muchos casos es preferible combinarla con otras métricas que reflejen tanto aciertos como errores en el conjunto completo.

- F1-Score: es la media armónica de precisión y recall. Equivale al coeficiente de Dice en clasificación binaria.

  $$\text{F1-Score} = \frac{2TP}{2TP + FP + FN} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Merece la pena incidir en el caso del F1-Score y analizarlo con mayor detalle. Esta medida aparece por primera vez en el libro Information Retrieval de Van Rijsbergen, en 1979 [CHK23]. En él se evidencia el interés en combinar precisión y recall, dos medidas muy útiles en la recuperación de información que, como ya hemos mencionado, resultan especialmente valiosas cuando se consideran conjuntamente. Antes de llegar a la fórmula definitiva del F-Score, se plantean distintas alternativas, aunque esta medida pronto se consolidó y empezó a utilizarse también en otros ámbitos de la informática, como la extracción de información y la clasificación de textos, convirtiéndose en una de las más conocidas y utilizadas. En cuanto a su naturaleza, el F1-Score ha generado cierto debate sobre si debe considerarse de asociación (simétrica) o direccional. En la mayoría de estudios se presenta como una medida direccional [HST+22], pues es la media armónica de precisión y recall, ambas medidas direccionales. Ahora bien, se trata de una medida simétrica, cuyo resultado no cambia dependiendo de qué clase se defina como positiva y negativa. Es decir, es invariante a intercambiar el conjunto

objetivo G con la aproximación $\tilde{G}$. Además, no depende de TN, sino que mide el parecido que hay entre G y $\tilde{G}$ sin tener en cuenta el resto del universo. De ahí su paralelismo con el coeficiente Dice, y su posible interpretación como medida asociativa.

En cuanto al caracter de estas medidas direccionales, por construcción, son externas al algoritmo utilizado para obtener los candidatos a solución [HM15]. Al igual que las medidas asociativas, comparan el conjunto objetivo con los elementos de $\mathcal{S}$, sin recurrir a pasos intermedios del proceso. Por ello, se aplican con posterioridad a la ejecución del algoritmo. No obstante, podemos adaptarlas para emplearlas también como medidas internas al algoritmo. Tomando como ejemplo la precisión, podemos aplicarla sobre la aproximación parcial en la iteracicón n, $\tilde{G}_n$, en lugar de sobre la solución final $\tilde{G}$. La manera exacta en la que estas medidas se aplican como internas depende del algoritmo concreto que utilicemos, como detallaremos más adelante.

Cabe destacar que tanto la exactitud como la especificidad dependen de los verdaderos negativos (TN), es decir, los elementos que no pertenecen ni a G ni a su aproximación, $\tilde{G}$. En campos como el diagnóstico médico, donde los negativos son tan relevantes como los positivos, este tipo de medidas son cruciales. En nuestro problema, sin embargo, los TN son los elementos de $(U \setminus G) \cap (U \setminus \tilde{G})$, que suelen ser muy numerosos y aportan poca información sobre la aproximación. Por ello, muchas métricas (como el F1-Score o Jaccard) los ignoran por completo.

Todas estas medidas direccionales pueden expresarse de forma natural como probabilidades condicionadas calculadas a partir de la tabla 2.2. Por ejemplo, la precisión corresponde a $P(G|\tilde{G})$, mientras que la sensibilidad es $P(\tilde{G}|G)$. Esto conecta directamente con la literatura de clasificación binaria y de tests diagnósticos, donde las medidas que hemos mencionado se definen en los mismos términos. En los campos como la medicina y epidemiología, se utilizan al definir la calidad de un test diagnóstico en los que se clasifican pacientes sanos y enfermos, de ahí la gran relación de nuestro problema con la biología computacional, como ya mencionamos en el apartado de aplicaciones e impacto.

2.3.3. Otras medidas

Dentro de esta sección incluimos todas aquellas medidas que no son ni asociativas ni direccionales. Estas evalúan diferentes criterios:

- Medidas de redundancia:

  $$\mu_n(F_i) = |F_i \cap \tilde{G}_n|$$

  Miden la intersección entre un conjunto candidato F_i y la solución parcial en el paso n, $\tilde{G}_n$. Esto permite priorizar soluciones donde los conjuntos seleccionados tienen poca intersección entre sí, lo cual favorece la diversidad. Es útil en contextos donde la redundancia es costosa o no deseable, como en casos de segmentación o clasificación. Sin embargo, puede llevar a descartar conjuntos útiles que, aunque se solapen, aportan una mejor cobertura sobre G.
  Al aplicarse sobre elementos de F y la aproximación parcial en cada paso, se trata de una medida de caracter interno, pues no depende de la aproximación final.

- Medidas de tamaño:
  - Tamaño local:
    $$\mu(F_i) = |F_i|, \qquad \mu(\tilde{G}_i) = |\tilde{G}_i|$$
    Esta medida nos da el cardinal de un conjunto F_i o de una aproximación parcial. Puede usarse para penalizar o priorizar conjuntos según su tamaño. Es útil si queremos incluir en nuestra aproximación conjuntos muy grandes, que cubren más elementos aunque quizás incluyan muchos ajenos a G. Podemos utilizarla cuando buscamos cubrir G usando el menor número posible de sunconjuntos. Por ejemplo, en casos prácticos en los que todos los subconjuntos tengan el mismo coste asociado, y queramos minimizar el coste total.
    De nuevo, al aplicarse sobre elementos de F o sobre las aproximaciones parciales, se trata de una medida de caracter interno, que no depende de la aproximación final.
  - Tamaño de la aproximación:
    $$\mu(\tilde{G}) = |\tilde{G}|, \qquad \mu(F') = |F'|$$
    Esta medida nos da el cardinal de la aproximación final, o bien el número de subconjuntos utilizados. Es útil en casos en los que cada elemento o cada nuevo subconjunto aumente el coste total, y queramos minimizarlo.
    En este caso, medimos el tamaño de la aproximación final o el número de subconjuntos de F utilizados en ella, por lo que necesitamos que el algoritmo haya concluido, y por tanto se trarta de una medida externa.
- Medidas ponderadas:
  $$\mu(F_i) = w_i, \quad w_i \in \mathbb{R}_0^+$$
  Esta medida asigna un peso o coste a cada subconjunto. Nos permite modelar situaciones específicas más complejas y realistas en las que el uso de algunos conjuntos puede tener un coste o un riesgo mayor que otros. Por ejemplo, aplicándolo al caso ya mencionado de publicidad, esta medida podría utilizarse cuando el coste del anuncio depende de la localización o zona. En este caso, la medida ponderada prioriza subconjuntos más baratos, aunque estos no garanticen la mejor cobertura.
  Como el coste está asociado a cada $F_i \in F$, se trata de una medida que guía la construcción de la aproximación final, y por tanto es de caracter interno.
- Medidas globales: son medidas que, si bien no se ven afectadas por cuál se toma como clase de referencia, no entran dentro de la categoría de medidas asociativas porque no miden directamente el grado de parecido entre los dos conjuntos, sino si coinciden o discrepan respecto a todo el universo. Se trata, por tanto, de medidas simétricas de caracter externo, porque trabajan sobre los conjuntos finales completos, no sobre construcciones intermedias.
  - Error global: mide la proporción de elementos mal clasificados sobre el total. En nuestro problema, mide los elementos que están o bien en G o bien en $\tilde{G}$ pero no en ambos ni en ninguno. Es una medida muy conocida y extendida en estadística y en el ámbito de machine learning. Su fórmula es:
    $$\text{Error global} = \frac{FP + FN}{TP + FP + FN + TN} = \frac{FP + FN}{|U|}$$

- Exactitud o Accuracy: es la proporción de elementos clasificados correctamente. En nuestro problema, cuántos de los elementos de todo el universo (ya sean de dentro de G o de $U \setminus G$) han sido correctamente aproximados. Su fórmula es:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{TP + TN}{|U|} = 1 - \text{Error global}$$

- Números de operaciones: esta medida cuantifica la complejidad de una aproximación en términos del número de operaciones entre conjuntos que han sido necesarias para construirla. No evalúa similitud ni relación direccional entre G y $\tilde{G}$, sino la simplicidad de la expresión que genera la aproximación. Puede usarse tanto de forma interna como de forma externa. Su fórmula es:

$$\mu(e) = \#\{\text{operaciones en la expresión } e\}.$$

Habiendo visto diferentes ejemplos de medidas, señalamos que siempre existe la posibilidad de definir nuevas medidas según nuestros intereses particulares. Una forma natural consiste en combinar varias de las medidas mencionadas aplicadas a un mismo subconjunto X de interés en nuestro problema:

$$\mu(X) = \mu_1(X) + \mu_2(X) + \cdots + \mu_n(X)$$

En este caso, basta con que todas las medidas se puedan aplicar sobre el mismo subconjunto.

En situaciones más complejas en las que nos interesa medir propiedades de distintos subconjuntos, podemos recurrir a una medida multidimensional. Por ejemplo, para dos subconjuntos relevantes X e Y:

$$\mu(X, Y) = \gamma_1(X) + \delta_1(Y) + \ldots,$$

donde las γ_i son medidas sobre X y las δ_i son medidas sobre Y. Esta idea es extrapolable a dimensiones superiores, permitiendo construir medidas más flexibles.

2.3.4. Restricción vs. Optimización

En la teoría, todas las medidas que hemos mencionado pueden usarse tanto en un sentido restrictivo, imponiendo un umbral inferior o superior que ha de respetarse para que una aproximación sea válida como potencial solución; como en un sentido de optimización, maximizando o minimizando su valor en función del objetivo. A pesar de ello, en la práctica algunas medidas se utilizan preferentemente en una de estas dos modalidades.

