

# INTRODUCCIÓN A LA HOMOLOGÍA SINGULAR. ANÁLISIS Y RESOLUCIÓN DEL PROBLEMA DEL CLIQUE MÁXIMO.

JOSÉ CARLOS ENTRENA JIMÉNEZ



**UNIVERSIDAD  
DE GRANADA**

Septiembre 2017

Francisco Urbano Pérez-Aranda, Salvador García López  
Universidad de Granada



### Licencia

Este trabajo está bajo una licencia [Creative Commons Attribution - ShareAlike 4.0 International License](#).

Con esta licencia, puedes:

- Compartir, copiar, y redistribuir el contenido.
- Modificar el material de cualquier forma, incluso con propósito comercial.

Con las siguientes obligaciones:

-  Dar el crédito apropiado, aportando un enlace a la licencia, e indicar los cambios realizados.
-  Cualquier copia, o modificación debe ser distribuida con la misma licencia que el original.

No aplicar medidas de carácter legal o tecnológico que impidan a otros hacer aquello que la licencia permita.

## ABSTRACT

---

In this work, we define the basics of singular homology theory, starting with the necessary geometric constructions and the definition of the singular homology groups. Following the definitions and a proof of homotopy invariance, we prove the baricentric subdivision theorem, the central result of this work, that leads us immediately to the Mayer-Vietoris sequence. This and other results are applied to obtain the homology groups of the spheres and a number of classical theorems, including the Brouwer fixed-point theorem and the Jordan-Brouwer separation theorem.

On the second part, we introduce the maximum clique problem, a well studied combinatorial optimization problem, known to be NP-hard. Due to this, the use of metaheuristics is necessary to solve it in reasonable time. Several heuristics are considered, explained in detail and applied to DIMACS instances. We discuss their behaviour and compare them, showing which of the proposed algorithms can yield satisfactory results.

**KEYWORDS** Homology theory, Homotopy invariance, Baricentric subdivision theorem, Mayer-Vietoris sequence, Maximum clique, Combinatorial optimization problem, Metaheuristics, NP-hard, DIMACS.

---

Yo, **José Carlos Entrena Jiménez**, alumno de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Facultad de Ciencias** y de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** de la **Universidad de Granada**, con DNI XXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca de ambos centros para que pueda ser consultada.

*Granada, 26 de septiembre de 2017*



José Carlos Entrena Jiménez

## ACKNOWLEDGMENTS

---

Me gustaría agradecer a mi familia por su apoyo y ayuda durante todos estos años. Han representado mucho para mí, y lo seguirán haciendo. También quiero agradecer a todas mis amistades y seres queridos, que me han aportado grandes momentos en mi vida.

Mi más sincero agradecimiento a mis dos tutores, Francisco Urbano y Salvador López, por instruirme y apoyarme durante este proyecto y durante mis estudios.

Finalmente, quisiera agradecer a la Universidad de Granada, a la Facultad de Ciencias y a la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación, por darme la oportunidad de realizar mis estudios en ellas.

## ÍNDICE GENERAL

---

<b>I</b>	<b>HOMOLOGÍA SINGULAR</b>	<b>1</b>
0.1	Introducción al trabajo	2
0.2	Objetivos	2
0.3	Organización de la memoria	2
0.4	Referencias	3
<b>1</b>	<b>INTRODUCCIÓN</b>	<b>4</b>
1.1	Definiciones iniciales	4
1.2	Homología de un punto	8
1.3	El grupo $H_0(X)$ y la arcoconexión	9
1.4	Invarianza homotópica de la homología singular	11
1.5	Homología singular de un par	15
<b>2</b>	<b>CÁLCULO DE LA HOMOLOGÍA SINGULAR</b>	<b>21</b>
2.1	Teorema de subdivisión baricéntrica	21
2.2	Subdivisión baricéntrica de símlices	22
2.3	Subdivisión baricéntrica de símlices singulares afines	24
2.4	Subdivisión baricéntrica de símlices singulares	28
2.5	Demostración del teorema	29
2.6	Consecuencias del teorema de subdivisión baricéntrica	32
2.6.1	Homología de las esferas	36
<b>3</b>	<b>ALGUNOS RESULTADOS CLÁSICOS</b>	<b>39</b>
3.1	Teorema del punto fijo de Brower	39
3.2	Teoremas de invarianza	41
3.3	Separación de Jordan-Brower. Invarianza del dominio	44
3.4	Grado de una aplicación continua	48
<b>II</b>	<b>ANÁLISIS Y RESOLUCIÓN DEL PROBLEMA DEL CLIQUE MÁXIMO</b>	<b>52</b>
<b>4</b>	<b>INTRODUCCIÓN</b>	<b>53</b>
4.1	Introducción al problema	53
4.2	Introducción a las metaheurísticas	53
4.3	El repositorio DIMACS	54
4.4	Objetivos	55
4.5	Metodología y trabajo realizado	56
4.6	Organización	57
<b>5</b>	<b>ALGORITMOS</b>	<b>58</b>
5.1	Consideraciones iniciales	58
5.1.1	Operador <i>add</i>	58
5.1.2	Operador <i>swap</i>	59
5.1.3	Operador <i>drop</i>	59
5.1.4	Entorno	60
5.2	Algoritmos <i>greedy</i>	60