Un ejemplo claro es la última medida que hemos presentado: el número de operaciones utilizadas. Resulta natural aplicarla como restricción, fijando un máximo de operaciones permitidas para evitar que se dispare la complejidad computacional y el tiempo de ejecución. Así, las aproximaciones que excedan dicho umbral pueden ser automáticamente descartadas. En cambio, no sería tan útil como medida a optimizar, pues si buscamos el menor número de operaciones posibles, obtendríamos trivialmente el valor cero; mientras que si buscamos el mayor, nos daría soluciones más largas y complejas, sin aportar un beneficio real en términos

de la calidad de la aproximación.

Otro ejemplo de medida esencialmente restrictiva es el test χ^2 de independencia, que en inferencia estadística se aplica siempre fijando un nivel de significación α. Dicho nivel actúa como umbral: si el valor calculado de χ^2 supera el valor crítico correspondiente, se rechaza la hipótesis nula de independencia [Mol24]. De la misma manera, la medida de tamaño local puede utilizarse en sentido restrictivo, estableciendo un límite máximo para los subconjuntos F_i, de manera que aquellos de tamaño superior no se utilizan en la aproximación.

Las medidas restrictivas juegan un papel crucial en nuestro problema pues en la mayoría de los casos, nos permiten aplicar las restricciones propias de la variante del problema con la que estamos tratando. Por ejemplo, limitar el número de operaciones a k en la variante de la búsqueda exhaustiva con profundidad limitada o la de Greedy clásico. Dentro de las medidas restrictivas, podríamos diferenciar entre "hard" y "soft" constraints, tal y como es muy habitual en programación lineal [Ken75]. Esto significa que las "hard" constraints son las restricciones estructurales del problema, por lo que son valores que tienen que cumplirse sí o sí. Mientras que las "soft" constraints son restricciones deseables pero no completamente obligtorias. Somos un poco más permisivos, pudiendo incumplirse si el coste de no hacerlo es demasiado grande.

El resto de medidas que hemos presentado tienen más sentido como criterios de optimización. Por ejemplo, buscamos maximizar el índice de Jaccard, el coeficiente de Dice, la precisión o el recall; o minimizar la redundancia y el error global. Algunas, no obstante, se adaptan a ambas interpretaciones: por ejemplo, el tamaño de la aproximación final puede imponerse como restricción o minimizarse como objetivo, y lo mismo ocurre con el peso total de los conjuntos utilizados, que podemos fijar en caso de un presupuesto estricto o reducir para optimizar recursos.

Dicho todo esto, conviene recordar que el papel de las medidas como criterios de optimización o de restricción siempre depende del algoritmo concreto en el que se apliquen. En los algoritmos Greedy suelen utilizarse como funciones objetivo que guían la elección del mejor subconjunto en cada iteración, mientras que en Branch&Bound suelen utilizarse en forma de restricciones que actúan como cotas de poda.

2.4. Complejidad y clasificación computacional

Una vez que hemos definido las medidas, sus distintos tipos e interpretaciones, y hemos presentado diferentes ejemplos y aplicaciones, nos focalizamos en analizar el coste de calcularlas y la complejidad de nuestro problema.

La teoría de la computación surge en el siglo XX a partir de cuestiones fundamentales sobre los límites del cálculo y las matemáticas. En los años veinte, Hilbert planteó su programa de formalización matemática, que fue posteriormente limitado por los teoremas de incompletitud de Gödel. Estos resultados llevaron a investigadores como Turing y Church a preguntarse qué problemas podían resolverse de forma algorítmica.

En este contexto, Alan Turing introdujo el concepto de Máquina de Turing, un modelo teórico que formaliza la noción de cálculo. Se trata de un autómata con una cinta infinita y un cabezal de lectura/escritura que, siguiendo un conjunto finito de reglas, permite describir cualquier procedimiento computacional. Este modelo sentó las bases de la computabilidad moderna y sigue siendo la referencia fundamental para definir qué significa que un problema sea resoluble mediante un algoritmo.

Con el desarrollo de los ordenadores, la atención se trasladó hacia la eficiencia en la resolución de problemas [FH03]. Esto dio lugar a la teoría de la complejidad computacional, donde se definieron clases como P y NP, para clasificar problemas en función de los recursos necesarios para resolverlos.

Por un lado tenemos la complejidad espacial que se refiere a la cantidad de memoria que requiere un algoritmo. Por otro tenemos la complejidad temporal, que es el tiempo de ejecución necesario de dicho algoritmo. Sin embargo, veremos que ambas están estrechamente relacionadas.

2.4.1. Complejidad temporal y notación asintótica

Aún cuando un problema sea resoluble computacionalmente en la teoría, puede no serlo en la práctica si el tiempo o la memoria requeridos resultan excesivos. Muchas veces es difícil calcular el tiempo exacto que tarda un algoritmo en ejecutarse, por lo que normalmente se recurre a estimaciones asintóticas [Sip12]. En particular, nos interesa el crecimiento de la función de tiempo de ejecución en el peor caso, es decir, el término de mayor orden cuando el tamaño de la entrada tiende a infinito. Para expresarlo, utilizamos la notación conocida como "big-O".

Definición 2.13 (Big-O)**.** [Sip12] Sean f y g funciones tales que $f, g : \mathbb{N} \to \mathbb{R}^+$. Decimos que $f(n) = O(g(n))$ si existen constantes positivas c, n_0 tales que para cada $n \in \mathbb{N}$, con $n \geq n_0$, $f(n) \leq cg(n)$. En ese caso, $g(n)$ actúa como cota superior asintótica para $f(n)$.

Mientras que $O(g(n))$ indica que una función $f(n)$ no crece más rápido que $g(n)$ salvo constantes, existe una notación más restrictiva que expresa que $f(n)$ crece estrictamente más despacio que $g(n)$ cuando $n \to \infty$:

Definición 2.14 (Small-o)**.** [Sip12] Sean f y g funciones tales que $f, g : \mathbb{N} \to \mathbb{R}^+$. Decimos que $f(n) = o(g(n))$ si para todo $c \in \mathbb{R}^+$, existe un n_0, tal que $f(n) < cg(n) \quad \forall n \geq n_0$. En este caso, $g(n)$ es una cota estricta para $f(n)$.

Veamos los órdenes de algunas de las medidas que mencionamos en la sección anterior, así como de algunas operaciones necesarias en nuestro problema:

La medida de tamaño local es un gran ejempo de orden constante, $O(1)$, pues basta con leer el contador de elementos $|F_i|$, independientemente del tamaño de la entrada. Ahora bien, si tenemos una solución $\tilde{G}$ y el conjunto objetivo G, y queremos comprobar si un elementlogno de la solución está en el objetivo, podemos usar dos estrategias distintas: una búsqueda binaria en una estructura ordenada, con coste logarítmico, $O(log(|G|))$; o recorrer

todos los elementos de G, con orden lineal, $O(|G|)$. Si repetimos este procedimiento para cada elemento de $\tilde{G}$, el coste asciende a $O(|\tilde{G}| \cdot log|G|)$ y $O(|\tilde{G}| \cdot |G|) \approx O(|G|^2)$ respectivamente. Finalmente, para un orden exponencial, $O(2^n)$, basta con calcular todas las posibles combinaciones de subconjuntos de F y ver si cubren G. Esto corresponde a una búsqueda exhaustiva, que veremos en la parte informática.

Es claro en estos ejemplos abstractos la importancia de estudiar y comparar en detalle las estructuras de datos que podemos utilizar en nuestro problema, así como las operaciones asociadas. La elección adecuada puede marcar la diferencia entre una solución ineficiente, y otra mucho más rápida. Más adelante, en la parte informática, se analizará la complejidad concreta de los algoritmos que implementaremos para resolver el problema.

Utilizamos los siguientes cuatro gráficos para presentar todo esto de forma más visual. En ellos analizamos cómo cambia el tiempo de ejecución de un algoritmo/programa/instrucciones de distintos órdenes, según el tamaño de entrada. Lo hacemos para distintos tamaños de n en cada gráfico.

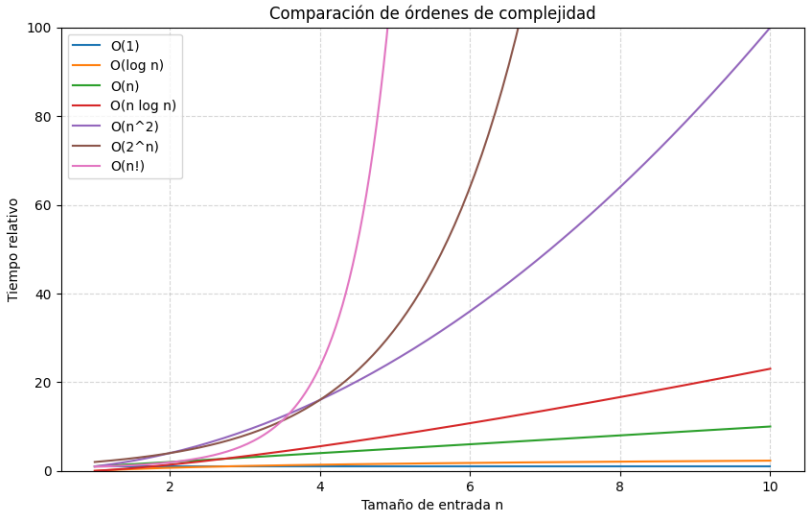

Figura 2.2.: Comparación de órdenes de complejidad para n hasta 10 (escala lineal).