5.2.1	<i>Greedy</i> en MCP	60
5.2.2	<i>Greedy</i> básico	61
5.2.3	<i>Greedy</i> adaptativo	61
5.3	Búsqueda local	62
5.3.1	Búsqueda local en MCP	63
5.3.2	1LS	63
5.3.3	Dynamic Local Search	65
5.4	Enfriamiento simulado	66
5.4.1	Enfriamiento simulado en MCP	67
5.4.2	Enfriamiento simulado básico	67
5.4.3	Enfriamiento simulado adaptado	68
5.5	Búsquedas multiarranque	69
5.5.1	GRASP	70
5.5.2	ILS	72
5.6	Algoritmos de colonia de hormigas	74
5.6.1	ACO en MCP	75
5.6.2	ACO básico	75
5.6.3	ACO con enfriamiento simulado	77
5.7	Algoritmos genéticos	77
5.7.1	Genéticos en MCP	78
5.7.2	Algoritmo estacionario	79
5.8	Algoritmos meméticos	80
5.8.1	Meméticos en MCP	80
5.8.2	Algoritmo implementado	81
6	IMPLEMENTACIÓN	82
6.1	Lenguaje utilizado	82
6.2	Funcionalidad	82
6.3	Inicialización y ejecución del programa	83
6.4	Algoritmos	84
6.4.1	Algoritmos <i>greedy</i>	84
6.4.2	Búsqueda local	84
6.4.3	Enfriamiento simulado	85
6.4.4	GRASP	85
6.4.5	ILS	86
6.4.6	Algoritmos de colonia de hormigas	86
6.4.7	Algoritmos genéticos	86
6.5	Métodos auxiliares	87
7	RESULTADOS Y ANÁLISIS	89
7.1	Consideraciones	89
7.2	Análisis	90
7.2.1	Algoritmos <i>greedy</i>	90
7.2.2	Búsqueda local	92
7.2.3	Enfriamiento simulado	94
7.2.4	ILS	95
7.2.5	GRASP	97
7.2.6	Algoritmos de colonia de hormigas	99

7.2.7	Algoritmos genéticos	101
7.2.8	Comportamiento general	102
7.3	Conclusiones	105
BIBLIOGRAFÍA		107



## ÍNDICE DE FIGURAS

---

Figura 2.1	$Sd(s_1)$	23
Figura 2.2	$Sd(s_2)$	23

## ÍNDICE DE CUADROS

---

Cuadro 1	Resultados en algoritmos greedy.	91
Cuadro 2	Resultados en búsqueda local.	93
Cuadro 3	Resultados en enfriamiento simulado.	95
Cuadro 4	Resultados en ILS.	96
Cuadro 5	Resultados en GRASP.	98
Cuadro 6	Resultados en algoritmos de colonia de hormigas.	100
Cuadro 7	Resultados en algoritmos genéticos.	102
Cuadro 8	Clasificación por grafos.	104
Cuadro 9	Recuento.	104

## Parte I

### HOMOLOGÍA SINGULAR

En esta parte, vamos a ver una introducción a la homología singular. Definiremos los grupos de homología singular de un espacio topológico y estudiaremos algunas de sus propiedades. Continuaremos demostrando el teorema de subdivisión baricéntrica, resultado que nos lleva a la obtención de herramientas para el cálculo de la homología, y las aplicaremos al caso particular de las esferas. Finalmente, demostraremos resultados clásicos haciéndonos valer de nuevas construcciones, unidas a los resultados ya demostrados.