En este primer gráfico, centrado en tamaños muy pequeños de entrada, vemos que incluso con n tan reducidos, los órdenes factorial y exponencial empiezan a crecer rápidamente. A partir de $n = 4$, $O(n!)$ supera drásticamente al resto, y a partir de $n = 6$, el orden $O(2^n)$ también se dispara. Los órdenes polinomiales como $O(n^2)$, resultan mejorables en este rango, en comparación con los lineales y logarítmicos, que practicamente no se distinguen entre sí. Con este gráfico vemos que, aún para entradas pequeñas, los algoritmos de complejidad exponencial o factorial dejan de ser prácticos.

Rescatamos nuestro ejemplo base 1.1 donde teníamos $|F| = 9, |G| = 5, |U| = 10$. Si quisieramos comprobar si un elemento de $\tilde{G}$ está en el objetivo, utilizando una estructura ordenada, que conlleva un orden $O(log|G|) = O(log(5))$, vemos en el gráfico que el tiempo es práctica-

mente el mismo que en el caso constante. Por su lado, si recorremos los 5 elementos de G, con orden $O(5)$, vemos que el tiempo crece, separandose del orden logarítmico, aún siendo un ejemplo con conjuntos muy pequeños.

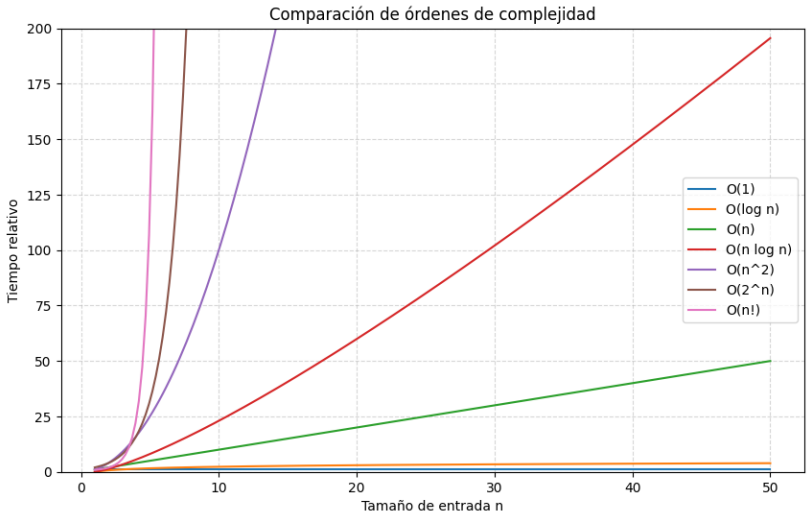

Figura 2.3.: Comparación de órdenes de complejidad para n hasta 50 (escala lineal).

Si ampliamos el rango hasta $n = 50$, observamos cómo los órdenes cuadráticos se evidencian como poco eficientes en comparación con los lineales y logarítmicos. El orden $O(nlogn)$ comienza a separarse de manera clara, poniendose en una posición intermedia: más costoso que los lineales, pero aún muy inferior a los exponenciales. Este gráfico muestra cómo, a medida que aumenta el tamaño de entrada, incluso órdenes considerados razonables, pueden llegar a ser limitantes.

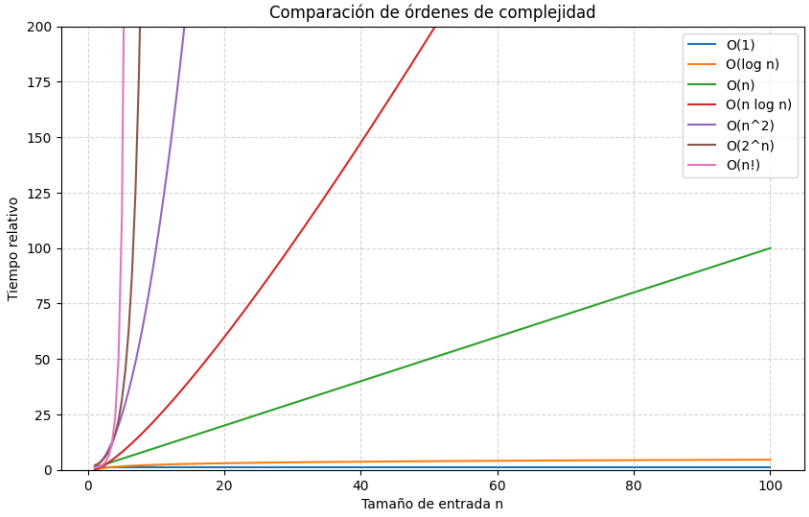

Figura 2.4.: Comparación de órdenes de complejidad para n hasta 100 (escala lineal).

Si analizamos tamaños hasta $n = 100$, la diferencia entre los órdenes es aún más evidente. Los factoriales, exponenciales y cuadráticos crecen tanto que dejan de ser representativos en la escala lineal, lo cual confirma su inviabilidad en la práctica. El orden $O(nlogn)$, aunque también se dispara, todavía se mantiene en un rango más asumible. Los órdenes lineal y logarítmico continúan siendo los únicos realmente escalables. En este gráfico vemos por qué los algoritmos cuadráticos, aunque más realistas que los exponenciales, pueden convertirse en un cuello de botella con entradas de tamaño medio.

Dado que nuestro problema surge de un caso real en el que se analizan coberturas de conjuntos de tamaño muy grande (miles o incluso millones de elementos), nos interesa estudiar qué ocurre en dichos casos. Para ello, en este último gráfico utilizamos una escala logarítmica que nos permite representar valores de n mucho mayores. Omitimos los órdenes factorial y exponencial, por su desproporción:

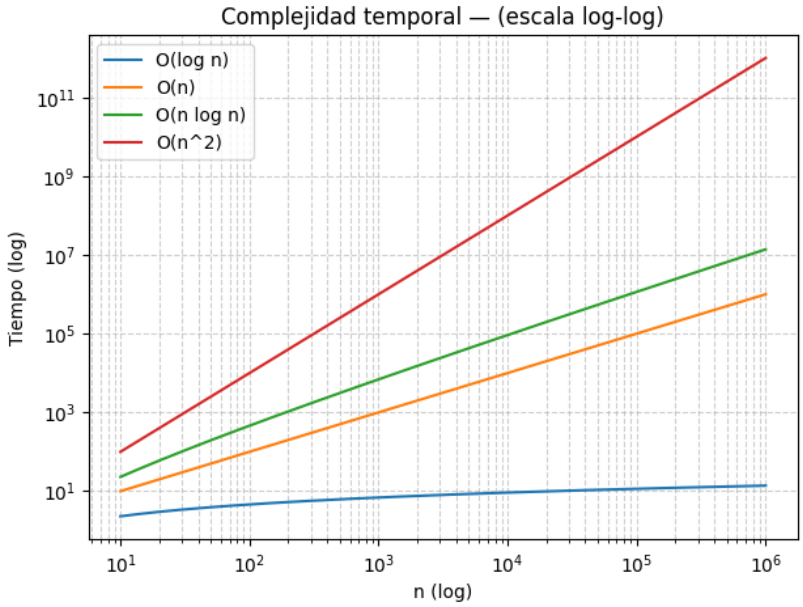

Figura 2.5.: Comparación de órdenes polinomiales en rango grande de n (escala log–log).

La escala logarítmica nos permite apreciar cómo los órdenes polinomiales se convierten en rectas con distinta pendiente. Aunque al principio parecen próximos, a medida que n crece la distancia entre ellos aumenta. Por ejemplo, para $n = 1000$, un algoritmo lineal puede requerir en torno a 10^3 pasos, uno $O(nlogn)$ en torno a 10^4, y uno cuadrático alrededor de 10^6. El orden $O(logn)$ se mantiene alrededor de 10 pasos para entradas de tamaño hasta $n = 10^5$, resultando mucho inferior que los otros órdenes.

A parte de los distintos órdenes temporales que hemos analizado y ejemplificado, de forma más en general se define la clase de complejidad temporal $TIME(t(n))$ como el conjunto de todos los problemas decidibles en tiempo $O(t(n))$ por una máquina de Turing. A partir de aquí se derivan las conocidas clases de complejidad P y NP.

2.4.2. Clases de complejidad y NP-Completitud

Definición 2.15 (Clase P)**.** [Sip12] Es la clase de todos los problemas de decisión resolubles por máquinas de Turing deterministas que trabajan en un tiempo polinomial. Siguiendo con la notación que veníamos usando:

$$P = \bigcup_k TIME(n^k)$$

De forma más intuitiva, son todos los problemas cuya solución se puede encontrar rápidamente. Un ejemplo es la multiplicación de matrices, de orden $O(n^3)$, o buscar un elemento en un array, de orden $O(n)$.

Antes de presentar la siguiente clase recordamos que, mientras que una Máquina de Turing determinista sólo tiene una trayectoria posible en cada paso de la ejecución, una Máquina no determinista, es aquella que puede tener varias transiciones posibles.

Definición 2.16 (Clase NP)**.** [Sip12] Es la clase de todos los problemas de decisión resolubles por máquinas de Turing no deterministas que trabajan en un tiempo polinomial.

De forma más intuitiva, son todos los problemas cuya solución se puede verificar en tiempo polinómico. Uno de los ejemplos más conocidos es el problema del viajante de comercio. En su versión de optimización: dado un conjunto de ciudades y las distancias entre ellas, encontrar el camino que visite cada ciudad una, y sólo una, vez, regrese al origen y minimice el coste total. La versión de decisión pregunta si existe un camino de coste $\leq k$ constante. En la figura 2.6 vemos un ejemplo de grafo asociado a este problema:

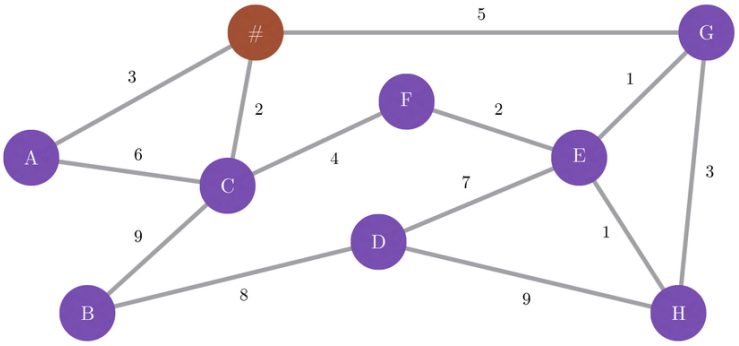

Figura 2.6.: Ejemplo de grafo de TSP. Fuente: [Bre13].