## 0.1 INTRODUCCIÓN AL TRABAJO

El propósito de este trabajo es introducir la teoría de homología singular sobre espacios topológicos. Esto lo haremos de forma constructiva, introduciendo los conceptos y estructuras necesarios para poder definir la homología singular. Tras definirla, estudiaremos sus propiedades más inmediatas y algunos casos sencillos, y extenderemos el concepto para pares topológicos, trasladando los resultados obtenidos con anterioridad.

Una vez asentadas las bases, trataremos el teorema de subdivisión baricéntrica, el resultado más importante del trabajo, que nos permite obtener herramientas para el cálculo de la homología de espacios topológicos. Será necesario introducir nociones geométricas y demostrar ciertas propiedades de estas nuevas construcciones para la demostración del teorema.

Demostrado el teorema de subdivisión baricéntrica, trataremos sus consecuencias, siendo las más importantes la sucesión de Mayer-Vietoris y el teorema de escisión. Con ellas, seremos capaces de calcular la homología de las esferas, y lo haremos de dos formas distintas; una mediante escisión, y otra usando la sucesión de Mayer-Vietoris.

Después de calcular la homología de las esferas demostraremos un número de resultados clásicos, para lo que definiremos la homología local de un espacio topológico. Obtendremos, entre otros, el teorema del punto fijo de Brower, el teorema de separación de Jordan-Brower y ciertos teoremas de invarianza. Finalmente, apoyándonos en la definición del grado de una aplicación continua sobre una esfera, seremos capaces de demostrar resultados sobre campos definidos en esferas.

## 0.2 OBJETIVOS

Los objetivos de este trabajo son los siguientes:

- Construcción de la homología singular y estudio de sus propiedades.
- Cálculo de la homología singular de las esferas y obtención de aplicaciones.

El primer objetivo se ha cumplido satisfactoriamente, y está desarrollado tanto en el [Capítulo 1](#) como en el [Capítulo 2](#).

El segundo objetivo también se ha cumplido, con el cálculo de la homología de las esferas al final del [Capítulo 2](#), y la obtención de aplicaciones desarrollada en el [Capítulo 3](#).

## 0.3 ORGANIZACIÓN DE LA MEMORIA

El [Capítulo 1](#) contiene todo lo referente a la definición de la homología singular de un espacio topológico: definiciones, propiedades y construcciones algebraicas y geométri-

cas. También se demostrará la invarianza homotópica de la homología, y se definirá la homología de los pares topológicos.

En el [Capítulo 2](#) se demostrará el teorema de subdivisión baricéntrica y se obtendrán herramientas para el cálculo de la homología singular: el teorema de escisión y la sucesión de Mayer-Vietoris. Estos serán utilizados para deducir la homología de las esferas.

Finalmente, en el [Capítulo 3](#) obtendremos resultados clásicos, como el teorema del punto fijo de Brower, el teorema de separación de Jordan-Brower y algunos teoremas de invarianza. Para ello, nos haremos valer de los conceptos de homología local y de grado de una aplicación continua definida en una esfera.

#### 0.4 REFERENCIAS

Para la realización de este trabajo se ha utilizado principalmente el libro de teoría de homología de James W. Vick [[18](#)], complementado por los títulos de William S. Massey [[14](#)] y Sze-Tsen Hu [[11](#)], que también tratan la teoría de homología.

## INTRODUCCIÓN

---

### 1.1 DEFINICIONES INICIALES

En este capítulo introduciremos la **homología singular** de un espacio topológico. Comenzaremos con un conjunto de definiciones iniciales, introduciendo la noción de **símplice singular**. Con ellos, construiremos un conjunto al que dotaremos de estructura algebraica, y sobre el que definiremos una aplicación que, gracias a sus propiedades, nos permitirá definir la homología singular del espacio. Seguidamente, demostraremos algunos resultados sencillos sobre ciertos espacios topológicos, continuando con la invarianza homotópica de la homología. Concluiremos el capítulo definiendo la homología sobre pares topológicos, y viendo como se trasladan los resultados demostrados con anterioridad.

#### Definición 1.1

Un ***p*-símplice**  $s_p$  es la envolvente convexa de  $p + 1$  puntos afínmente independientes en  $\mathbb{R}^p$ , a los que llamaremos *vértices*,  $\{x_0, \dots, x_n\}$ , esto es

$$s_p = \{\lambda_0 x_0 + \dots + \lambda_p x_p \mid \lambda_0 + \dots + \lambda_p = 1, \lambda_i \geq 0 \forall i\}.$$

Cuando los vértices son  $x_i = (0, \dots, 1, \dots, 0)$ , con el 1 en la posición  $i$ -ésima, le llamaremos el ***p*-símplice standard**, y lo notaremos por  $\sigma_p$ .

Podemos establecer un homeomorfismo  $h: \sigma_p \rightarrow s_p$  por

$$h(t_0, \dots, t_p) = \sum_{i=0}^p t_i x_i.$$

#### Definición 1.2

Sea  $X$  un espacio topológico. Un ***p*-símplice singular** en  $X$  es una aplicación continua  $\phi: \sigma_p \rightarrow X$ ,  $p \geq 0$ .

Representaremos el conjunto de  $p$ -símplices singulares de  $X$  como

$$F_p(X) = \{\phi: \sigma_p \rightarrow X \mid \phi \text{ es } p\text{-símplice singular}\}.$$

Es claro que  $F_0(X) = X$ . Además,  $F_1(X)$  son los arcos en  $X$ , identificables por  $C([0, 1], X)$ .

Si tenemos una aplicación continua entre espacios topológicos  $f: X \rightarrow Y$ , podemos definir

$$f_\#: F_p(X) \rightarrow F_p(Y) \quad \text{por} \quad f_\# = f \circ \phi: \sigma_p \rightarrow Y,$$

que es una aplicación entre los  $p$ -símplices singulares de ambos espacios.

Se verifica trivialmente que  $I_{\#} = \text{Id}$ , puesto que  $I_{\#}(\phi) = I \circ \phi = \phi$ . Además, se tiene que  $(g \circ f)_{\#} = g_{\#} \circ f_{\#}$ .

**Definición 1.3**

Sea  $p \geq 1$ ,  $0 \leq i < p$ . Definimos  $F_p^i$  como la aplicación de un  $(p-1)$ -símplice a un  $p$ -símplice de la siguiente forma:

$$\begin{aligned} F_p^i: \sigma_{p-1} &\rightarrow \sigma_p \\ (t_0, \dots, t_{p-1}) &\mapsto (t_0, \dots, t_{i-1}, 0, t_i, \dots, t_{p-1}) \end{aligned}$$

Podemos definir una nueva aplicación a partir de esta, a la que llamaremos *i-esima cara*:

$$\begin{aligned} \partial_i: F_p(\mathbb{X}) &\rightarrow F_{p-1}(\mathbb{X}) \\ \partial_i(\phi) &= \phi \circ F_p^i \end{aligned}$$

Vamos a probar un resultado sobre la aplicación que acabamos de definir.

**Lema 1.4**

Se verifica que  $F_p^i \circ F_{p-1}^j = F_p^j \circ F_{p-1}^{i-1} \quad \forall p \geq 2, \forall i, j: 0 \leq j < i \leq p$ .

*Demostración.* Lo comprobamos realizando los cálculos pertinentes:

$$\begin{aligned} F_p^i \circ F_{p-1}^j(t_0, \dots, t_{p-2}) &= F_p^i(t_0, \dots, t_{j-1}, 0, t_j, \dots, t_{p-2}) \\ &= (t_0, \dots, t_{j-1}, 0, t_j, \dots, t_{i-2}, 0, t_{i-1}, \dots, t_{p-2}) \\ &= F_p^j(t_0, \dots, t_{i-2}, 0, t_{i-1}, \dots, t_{p-2}) \\ &= F_p^j \circ F_{p-1}^{i-1}(t_0, \dots, t_{p-2}). \end{aligned}$$

□

Se deduce directamente este corolario:

**Corolario 1.5**

Para  $p \geq 2, 0 \leq j < i \leq p$ , se verifica  $\partial_j \circ \partial_i = \partial_{i-1} \circ \partial_j$ .

Debido a que  $F_p(\mathbb{X})$  no está dotado de estructura algebraica, es posible realizar la siguiente construcción:

**Definición 1.6**

Si  $\mathbb{X}$  es un espacio topológico, y  $p \geq 0$ , se define el **grupo de  $p$ -cadenas singulares de  $\mathbb{X}$**  como el grupo abeliano libre generado por  $F_p(\mathbb{X})$

$$S_p(\mathbb{X}) = \left\{ \sum_{\phi \in F_p(\mathbb{X})} n_{\phi} \cdot \phi \mid n_{\phi} \in \mathbb{Z}, \text{ todos cero salvo un número finito} \right\}.$$