De forma intuitiva, decimos que un problema A se reduce a otro problema B cuando una solución para B puede ser utilizada para obtener una solución para A [Sip12]. Para ello se utiliza una función de reducción intermedia que convierte las instancias de A en instancias de B, de forma que resolver B implica automáticamente haber resuelto A. Se dice que un problema es NP-Hard, o NP-Duro, si todo problema de NP se reduce polinómicamente a él [Unid].

Con esto podemos presentar el concepto de NP-Completitud:

Definición 2.17 (NP-Completo)**.** [Sip12] Un problema es NP-Completo si pertenece a la clase NP y además todos los otros problemas de NP son reducibles a él. Es decir, pertenece a NP y es NP-Hard.

En 1971, Stephen Cook demostró con el Teorema de Cook, que ciertos problemas, como el de satisfacibilidad booleana (SAT), son NP-completos, introduciendo la noción de reducibilidad polinómica. Un año después, el informático Richard Karp, publicó un paper que contenía un con otros 21 problemas que también son NP-Completos. Esto supuso un gran avance, demostrando que eran equivalentes entre ellos y que lo que Cook había encontrado no era un problema puntual, sino parte de una clase mayor que permitiría resolver muchos otros problemas relevantes [Kar72]. Este marco constituye la base del análisis moderno de complejidad y motiva el estudio de problemas como la cobertura de conjuntos, aunque no fue reconocido inmediatamente tras su publicación. Además, ayudó a plantear el famoso problema P vs NP. Este, aún sin resolver, es uno de los grandes desafíos de la teoría de la complejidad y ha sido reconocido como uno de los Problemas del Milenio por el Clay Mathematics Institute.

La cuestión tiene distintas interpretaciones, asociadas a las diferentes definiciones de ambas clases, que son equivalentes: desde el punto de vista teórico, se trata de comparar lo que una Máquina de Turing no determinista puede resolver rápidamente con lo que una determinista puede hacer; desde el punto de vista práctico, se quiere comprobar si la capacidad de verificar soluciones rápidamente implica también la capacidad de encontrarlas con la misma eficiencia [Cha25].

Es bien conocido, y no nos detendremos en su demostración (pues no es directamente útil a nuestro problema), que $P \subseteq NP$, pues toda máquina determinista es también no determinista, siendo la inclusión trivial. Sin embargo, a lo que aún no hay respuesta es a si dicha inclusión es estricta, cosa que tendría consecuencias profundas en múltiples campos. Si $P = NP$, muchos problemas considerados intratables podrían resolverse de manera eficiente, lo que transformaría áreas como la inteligencia artificial, la optimización de recursos o la biología computacional. Un ejemplo más concreto e interesante, son los sistemas como RSA, que se considera NP-Completo y se basa en la dificultad de ciertos problemas, como la factorización de enteros grandes, que se consideran intratables en la práctica [Abd19]. Si se demostrase que $P = NP$, se pondría en riesgo la seguridad de estos sistemas, al hacerlos resolubles de forma eficiente.

Además, la frontera entre las clases P y NP, no está del todo clara y hay problemas que han pasado de una clase a otra cuando se ha descubierto un algoritmo que lo resolvía eficientemente. Es el caso del problema de demostrar si un número es primo. Hasta 2002, se consideraba que este problema pertenecía a la clase NP, pero tres científicos indios desarrollaron un algoritmo que demostró que el problema también pertenece a la clase P. Este cambio evidencia la importancia de encontrar algoritmos más eficientes, incluso para problemas que parecen intratables.

2.4.3. Complejidad teórica de nuestro problema

Nuestro problema es muy general y flexible, por lo que no resulta adecuado analizar su complejidad directamente, sino que conviene hacerlo sobre sus distintas variantes, donde

podemos hablar en términos de medidas y objetivos concretos. No obstante, haremos un razonamiento general, que abarque todas sus variantes, sin entrar a analizar en detalle cada una de ellas, sobreajustando el análisis.

Para estudiar nuestro problema en términos de las clases P y NP, hace falta convertirlo en uno de decisión que tenga el mismo grado de dificultad, cosa que siempre es posible [Unib]. Esto ocurre de forma natural cuando la formulación incluye un umbral sobre una medida con carácter restrictivo. Por ejemplo, en la variante de Recubrimiento de G, la cuestión es: ¿es posible encontrar una familia de subconjuntos $F' \subseteq F$ cuya unión forme un recubrimiento de G? Cuando por el contrario se trate de optimizar una medida, la transformación en problema de decisión la conseguimos imponiendo también un umbral k sobre la medida en cuestión. Por ejemplo, la variante sin repetición: ¿es posible encontrar una aproximación de G, sin repetir subconjuntos de F, cuya precisión sea al menos 0.8?

Recordamos el problema del Set Cover que ya mencionamos en la introducción. En su versión de decisión consiste en responder a lo siguiente: dado un conjunto universo U, una ccolección de m subconjuntos $S_i \subseteq U$, y un entero k, ¿es posible encontrar una familia C de como máximo k de estos subconjuntos S_i, tal que al hacer su unión, cubran todo U? [Cor]. Actualmente se considera un problema NP-Completo, por lo que no se conoce un algoritmo polinómico que lo resuelva en todos los casos. Es por esto que en la práctica, se emplean algoritmos aproximados y heurísticos para su resolución. Pasa igual con el Exact Cover Problem, que también se ha demostrado como NP-Completo.

Nuestro problema puede entenderse como una generalización de ambos. A diferencia de Set Cover o Exact Cover, no buscamos exclusivamente una partición ni un recubrimiento exacto, sino que definimos el objetivo en términos de una medida de aproximción más flexible. En algunas de nuestras variantes, sin embargo, sí que es posible formular de manera explícita estas condiciones, de tal manera que tanto el Exact Cover Problem como el Set Cover Problem se pueden formular como variantes concretas de nuestro problema.

Como estas dos variantes son NP-Completas, es claro que nuestro problema es NP-Duro, pues cualquier generalización de un problema NP-Duro hereda su complejidad. En particular, Set Cover se reduce trivialmente a nuestro problema, ya que basta con permitir sólo la unión, y definir la medida de evaluación como la cobertura exacta, con caracter restrictivo. Por tanto, si nuestro problema pudiese resolverse en tiempo polinómico, también se podría hacer con el Set Cover, lo cual contradice su pertenencia a NP.

Queda ahora por analizar si nuestro problema además pertenece a la clase NP, para determinar si puede considerarse NP-Completo. De manera intuitiva, queremos ver si es posible comprobar una solución candidata en tiempo polinómico. En nuestro caso, esto implica evaluar una medida concreta con respecto al conjunto objetivo $\tilde{G}$. Es importante atender a que las medidas utilizadas para comprobar la pertenencia a NP de cualquier variante de nuestro problema, han de ser medidas externas, definidas sobre la aproximación obtenida, y no medidas internas que dependan del proceso de resolución o del algoritmo.

Todas las medidas externas de asociación y direccionales que hemos explicado en la sección anterior, dependen de la tabla de contingencia 2×2 que presentamos con anterioridad. Construir esta tabla requiere, en el peor de los casos, recorrer todos los elementos de G para cada

elemento de $\tilde{G}$. Esto da lugar a una complejidad es $O(|\tilde{G}| \cdot |G|) = O(\tilde{g} \cdot g) \approx O(n^2)$, es decir, polinómica.

En cuanto a otras medidas externas, su complejidad es la siguiente:

- Tamaño de la aproximación: una única consulta al número de elementos de la estructura de datos correspondiente, con coste $O(1)$. Si la estructura no almacena directamente el tamaño, habría que recorrer el conjunto, lo que supondría un coste lineal.
- Error global y exactitud: se definen igualmente a partir de la tabla de contingencia, siendo como mucho un cálculo polinómico.

Visto todo esto, podemos concluir que la versión de decisión de cualquier variante presentada de nuestro problema, utilizando las medidas que consideramos, puede comprobarse en tiempo polinómico y, por tanto, pertenece a NP. Dado que además hemos demostrado su NP-dureza, afirmamos que nuestro problema es NP-Completo. Esto implica que, si nuestro problema tuviera un algoritmo que lo decidiese en tiempo polinómico en todos los casos, también lo tendrían otros como el Set Cover o el Exact Cover, lo cual equivaldría a demostrar que $P = NP$.

Al ser NP-Completo nuestro problema, surge la necesidad de recurrir a algoritmos aproximados y heurísticos para resolverlos, pues salvo que se demostrase que $P = NP$, no esperamos encontrar un algoritmo exacto en tiempo polinómico que lo resuelva en todos los casos. Por ello, en la práctica resulta más realista desarrollar métodos que aporten soluciones suficientemente buenas en un tiempo razonable, aunque no siempre sean óptimas.

2.4.4. Complejidad espacial

Aunque en nuestro problema nuestro foco principal es la complejidad temporal, es también conveniente entender como afecta a la memoria necesaria en escenarios especialmente grandes, para poder estudiar aproximaciones viables. Como ya anticipamos, hay una estrecha relación entre las clases de complejidad espacial y temporal que veremos a continuación.

Se define $SPACE(t(n))$ como el conjunto de problemas que pueden resolverse en espacio $O(t(n))$ por una Máquina de Turing determinista [AB09]. Más concretamente, para cada entrada x, el número total de celdas de la cinta que en algún momento han tenido algún símbolo durante la ejecución, es como mucho $c \cdot t(|x|)$, con $c > 0$ constante. De la misma manera, existe una clase $PSPACE$ definida como el conjunto de problemas que pueden resolverse utilizando espacio polinómico:

$$PSPACE = \bigcup_k SPACE(n^k)$$

En cuanto a su relación con las clases temporales, se tiene la siguiente cadena de inclusiones, que no nos detendremos a demostrar [AB09]:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME = \bigcup_k TIME(2^{n^k}).$$

Estas inclusiones implican que, como en la sección anterior vimos que nuestro problema pertenece a la clase NP, también pertenece a la clase $PSPACE$. Esto significa que puede resolverse utilizando, en el peor de los casos, un espacio polinómico en el tamaño de la entrada, lo cual es perfectamente asumible desde el punto de vista práctico.

2.4.5. Relación con SAT

A parte de los ya mencionados como Set Cover o Exact Cover, otro problema en que merece la pena reparar es el conocido como SAT o Problema de satisfacibilidad booleana. Este fue el primer problema demostrado como NP-Completo por el propio Stephen Cook en su famoso teorema. Esto significa que fue el primer problema conocido al que se pueden reducir todos los demás problemas de la clase NP, lo cual le aporta un papel central en el ámbito de la teoría de la computación.

Este problema de decisión consiste en lo siguiente [Gup23]: dada una fórmula de lógica booleana, ¿es posible encontrar una asignación de variables de manera que la fórmula se cumpla sin inconsistencias? Es decir, para n variables, hay 2^n posibilidades que hace falta comprobar.

Si bien nuestro problema no es exactamente SAT, y a priori puede no parecer similar, sí que existen paralelismos. Como mencionamos en la sección de estructuras algebraicas, en nuestro problema, $\mathcal{P}(U)$ es un álgebra de Boole y podemos traducir subconjuntos a proposiciones y operadores a conectores lógicos. Esto supone que alguna variante de nuestro problema, podría formularse como una instancia de SAT. Cada subconjunto de F sería una variable proposicional, y buscamos asignaciones que satisfagan las restricciones lógicas que equivalen a cubrir G.

Añadimos F y G para seguir con el ejemplo 2.3:

$$U = \{1,2,3,4\}, \qquad G = \{1,3\}, \qquad F = \{\{1\},\{2,3\},\{3\}\}$$

Como habíamos dicho, a cada subconjunto de $\mathcal{P}(U)$ le asignamos una proposición, excepto al vacío y al total. Enumeramos:

$$\{\{1\},\{2\},\{3\},\{4\},\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\}$$

como $x_1,\ldots,x_{14}$, respectivamente. En particular, nos interesa que x_1 corresponde a $\{1\}$, x_3 a $\{3\}$ y x_8 a $\{2,3\}$.

Otra opción viable es asignar proposiciones únicamente a los subconjuntos de F. Esto podemos hacerlo gracias a que todas las variantes de nuestro problema comparten la condición de que sólo podemos utilizar los subconjuntos de F en la aproximacicón de G. Plantearlo de esta manera reduce el número de proposiciones con el que trabajamos y el número de restricciones al formular el problema SAT, pues no hemos de prohibir utilizar los subconjuntos de $\mathcal{P}(U)$. Sin embargo, para seguir con la idea de que $\mathcal{P}(U)$ es un álgebra de Boole, planteada con anterioridad, y aprovecharnos de la utilidad que ello conlleva, trabajaremos sobre $\mathcal{P}(U)$.

Para formular la variante Exact Cover de G en SAT, imponemos:

1. Sólo se pueden utilizar subconjuntos de F, por lo que prohibimos todos los subconjuntos que no pertenecen a F:
$$\bigwedge_{x_A \in \mathcal{P}(U) \setminus F} \neg x_A$$

2. El elemento 1 debe aparecer una y sólo una vez. Cómo sólo aparece en un elemento de F: $\{1\} = x_1$, forzamos su elección.

3. El elemento 3 debe aparecer una y sólo una vez. Cómo aparece en dos elemento de F: $\{3\} = x_3$ y $\{2,3\} = x_8$, exigimos incluir exactamente uno de los dos:
$$(x_3 \vee x_8) \ \wedge \ (\neg x_3 \vee \neg x_8).$$

4. Hay que asegurarse de no incluir elementos de F que estén fuera de G. En particular, de no incluir el 2:
$$(\neg x_8).$$

Agrupando todas estas condiciones, obtenemos la siguiente fórmula proposicional para SAT:

$$x_1 \wedge (x_3 \vee x_8) \wedge (\neg x_3 \vee \neg x_8) \wedge (\neg x_8) \wedge \bigwedge_{x_i \in \mathcal{P}(U) \setminus F} x_i$$

Simplificando, esto sería equivalente a decir:

$$x_1 \wedge x_3 \wedge (\neg x_8) \wedge \bigwedge_{x_i \in \mathcal{P}(U) \setminus F} x_i$$

que corresponde a seleccionar exactamente $\{1\}$ y $\{3\}$, cuya unión es G y sin cubrir nada fuera de G.

En este caso, resolver esta instancia de SAT que hemos construido equivale a decidir la existencia de una solución de Exact Cover para G con los subconjuntos de F. Además, los subconjuntos utilizados en la cobertura son precisamente aquellos cuyas proposiciones toman el valor de verdad.

De forma análoga, podemos formular la variante Set Cover reformulando sin exigir que los elementos estén presentes en un único subconjunto. En el ejemplo anterior, la condición del punto 3 pasaría a ser la siguiente:
$$(x_3 \vee x_8)$$
Por la simplicidad de este ejemplo, la fórmula proposicional final es la misma, pero no siempre tiene por qué serlo.

Comparar SAT con nuestras variantes es especialmente interesante por dos razones. Primero, refuerza la utilidad de las condiciones estructurales que habíamos visto en la sección de estructuras algebraicas: como $\mathcal{P}(U)$ es álgebra de Boole, se pude conmutar fácilmente entre operaciones de conjuntos y operaciones lógicas. Por otro lado, esta equivalencia nos ayuda a demostrar que nuestro problema es NP-Completo, a través de reducciones, como lo hicimos

a partir del Exact Cover o Set Cover Problem. Además de todo esto, añade otra conexión entre nuestro problema y problemas centrales en el mundo de la teoría de la computación, siendo las soluciones de uno soluciones también del otro en algunos casos.

HASTA AQUÍ!!!

3. Implementación algorítmica

3.1. Lenguaje de programación

Para la parte informática de este Trabajo, nos centramos en implementar y comparar distintos algoritmos para resolver nuestro problema de aproximar un conjunto G mediante operaciones sobre una familia de conjuntos F, optimizando una métrica escogida por el usuario.

Hay varios aspectos que ya hemos mencionado y son claves a la hora de escoger los algoritmos que vamos a implementar. Entre ellos están: la eficiencia del algoritmo, la flexibilidad para adaptarse al uso de diferentes métricas de evaluación, y la facilidad de implementación y análisis. Por otro lado, como sabemos que nuestro espacio de búsqueda es exponencial, y podemos trabajar con conjuntos de grandes tamaños, la complejidad del algoritmo también juega un papel importante.

Hemos decidido utilizar el lenguaje C++, que permite tener un gran control sobre estructuras de datos y memoria, y es muy útil para algoritmos en los que realizamos muchas operaciones sobre conjuntos. En particular, hemos utilizado la librería estándar (STL), que tiene estructuras de datos interesantes como el *unordered_set* que utilizamos para representar conjuntos sin duplicados y con inserciones y búsquedas rápidas. Además, en C++ es fácil integrar el uso de OpenMP para paralelizar partes del algoritmo y acelerarlo, como hemos hecho con Greedy.

Aunque otros lenguajes como Python podrían haber sido utilizados, pues ofrecen un desarrollo simple y lectura de código fácil, al ser lenguajes interpretados y tener mayores tiemos de ejecución, son menos adecuados para tareas intensivas. Por otro lado, Java, al ejecutarse sobre una máquina virtual, tiene mayor abstracción y esto conlleva un rendimiento inferior al de C++. [Pan23]

3.2. Métricas de evaluación de resultados

Para evaluar el rendimiento de los distintos algoritmos en nuestro problema y compararlos entre sí de forma objetiva, hemos utilizado varias métricas comunes en el análisis de similitud entre conjuntos. Estas métricas no forman parte del algoritmo en sí, sino que las utilizamos a posteriori, una vez construida la aproximación $\tilde{G}$, con el objetivo de valorar su calidad y bondad respecto al conjunto objetivo, G.

Al tratarse de un problema de cobertura aproximada, es importante considerar distintas métricas que valoren varios aspectos, como que se escojan elementos correctos, que no queden elementos sin cubrir, y que no se incluyan elementos erróneos.

Estas son las métricas escogidas para el estudio, junto con sus propiedades claves:

- Índice de Jaccard: mide la proporción de elementos comunes entre el conjunto aproximado $\tilde{G}$ y el conjunto objetivo G, respecto al total de elementos que están en al menos uno de los dos. Toma valores entre 0 y 1, siendo 1 la igualdad entre ambos conjuntos, por lo que es relativa. Se calcula como:

$$J(\tilde{G}, G) = \frac{|\tilde{G} \cap G|}{|\tilde{G} \cup G|}$$

  Prioriza soluciones que equilibran cobertura y precisión, es decir, penaliza tanto los elementos faltantes como los sobrantes, sin distinguir entre errores por exceso y defecto. Es muy interesante cuando la similitud global nos importa más que los errores específicos.