Podemos definir un operador **borde**  $\partial$  con las aplicaciones cara  $\partial_i$  de la siguiente forma:

$$\begin{aligned}\partial: S_p(\mathbb{X}) &\rightarrow S_{p-1}(\mathbb{X}) \\ \partial(\phi) &= \sum_{i=0}^p (-1)^i \partial_i(\phi) \quad (p \geq 1)\end{aligned}$$

considerando  $\partial_i$  como una extensión de la aplicación cara a un homomorfismo:

$$\partial_i\left(\sum_{\phi \in F_p} n_\phi \cdot \phi\right) = \sum_{\phi \in F_p} n_\phi \cdot \partial_i(\phi)$$

Por conveniencia, consideraremos  $\partial(S_0(\mathbb{X})) = S_{-1}(\mathbb{X}) = \{0\}$ .

Veamos un resultado importante sobre la aplicación borde que acabamos de definir.

**Proposición 1.7**

$\partial \circ \partial: S_n(\mathbb{X}) \rightarrow S_{n-2}(\mathbb{X})$  es el homomorfismo cero.

Geométricamente, este resultado nos dice que el borde de una  $p$ -cadena es una  $(p-1)$ -cadena sin borde. Esta propiedad nos permitirá definir los grupos de homología singular.

*Demostración.* Consideramos  $S_p(\mathbb{X}) \xrightarrow{\partial} S_{p-1}(\mathbb{X}) \xrightarrow{\partial} S_{p-2}(\mathbb{X})$ .

Es claro que si  $p = 0, 1$ , la cuestión es trivial, pues  $S_{-1} = \{0\}$ . Tomaremos  $p > 1$ .

Sea  $\phi \in F_p(\mathbb{X})$ . Realizamos las operaciones necesarias:

$$\begin{aligned}\partial^2(\phi) &= \partial\left(\sum_{i=0}^p (-1)^i \partial_i(\phi)\right) = \sum_{i=0}^p (-1)^i \partial(\partial_i(\phi)) \\ &= \sum_{i=0}^p (-1)^i \sum_{j=0}^{p-1} (-1)^j \partial_j \partial_i(\phi) \\ &= \sum_{0 \leq j < i \leq p} (-1)^{i+j} \partial_j \partial_i(\phi) + \sum_{0 \leq i \leq j \leq p-1} (-1)^{i+j} \partial_j \partial_i(\phi) \\ &= \sum_{0 \leq j < i \leq p} (-1)^{i+j} \partial_{i-1} \partial_j(\phi) + \sum_{0 \leq i \leq j \leq p-1} (-1)^{i+j} \partial_j \partial_i(\phi) \\ &= \sum_{0 \leq j \leq i-1 \leq p} (-1)^{i+j} \partial_{i-1} \partial_j(\phi) + \sum_{0 \leq i \leq j \leq p-1} (-1)^{i+j} \partial_j \partial_i(\phi) = 0,\end{aligned}$$

pues son iguales salvo el signo.

□

Hasta ahora, hemos tomado un espacio topológico  $\mathbb{X}$  y le hemos asignado un conjunto de  $p$ -símplices singulares  $F_p(\mathbb{X})$ , del que hemos tomado un  $\mathbb{Z}$ -módulo, obteniendo el grupo abeliano libre de  $p$ -cadenas singulares,  $S_p(\mathbb{X})$ . Sobre él, hemos definido un operador borde  $\partial$ , que verifica  $\partial^2 = 0$ . A la familia  $\{S_p(\mathbb{X}), \partial\}_{p \geq 0}$  se le llama el complejo de cadenas singular asociado a  $\mathbb{X}$ .

Representaremos también por  $f_{\#}: S_p(\mathbb{X}) \rightarrow S_p(\mathbb{Y})$  a la composición entre  $f: \mathbb{X} \rightarrow \mathbb{Y}$ , continua, y  $\phi \in S_p(\mathbb{X})$ , de forma que:

$$f_{\#}\left(\sum_{\phi} n_{\phi} \cdot \phi\right) = \sum_{\phi} n_{\phi} \cdot (f \circ \phi).$$

Vamos a ver que esta aplicación conmuta con el borde, es decir,  $f_{\#} \circ \partial = \partial \circ f_{\#}$ . Esto hace conmutativo el siguiente diagrama:

$$\begin{array}{ccccc} \dots S_{p+1}(\mathbb{X}) & \xrightarrow{\partial} & S_p(\mathbb{X}) & \xrightarrow{\partial} & S_{p-1}(\mathbb{X}) \dots \\ \downarrow f_{\#} & & \downarrow f_{\#} & & \downarrow f_{\#} \\ \dots S_{p+1}(\mathbb{Y}) & \xrightarrow{\partial} & S_p(\mathbb{Y}) & \xrightarrow{\partial} & S_{p-1}(\mathbb{Y}) \dots \end{array}$$

Su demostración es casi inmediata, pues se tiene que

$$f_{\#} \circ \partial_i(\phi) = f_{\#}(\phi \circ F_p^i) = f \circ \phi \circ F_p^i = \partial_i(f \circ \phi) = \partial_i \circ f_{\#}(\phi).$$

El resultado obtenido implica, por la definición de  $\partial$ , que  $\partial f_{\#} = f_{\#} \partial$ . De hecho, es más fuerte, pues vemos como las aplicaciones cara conmutan una a una con el homomorfismo  $f_{\#}$ .

A esta aplicación entre complejos de cadenas se le llama en álgebra una aplicación de cadenas. Nos referiremos a ella como la aplicación de cadenas inducida por la aplicación continua  $f$ .

### Definición 1.8

Diremos que  $c \in S_p(\mathbb{X})$  es un ***p*-ciclo** si  $\partial(c) = 0$ .

Diremos que  $c \in S_p(\mathbb{X})$  es un ***p*-borde** si existe  $d \in S_{p+1}(\mathbb{X})$  tal que  $\partial(d) = c$ .

Representaremos por:

$$Z_p(\mathbb{X}) = \{c \in S_p(\mathbb{X}) \mid c \text{ es un } p\text{-ciclo}\},$$

$$B_p(\mathbb{X}) = \{c \in S_p(\mathbb{X}) \mid c \text{ es un } p\text{-borde}\}.$$

Se verifica que:

$$Z_p(\mathbb{X}) = \text{Ker}(\partial: S_p(\mathbb{X}) \rightarrow S_{p-1}(\mathbb{X})),$$

$$B_p(\mathbb{X}) = \text{Img}(\partial: S_{p+1}(\mathbb{X}) \rightarrow S_p(\mathbb{X})).$$

Ambos son subgrupos de  $S_p(\mathbb{X})$ , llamados grupos de los ciclos y los bordes, respectivamente. Como hemos visto que  $\partial^2 = 0$ , se deduce inmediatamente que  $B_p(\mathbb{X}) \subseteq Z_p(\mathbb{X}) \forall p \geq 0$ , luego se puede construir el grupo cociente.

$$H_p(\mathbb{X}) = \frac{Z_p(\mathbb{X})}{B_p(\mathbb{X})}$$

al que llamaremos el **p-ésimo grupo de homología singular**, el cual es un grupo abeliano.



**Nota 1.9**

Nótese que  $Z_0(\mathbb{X}) = S_0(\mathbb{X})$ , pues  $\partial(S_0(\mathbb{X})) = 0$ .

A los elementos de  $H_p(\mathbb{X})$  los notaremos  $[c]$ , con  $\partial(c) = 0$ , esto es,  $c \in Z_p(\mathbb{X})$ , y los llamaremos clases de homología.

La motivación geométrica para la construcción de los grupos de homología singular se basa en querer estudiar los ciclos en espacios topológicos. Como este conjunto tiene un tamaño demasiado grande para poder estudiarlo de forma efectiva, restringimos nuestra atención a las clases de equivalencia, haciendo que dos ciclos sean equivalentes si su diferencia es el borde de una cadena en una dimensión superior.

Tomemos nuevamente una función continua  $f: \mathbb{X} \rightarrow \mathbb{Y}$ , y definamos

$$f_*: H_p(\mathbb{X}) \rightarrow H_p(\mathbb{Y}) \text{ por } f_*([c]) = [f_\#(c)].$$

Comprobemos que está bien definida, esto es, que no depende del representante que se elija.

Sean  $[c_1] = [c_2]$ , lo que implica  $c_1 - c_2 \in B_p(\mathbb{X})$ . Por tanto, existe  $d \in S_{p+1}(\mathbb{X})$  tal que  $\partial(d) = c_1 - c_2$ . Así,  $f_\#(c_1) - f_\#(c_2) = f_\#(c_1 - c_2) = f_\#(\partial(d)) = \partial(f_\#(d))$ . Como  $f_\#(d) \in S_{p+1}(\mathbb{Y})$ , se tiene  $[f_\#(c_1)] = [f_\#(c_2)]$ .