- Diferencia simétrica: mide el número de elementos que están en uno de los conjuntos pero no en ambos. Es absoluta, midiendo en cardinales enteros. Se calcula como:

$$|\tilde{G} \triangle G| = |(\tilde{G} \setminus G) \cup (G \setminus \tilde{G})|$$

  Está estrechamente relacionada con el índice de Jaccard, y si este es pequeño, la diferencia simétrica será grande. Como premia soluciones que coinciden lo máximo posible con G, penaliza mucho incluso errores pequeños.

- Precisión: indica el número de elementos de $\tilde{G}$ que están también en G. Es relativa, tomando valores entre 0 y 1. Se calcula como:

$$Precision = \frac{|\tilde{G} \cap G|}{|\tilde{G}|}$$

  Premia soluciones que no contienen elementos erróneos, es decir, que evitan sobrecobertura. Puede ser muy alto si $\tilde{G}$ es muy pequeño, o incluso vacío, y esto no es necesariamente bueno.

- Recall: determina el número de elementos de G que hemos incluido también en $\tilde{G}$. Es relativa, pues se toma valores entre 0 y 1. Se calcula como:

$$Recall = \frac{|\tilde{G} \cap G|}{|G|}$$

  Da mejores resultados con soluciones que cubren completamente el conjunto objetivo, sin dejar elementos sin cubrir, lo cual puede implicar que se incluyan muchos elementos incorrectos.

- F1-score: es la media armónica entre precisión y recall. Se suele utilizar cuando necesitamos un equilibrio entre ambas, aunque no podemos reflejar si uno de los dos objetivos es más importante. Es también relativa, tomando valores entre 0 y 1. Se calcula como:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Estas métricas son muy comunes, no sólo en el estudio y comparación de conjuntos, sino también en problemas de clasificación, recuperación de información y evaluación de modelos

de aprendizaje automático. En nuestro caso, nos permiten evaluar tanto el comportamiento de cada algoritmo en la resolución del problema de aproximación, como el uso de las distintas medidas dentro de los algoritmos. [KGU+24]

3.3. Algoritmos escogidos

3.3.1. Greedy

El primer algoritmo que hemos escogido es Greedy (Voraz). Este selecciona conjuntos de F de forma iterativa para optimizar la métrica definida por el usuario. En cada iteración, evaluamos la métrica sobre el conjunto resultante de aplicar una de las operaciones entre un elementos de F y el conjunto $\tilde{G}$ actual, y nos quedamos con el subconjunto que más nos acerca a nuestro objetivo. [Muno6]

Al tomar decisiones paso a paso, escogiendo en cada momento la mejor opción local, destaca por su eficiencia computacional, pues no necesita explorar todas las posibles combinaciones. Simplemente escoge la mejor opción disponible según un criterio, con el objetivo de construir una buena solución. Sin embargo, estos algoritmos no garantizan que obtengamos la solución óptima, ya que pueden alcanzar óptimos locales al no considerar cómo afectan las decisiones tomadas en cada paso al resultado final. El proceso de elección de subconjuntos termina cuando alcanzamos un número de iteraciones máximo, cuando agotamos las opciones disponibles, o cuando tenemos una solución completa, que no mejora.

Un algoritmo Greedy tiene los siguientes elementos clave:

- Conjunto de candidatos del cual se escogerá un subconjunto que optimice alguna propiedad. En nuestro caso esto equivale al conjunto F, del que vamos escogiendo elementos.
- Conjunto de candidatos aceptados, que representa la solución parcial actual. En nuestro caso, es el conjunto $\tilde{G}$, que se va construyendo progresivamente.
- Función *esSolución()*, que determina si el conjunto de actual es una solución válida, aunque no sea óptima. Este punto no es estrictamente necesario en nuestro caso, pues cualquier combinación de elementos de F, mediante operaciones de conjuntos, es válida. Sin embargo, podría ser interesante para futuros trabajos incluir condiciones adicionales, como por ejemplo un tamaño mínimo.
- Función *esViable()*, que indica si la solución parcial podría llegar a formar una solución completa. En nuestro caso, esta función siempre va a devolver *TRUE*, ya que aceptamos cualquier combinación de subconjuntos de F como válida.
- Función de selección, que devuelve el candidato más prometedor entre los disponibles. En nuestro problema, equivale a escoger el par subconjunto-operación que maximiza la función *eval*, definida por el usuario.

[Fil17]

A continuación, mostramos los resultados obtenidos para una ejecución del algoritmo Greedy, con diferentes funciones evaluadoras (métricas). Para todos los casos hemos considerado un

universo U de 2500 elementos, el conjunto objetivo G de 645, y F que contiene 246 subconjuntos. Para medir la bondad del algoritmo hemos utilizado cuatro de las métricas anteriormente mencionadas: jaccard, precisión, recall y f1-score. Mostramos también el número de conjuntos incluidos en el conjunto aproximado $\tilde{G}$.

Evaluador	Jaccard	Precisión	Recall	F1-Score	#Subconjuntos	Tiempo (ms)
Jaccard	0.2711	0.2741	0.9612	0.4266	13	3398
Infracobertura	0.2585	0.2585	1.0000	0.4108	1	406
Conteo (grandes)	0.2580	0.2580	1.0000	0.4102	2	601
Diferencia simétrica	0.0047	1.0000	0.0047	0.0093	2	579
Balanceado	0.1408	0.4580	0.1690	0.2469	15	3626

Tabla 3.1.: Resultados de evaluación para diferentes funciones objetivo y sus métricas asociadas

Como era de esperar, con el evaluador Jaccard obtenemos el mejor F1-Score (0.4266) y el mayor índice Jaccard (0.2711), ya que es justo esta métrica la que estamos optimizando en cada iteración del algoritmo. Se utilizan 13 subconjuntos, que aunque no es el mayor número de todos los evaluadores, sí que implica un coste algo mayor en número de operaciones. Obtenemos también un tiempo de ejecución más alto, por el número de iteraciones y la evaluación costosa. Este evaluador consigue un buen resultado evitando la infracobertura, pero no tanto la sobrecobertura.

El evaluador de infracobertura penaliza exclusivamente que queden elementos de G sin cubrir, por lo que favorece subconjuntos muy grandes, sin importar los elementos irrelevantes que se incluyan. En este caso se ha seleccionado un sólo subconjunto, que contiene todos los elementos de G, como nos indica el recall. Sin embargo, vemos que sólo un 25 % de los elementos de $\tilde{G}$ están también en G, por lo que la sobrecobertura es un problema que no soluciona este evaluador, como era de esperar. Como ventaja, tiene un tiempo de ejecución muy bajo, de acorde al bajo número de operaciones e iteraciones que se han realizado.

El evaluador de conteo (grandes) premia los conjuntos grandes. Esto podría ser de interés si cada conjunto tiene el mismo coste asociado, y queremos gastar lo menos posible, eligiendo pocos subconjuntos de gran tamaño, que nos sugieren que cubren todo el objetivo. De hecho, este evaluador también consigue cubrir todo G, pero también tiene problemas con la sobrecobertura, como vemos en su precisión (0.2580). Vemos por tanto que presenta resultados muy similares al evaluador de infracobertura, pero con ligeras pérdidas en algunas de las métricas.

El evaluador de diferencia simétrica tiene el recall más bajo de todos, a pesar de tener una precisión perfecta. Esto nos indica que el conjunto aproximado no contiene ningún elemento fuera de G, por lo que se ha logrado mitigar la sobrecobertura, pero a cambio hay muchos elementos de G que no están siendo considerados en $\tilde{G}$. Esto nos sugiere que el conjunto aproximado es muy pequeño.

Por último el evaluador balanceado premia los elementos que se tienen en común, pero también penaliza los que están sólo en uno de los dos conjuntos, aunque en menor medida de lo que se premian los otros. Se tiene una buena precisión pero un recall bastante bajo también. Esto podría cambiar si subimos el peso asociado a la penalización de los elementos

que están en sólo un subconjunto. Usa el mayor número de subconjuntos y por ello requiere mucho más tiempo de ejecución.

Por lo tanto, si el objetivo es máxima cobertura con buena precisión, el evaluador Jaccard es el más equilibrado. Si es crucial cubrir todo G, aunque a costa de precisión, los evaluadores de infracobertura y conteo (grande) son opciones válidas. Si el objetivo es una alta precisión sin añadir elementos de fuera, pero no se penaliza dejar elementos sin cubrir, el evaluador de diferencia simétrica es extremo pero puede servir. El evaluador balanceado parece el menos útil, pues aunque puede servir cuando se busca evitar tanto sobrecobertura como infracobertura, es menos eficiente y más costoso.

Además, podemos concluir que Greedy se adapta muy bien a la métrica que le proporcionemos, optimizando la función que defina el usuario en cada caso. Sin embargo, esto no garantiza que obtengamos buenos resultados en las diferentes métricas, por lo que es crucial elegir cuidadosamente el evaluador que sigue el algoritmo. A pesar de que no siempre es posible obtener métricas óptimas por la construcción de G y F, los bajos valores en algunas de las métricas pueden deberse a que Greedy tiende a quedarse estancado en óptimos locales, al explorar la mejor opción paso a paso. Como punto positivo, los tiempos de ejecución de este algoritmo son muy razonables para los diferentes evaluadores, convirtiendolo en una alternativa adaptable y buena para una primera aproximación o casos con conjuntos grandes.

3.3.2. Backtracking

AÚN NO ESTÁ BIEN MIRADO!!!