Además,  $\partial(f_\#(c)) = f_\#(\partial(c)) = 0$ , por lo que  $f_\#(c)$  es un ciclo en  $\mathbb{Y}$ . Con esto hemos visto que  $f_*$  está bien definida, y es un homomorfismo, al serlo  $f_\#$ , al que llamaremos homomorfismo inducido por  $f$  en la homología.

De las propiedades de  $f_\#$  obtenemos directamente que:

- $(g \circ f)_* = g_* \circ f_*$ ,
- $I_* = \text{Id}$ .

Así, hemos construido un funtor covariante de la categoría de espacios topológicos en la categoría de grupos abelianos, el funtor de la homología singular.

$$\begin{array}{ccc} \mathbb{X} & \longrightarrow & H_p(\mathbb{X}) \\ \downarrow f & & \downarrow f_* \\ \mathbb{Y} & \longrightarrow & H_p(\mathbb{Y}) \end{array}$$

Si  $\mathbb{X}$  e  $\mathbb{Y}$  son homeomorfos, el homomorfismo que induce dicho homeomorfismo en la homología es un isomorfismo de grupos de homología, pues basta considerar el inverso del homeomorfismo para construir el inverso del homomorfismo inducido.

## 1.2 HOMOLOGÍA DE UN PUNTO

Vamos a calcular los grupos de homología singular de los puntos, sin más que ver como se comporta la función borde en ellos y utilizando la definición de los grupos de ciclos y bordes como núcleo e imagen de dicha aplicación.

Sea  $\mathbb{X}$  un espacio topológico formado por un punto,  $\mathbb{X} = \{x\}$ .  $F_p(\mathbb{X}) = \{\phi: \sigma_p \rightarrow \{x\}\} = \{\phi_p\}$  donde  $\phi_p$  es la función constante.

$S_p(\{x\})$  son grupos abelianos libres con  $\phi_p$  como generador. Además,  $\partial: S_p(\mathbb{X}) \rightarrow S_{p-1}(\mathbb{X})$  verifica:

$$\partial(\phi_p) = \sum_{i=0}^p (-1)^i \partial_i(\phi_p) = \sum_{i=0}^p (-1)^i \phi_{p-1} = \begin{cases} 0 & p \text{ es impar,} \\ \phi_{p-1} & p \text{ es par.} \end{cases}$$

Así, si  $p$  es impar,  $S_{p+1}(\{x\}) \rightarrow S_p(\{x\}) \xrightarrow{0} S_{p-1}(\{x\})$  y  $Z_p(\{x\}) = B_p(\{x\}) = S_p(\{x\})$ , luego  $H_p(\{x\}) = 0$ .

Si  $p$  es par, con  $p > 0$ ,  $Z_p(\{x\}) = B_p(\{x\}) = 0$ , y  $H_p(\{x\}) = 0$ .

Por último, si  $p = 0$ ,  $H_0(\{x\}) = \frac{S_0(\{x\})}{B_0(\{x\})} = S_0(\{x\}) \cong \mathbb{Z}$ .

Así,  $H_p(\{x\}) = 0 \quad \forall p \geq 1, \quad H_0(\{x\}) \cong \mathbb{Z}$ .

### 1.3 EL GRUPO $H_0(\mathbb{X})$ Y LA ARCOCONEXIÓN

En esta sección vamos a estudiar algunas de las propiedades de los grupos de homología singular, asumiendo que el espacio sobre el que trabajamos es arcoconexo.

#### Proposición 1.10

Sea  $\mathbb{X}$  un espacio topológico arcoconexo. Entonces  $H_0(\mathbb{X}) \cong \mathbb{Z}$ .

*Demostración.* Consideramos  $S_1(\mathbb{X}) \xrightarrow{\partial} S_0(\mathbb{X}) \rightarrow \{0\}$ . Tomamos  $H_0(\mathbb{X})$ .

$$H_0(\mathbb{X}) = \frac{Z_0(\mathbb{X})}{B_0(\mathbb{X})} = \frac{S_0(\mathbb{X})}{\text{Im } \partial}$$

Sea  $\epsilon: S_0(\mathbb{X}) \rightarrow \mathbb{Z}$  definido por

$$\epsilon\left(\sum_{\phi} n_{\phi} \phi\right) = \epsilon\left(\sum_x n_x \cdot x\right) = \sum_x n_x,$$

suma finita, por cómo definimos  $S_0$ .