La siguiente técnica algorítmica que hemos escogido es Backtracking, que es un método de búsqueda de soluciones exhaustiva sobre grafos dirigidos, donde se podan ramas poco prometedoras. Se para la búsqueda cuando alcanzamos un estado solución o cuando alcanzamos todos los estados solución. Normalmente se suelen implementar como un procedimiento recursivo, utilizando funciones de cota para evotar que se geneen estados si no van a conducir a ninguna solución, o a una peor de la que ya tenemos. Así se ahorra espacio en memoria y tiempo de ejecución.

Un algoritmo de backtracking tiene los siguientes elementos clave:

- Representación de la solución como una lista ordenada $(X_1, \ldots, X_n)$. En nuestro caso, representamos una solución como una secuencia ordenada de operaciones aplicadas a subconjuntos de F, donde cada X_i es un par subconjunto-operación.
- Función objetivo para determinar si la lista es una solución. Al igual que ocurría en Greedy, en nuestro caso no es estrictamente necesaria, pues cualquier combinación de elementos de F con operaciones es una solución, pero podemos añadir otras condiciones si lo consideramos necesario.
- Restricciones a los candidatos:
  - Implícitas del problema: valores que puede tomar cada X_i. En nuestro caso, sólo se pueden utilizar subconjuntos de F y una operación de entre las siguientes: unión, intersección, complemento.

    - Explícitas. Como el número k de pasos máximos que podemos realizar. Podemos además imponer restricciones adicionales para evitar la repetición de subconjuntos o evitar ciertos tamaños.

- Función de poda. En nuestro caso podamos ramas que no mejoran la métrica respecto al estado anterior.
- Organización del problema en un árbol de búsqueda. Para nosotros, cada nodo representa un estado parcial de $\tilde{G}$, y cada rama una nueva operación entre un subconjunto F_i y el $\tilde{G}$ actual. La raíz del árbol es el conjunto vacío, y las hojas representan solcuiones completas.
- Construcción de la solución en etapas. En cada etapa añadimos un nuevo par subconjunto-operación a la solución parcial actual. Repetimos el proceso hasta que se cumple la condición de parada.
- Retroceso si no es posible continuar. Si el nuevo estado no nos lleva a una solución viable, deshacemos la última operación y probamos con otra opción. De la misma manera, si no quedan candidatos, volvemos atrás en la solución e intentamos otra combinación.

[Bos19]

3.3.3. Algoritmos descartados

3.4. Referencias a elementos del texto

Para las referencias a los elementos del texto (secciones, capítulos, teoremas,...) se procede de la siguiente manera:

- Se *marca* el elemento (justo después del mismo si se trata de un capítulo o sección o en el interior del *entorno* en otro caso), mediante el comando `\label{`*etiqueta*`}`, donde *etiqueta* debe ser un identificador único.
- Para crear una referencia al elemento en cualquier otra parte del texto se usa el comando `\ref{`*etiqueta*`}` (únicamente imprime la numeración asociada a dicho elemento, por ejemplo 1 o **??**) o bien `\autoref{`*etiqueta*`}` (imprime la numeración del elemento así como un texto previo indicando su tipo, por ejemplo Capítulo 1 o **??**)

3.5. Bibliografía e índice

Esto es un ejemplo de texto en un capítulo. Incluye varias citas tanto a libros [**?**], artículos de investigación [Eul85], recursos online [Wik23] (páginas web), tesis [**?**], trabajo fin de máster [Tan96], trabajo fin de grado [Doe03] así como artículos sin publicar (preprints) [CIMT22] (en estos últimos usar el campo `note` para añadir la información relevante).
Ver el fichero `library.bib` para las distintas plantillas. Cada nueva referencia debe añadirse en dicho fichero siguiendo el estilo del código `bibtex` según el tipo de referencia (página web, tesis, trabajo fin de grado o máster, artículo de investigación, libro,...). Alternativamente se puede usar la web https://zbib.org para generar automáticamente el código `bibtex`.

3.6. Normativa de la comisión del Grado en Matemáticas

El TFG lo rigen dos normativas:

- una a nivel general de la UGR (Reglamento del Trabajo o Proyecto fin de Grado de la Universidad de Granada[1]) y
- otra complementaria a nivel de la Facultad de Ciencias (Reglamento del trabajo fin de grado en la Facultad de Ciencias de la Universidad de Granada[2]).

Además, la comisión del Grado de Matemáticas impone unos Requisitos de la memoria[3].
El TFG hay que elaborarlo preferiblemente en LaTeX y puede usar la plantilla disponible en Plantilla TFG grado en matemáticas formato .tex[4].
Toda la información anterior puede encontrarse en la web del Grado en Matemáticas[5].
Es conveniente tener presente la documentación anterior para la elaboración del TFG. En especial en lo relativo a las fechas de depósito del TFG para su defensa.
A continuación destaco algunos aspectos importantes de la misma:

- El plagio, entendido como la presentación de un trabajo u obra hecho por otra persona como propio o la copia de textos sin citar su procedencia y dándolos como de elaboración propia, conllevará automáticamente la calificación numérica de cero. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.
- Las memorias entregadas por parte de los estudiantes tendrán que ir firmadas sobre una declaración explícita en la que se asume la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

3.7. Formato de la memoria

La memoria se presentará usando un editor de textos científico, preferiblemente LaTeX, e incluir los siguientes apartados:

1. *Resumen en inglés*: Deberá estar escrito completamente en inglés y tener una longitud recomendada entre 800 y 1500 palabras.
2. *Introducción*. Deberá:
   - Indicar los *Objetivos del trabajo*: deberán aparecer con claridad los objetivos inicialmente previstos en la propuesta de TFG y los finalmente alcanzados con indicación de dificultades, cambios y mejoras respecto a la propuesta inicial. Si procede, es conveniente apuntar de manera precisa las interdependencias entre los distintos objetivos y conectarlos con los diferentes apartados de la memoria. Se pueden destacar aquí los aspectos formativos previos más utilizados.
   - Contextualizar el trabajo explicando antecedentes importantes para el desarrollo realizado y efectuando, en su caso, un estudio de los progresos recientes.

[1] https://secretariageneral.ugr.es/sites/webugr/secretariageneral/public/inline-files/BOUGR/187/PLANTILLA%20CABECERASDoc2.pdf
[2] https://fciencias.ugr.es/images/stories/documentos/reglamentos/reglamentoTfgCiencias23.pdf
[3] https://grados.ugr.es/matematicas/pages/infoacademica/tfg/requisitosTFG
[4] https://github.com/latex-mat-ugr/Plantilla-TFG/archive/master.zip
[5] https://grados.ugr.es/matematicas/pages/infoacademica/trabajofingrado

- Describir el problema abordado, de forma que el lector tenga desde este momento una idea clara de la cuestión a resolver o del producto a desarrollar y una visión general de la solución alcanzada.
- Indicar los resultados obtenidos.
- Citar las principales fuentes consultadas.

3. *Desarrollo del trabajo*: El trabajo se estructurará en partes o capítulos según convengan, con la posibilidad de incluir apéndices. Se recomienda que la extensión de esta parte (sin incluir los apéndices) sea de unas 50 páginas.
4. *Conclusiones y vías futuras*: Las conclusiones deberán incluir todas aquellas de tipo profesional y académico. Si hubiese posibles vías claras de desarrollo posterior sería interesante destacarlas aquí, poniéndolas en valor en el contexto inicial del trabajo.
5. *Bibliografía final*: Se incluirán tanto las fuentes primarias como todas aquellas cuyo peso haya sido menor en la realización del trabajo. Se recomienda un breve comentario de las referencias, ya sea individualizado, por grupos de referencias o global. En caso de incluir URLs de páginas web deberán ir acompañadas de título, autor y fecha de último acceso, entre otros datos relevantes. Se recomienda no abusar de este tipo de fuentes.

3.8. Recomendaciones

A la hora de abordar un trabajo como este, de cierta complejidad y extensión, es conveniente tener ciertas consideraciones desde un principio que ayuden a la organización y realización del mismo.

- La memoria deberá ceñirse a las directrices dadas en la sección precedente.
- Cualquier consulta externa (libro, artículo, página web, imagen,...) debe estar debidamente referenciada tanto en el texto como en la bibliografía al final del trabajo. La bibliografía debe de aparecer en orden alfabético (del primer autor) en el formato indicado en la plantilla.
- Se debe evitar copiar texto de forma literal, salvo citas literales, que se indicarán como tales y entrecomilladas. LaTeX proporciona el entorno `quote` para ello.
- Todas las imágenes y tablas incluidas en el documento deben figurar con su respectivos créditos (excepto que sean de elaboración propia). Por tanto, es recomendable guardar las referencias consultadas (direcciones web, libros) para la obtención de cualquier material gráfico o de datos.
- Si el trabajo contiene gran cantidad de vocabulario específico, conviene añadir un glosario de términos al final del mismo. Esto es mejor ir haciéndolo conforme se avanza en la redacción del trabajo.
- Es conveniente hacer un esquema inicial con la estructura general de la memoria: ¿de cuántas partes constará? ¿en qué orden? ¿qué incluirá cada una de ellas? En la plantilla proporcionada se recomienda una estructura general. Ello ayudará a organizar mejor el trabajo. No obstante, dicha estructura inicial puede ser modificada cuando el trabajo esté avanzado si el contenido lo requiere.

4. Conclusiones y Perspectivas Futuras

4.1. Conclusiones y trabajo futuro

4.1.1. Resumen de los principales hallazgos

4.1.2. Limitaciones del trabajo

4.1.3. Posibles líneas de investigación futuras

Este fichero `capitulo-ejemplo.tex` es una plantilla para añadir capítulos al TFG. Para ello, es necesario:

- Crear una copia de este fichero `capitulo-ejemplo.tex` en la carpeta `capitulos` con un nombre apropiado (p.e. `capitulo01.tex`).
- Añadir el comando `\input{capitulos/capitulo01}` en el fichero principal `tfg.tex` donde queremos que aparezca dicho capítulo.

A. Ejemplo de apéndice

Los apéndices son opcionales.
Este fichero `apendice-ejemplo.tex` es una plantilla para añadir apéndices al TFG. Para ello, es necesario:

- Crear una copia de este fichero `apendice-ejemplo.tex` en la carpeta `apendices` con un nombre apropiado (p.e. `apendice01.tex`).
- Añadir el comando `\input{apendices/apendice01}` en el fichero principal `tfg.tex` donde queremos que aparezca dicho apéndice (debe de ser después del comando `\appendix`).

Glosario

La inclusión de un glosario es opcional.
Archivo: `glosario.tex`

$\mathbb{R}$ Conjunto de números reales.

$\mathbb{C}$ Conjunto de números complejos.

$\mathbb{Z}$ Conjunto de números enteros.

Bibliografía

[ABo09] Sanjeev Arora y Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[Abd19] Mohamed Abdelnaby. Rsa algorithm in p vs np domain and time complexity, 05 2019.

[Alo] José A. Alonso. Números de bell. https://www.glc.us.es/~jalonso/exercitium/numeros-de-bell/. Exercitium, Universidad de Sevilla. [Accedido: 18-abr-2025].

[BBo1] P. Baldi y S. Brunak. *Bioinformatics, second edition: The Machine Learning Approach*. Adaptive Computation and Machine Learning series. MIT Press, 2001.

[BL21] Eric J. Beh y Rosaria Lombardo. *An introduction to correspondence analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, Incorporated, Hoboken, New Jersey, 2021.

[Bos19] Universidad Don Bosco. Guía 11: Algoritmo backtracking. https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-11.pdf, 2019. Programación IV, Ingeniería en Informática, II ciclo 2019.

[Bre13] Lucas Bremond. *An Optimal Tour Generation Strategy for a Multiple Rendezvous Mission to the Trojan Asteroids*. PhD thesis, Universite Paul Sabatier - Toulouse III, August 2013.

[Cha25] Julio Chai. *Los 7 Problemas del Milenio: El Camino Hacia lo Imposible*. 04 2025.

[CHK23] Peter Christen, David J. Hand, y Nishadi Kirielle. A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM Comput. Surv.*, 56(3), October 2023.

[CIMT22] Jesús Castro-Infantes, José M. Manzano, y Francisco Torralbo. Conjugate plateau constructions in product spaces, 2022. Preprint. arXiv: 2203.13162 [math.DG].

[Cor] Cornell University. Set cover. Course materials. Accedido: 1 septiembre 2025.

[Cru22] Esteban Rubén Hurtado Cruz. Álgebra superior 1 - unidad 1.3: Potencia, producto cartesiano, familias, 2022. Último acceso: 12 de marzo de 2025.

[Doe03] John Doe. Are we living in a simulation?, July 2003. Bacherlo's Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[Eul85] Leonhard Euler. An essay on continued fractions. *Math. Systems Theory*, 18(4):295–328, 1985. Translated from the Latin by B. F. Wyman and M. F. Wyman.

[FHo3] L. Fortnow y S. Homer. A short history of computational complexity. 80, June 2003. Computational Complexity Column.

[Fil17] Pablo R. Fillottrani. Algoritmos greedy. http://www.cs.uns.edu.ar/~prf/teaching/AyC17/downloads/Teoria/Greedy-1x1.pdf, 2017. Apuntes de la asignatura .Algoritmos y Complejidad", Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur.

[GK79] Leo A. Goodman y William H. Kruskal. *Measures of association for cross classifications*. Springer series in statistics. Springer-Verlag, New York, 1979.

[Gup23] Aarti Gupta. Cos 326: Functional programming. lecture: Boolean satisfiability (sat) solvers. https://www.cs.princeton.edu/lec/23-01-sat, 2023. Princeton University. Acknowledgements: Sharad Malik, Emina Torlak.

[Hal74] Paul R. Halmos. *Measure Theory*. Graduate Texts in Mathematics. Springer-Verlag, Berlin, first edición, 1974. Originally published by Litton Educational Publishing, Inc., 1950.

[HM15] Mohammad Hossin y Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 03 2015.

[HST+22] Steven A. Hicks, Inga Strümke, Vajira Thambawita, Mohamed Hammou, Michael A. Riegler, Pål Halvorsen, y Sravanthi Parasa. On evaluation metrics for medical applications of artificial intelligence. *Scientific Reports*, 12(1):5979, 2022.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. En R. E. Miller, J. W. Thatcher, y J. D. Bohlinger, editores, *Complexity of Computer Computations*, páginas 85–103. Plenum, New York, 1972.

[Ken75] JW Kendall. Hard and soft constraints in linear programming. *Omega*, 3(6):709–715, 1975.

[KGU+24] Katarzyna Krasnodębska, Wojciech Goch, Johannes H. Uhl, Judith A. Verstegen, y Martino Pesaresi. Advancing precision, recall, f-score, and jaccard index: An approach for continuous, ratio-scale measurements. SSRN Electronic Journal, 2024.

[KV12] Bernhard Korte y Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volumen 21 de *Algorithms and Combinatorics*. Springer, 5 edición, 2012.

[Lip98] Seymour Lipschutz. *Teoría de conjuntos y temas afines*. Schaum's Outline. McGraw-Hill, 1ª edición en español edición, 1998.

[Llu16] Cristina Jordan Lluch. Recubrimiento y partición de un conjunto. https://riunet.upv.es/handle/10251/64155, 2016. Universitat Politècnica de València. [Accedido: 18-abr-2025].

[LP14] David J. Leep y Pam Pierce. Exact cover by 3-sets. https://npcomplete.owu.edu/2014/06/10/exact-cover-by-3-sets/, 2014. NP-Complete Problems and What To Do With Them, Ohio Wesleyan University. [Accedido: 18-abr-2025].

[Mir] Jesús García Miranda. Capítulo 3. conjuntos ordenados. retículos y álgebras de boole. http://www.ugr.es/~jesusgm/Ordenes. Universidad de Granada. Apuntes de docencia. Accedido: 3 de septiembre de 2025.

[MMW24] Nima Moradi, Fereshteh Mafakheri, y Chun Wang. Set covering routing problems: A review and classification scheme. *Computers & Industrial Engineering*, 198:110730, 2024.

[Mol24] Desiré Romero Molina. Tema 7: Contraste de hipótesis. Apuntes de la asignatura Inferencia Estadística, Universidad [poner nombre de tu uni], 2024. Material docente facilitado por la profesora.

[Muno6] Tamara Munzner. Approximation algorithms for np-hard problems: The set cover problem. https://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf, 2006. Massachusetts Institute of Technology (MIT). [Accedido: 18-abr-2025].

[Nie] Lars Tyge Nielsen. Curso de probabilidad ii. http://lya.fciencias.unam.mx/lars/pub/proba2.pdf. Facultad de Ciencias, UNAM, México. [Accedido: 18-abr-2025].

[Pal] Eugenio Miranda Palacios. Álgebra básica. https://www.ugr.es/~bullejos/AlgI/anillos.pdf. Apuntes de Álgebra I, Universidad de Granada.

[Pal99] Antonio Pallares. Apuntes de teoría de la medida. https://webs.um.es/apall/archivos/medida/medida1998-99.pdf, 1999. Universidad de Murcia, 1998–1999.

[Pan23] Abhinn Pandey. Python vs. c++ vs. java: Choosing the right language for your project. https://abhinnpandey.medium.com/python-vs-c-vs-java-choosing-the-right-language-for-your-project-31947682a1fd, 2023. Consultado el 5 de mayo de 2025.

[Rob21] Robinteuwens. Precision vs recall: Optimizing fraud costs, 2021. Accedido: 3 de septiembre de 2025.

[RTF76] Charles ReVelle, Constantine Toregas, y Louis Falkson. Applications of the location set-covering problem. *Geographical Analysis*, 8(1):65–76, 1976.

[RV96] Raimundo Real y Juan M. Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic Biology*, 45(3):380–385, 09 1996.

[Set20] Capítulo 1: Teoría axiomática de conjuntos. https://www.fing.edu.uy/fundamentos-1, 2020. Apuntes de curso.

[Sip12] M. Sipser. *Introduction to the Theory of Computation*. Introduction to the Theory of Computation. Cengage Learning, 2012.

[Tan96] Jian Tang. Spin structure of the nucleon in the asymptotic limit. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1996.

[Unia] Generalidades de teoría de conjuntos. https://riemann.unizar.es/~jicogo/docencia/topoi/At1.pdf. Departamento de Matemáticas, Universidad de Zaragoza. Accedido el 22 de mayo de 2025.

[Unib] Universidad Nacional del Centro de la Provincia de Buenos Aires. Algoritmos heurísticos y aproximados. https://users.exa.unicen.edu.ar/docs/Teorica-1.pdf. Accedido: 3 de septiembre de 2025.

[Unic] University of Cambridge. Sets and logic. https://www.cl.cam.ac.uk/teaching/DiscMathII/. Accedido: 3 de septiembre de 2025.

[Unid] University of Illinois Urbana-Champaign. 30 np-hard problems. Course notes. Accedido: 1 septiembre 2025.

[Urd05] T.C. Urdan. *Statistics in Plain English*. Online access with subscription: Proquest Ebook Central. Lawrence Erlbaum Associates, 2005.

[Wik23] Wikipedia. Leonhard Euler — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Leonhard_Euler, 2023. [Recurso online, accedido el 27 de julio de 2023].

[Zum] Jörg Zumbraegel. Semirings and semiring modules. https://staff.fim.uni-passau.de/~zumbraegel/semirings.html. University of Passau. [Accedido: 18-abr-2025].