Se verifica que  $\epsilon$  es un epimorfismo de grupos, pues podemos definir el conveniente elemento de  $S_0(\mathbb{X})$  para obtener cualquier entero. Calculamos el núcleo de la aplicación:

$$\epsilon \circ \partial(\phi_1) = \epsilon\left(\sum_{i=0}^1 \partial_i(\phi_1)\right) = 1 - 1 = 0.$$

Así,  $\text{Im } \partial \subseteq \text{Ker } \epsilon$ .

Sea ahora  $\sum n_x \cdot x \in \text{Ker } \epsilon$ , esto es,  $\sum n_x = 0$ .

$\sum n_x \cdot x = n_1 x_1 + \cdots + n_k x_k$  con  $\sum_{i=1}^k n_i = 0, x_i \in \mathbb{X}$ .

Sean  $\phi^i: \sigma_1 \rightarrow \mathbb{X}$  arcos tales que  $\phi^i(e_1) = x_i, \phi^i(e_0) = x$ , donde  $e_1, e_0$  representan los vértices de  $\sigma_1$  y  $x \in \mathbb{X}$  es un punto fijo. Es posible hacer esta construcción gracias a la arcoconexión de  $\mathbb{X}$ . De esta forma:

$$\sum_i n_i \phi^i \in S_1(\mathbb{X}), \text{ y se tiene}$$

$$\partial\left(\sum_i n_i \phi^i\right) = \sum_i n_i \partial(\phi^i) = \sum_i n_i (x_i - x) = \left(\sum_i n_i x_i\right) - \left(\sum_i n_i\right)x = \sum_i n_i x_i.$$

De aquí se deduce que  $\text{Ker } \epsilon \subseteq \text{Img } \partial$ , y por tanto tenemos la igualdad.

Finalmente como  $\text{Ker } \epsilon = \text{Img } \partial$ ,  $H_0(\mathbb{X}) = \frac{S_0(\mathbb{X})}{\text{Ker } \epsilon} \cong \mathbb{Z}$ , al ser  $\epsilon$  epimorfismo.  $\square$

**Proposición 1.11**

Sea  $\mathbb{X}$  un espacio topológico,  $\mathbb{X} = \bigcup_{\alpha \in A} \mathbb{X}_\alpha$  su descomposición en componentes arcoconexas.

Entonces se verifican:

- a)  $\forall \alpha \in A$ , la función  $i_{\alpha*}: H_p(\mathbb{X}_\alpha) \rightarrow H_p(\mathbb{X})$  es un monomorfismo, donde la aplicación  $i_\alpha: \mathbb{X}_\alpha \rightarrow \mathbb{X}$  representa la inclusión.
- b)  $H_p(\mathbb{X}) = \bigoplus_{\alpha \in A} i_{\alpha*}(H_p(\mathbb{X}_\alpha)) \quad \forall p \geq 0$ . En particular, por la inyectividad de  $i_{\alpha*}$ ,

$$H_p(\mathbb{X}) \cong \bigoplus_{\alpha \in A} H_p(\mathbb{X}_\alpha).$$

*Demostración.* Sea  $\phi: \sigma_p \rightarrow \mathbb{X}$  un  $p$ -símplice singular. Como  $\sigma_p$  es convexo y  $\phi$  es continua,  $\phi(\sigma_p) \subseteq \mathbb{X}_\alpha$ , para algún  $\alpha \in A$ , por lo que  $\phi = i_{\alpha*}(\phi)$  para dicho  $\alpha$ .

Si  $c = \sum n_\phi \cdot \phi$  es una  $p$ -cadena, poniendo cada  $\phi$  de la forma anterior y agrupando por componentes conexas:

$$c = \sum_{\alpha \in A} i_{\alpha*}(c_\alpha), \quad c_\alpha \in S_p(\mathbb{X}_\alpha). \quad (*)$$

Sea  $[c]_\alpha \in H_p(\mathbb{X}_\alpha)$ , y supongamos que  $i_{\alpha*}[c]_\alpha = 0$  en  $H_p(\mathbb{X})$ . En ese caso,  $[i_{\alpha*}(c)] = 0$  en  $H_p(\mathbb{X})$ , luego existe  $d \in S_{p+1}(\mathbb{X})$  tal que  $i_{\alpha\#}(c) = \partial(d)$ , y además:

$$\partial(d) = \sum_{\beta \in A} i_{\beta\#}(\partial(d_\beta)) = i_{\alpha\#}(c),$$

por lo que se tiene  $c = \partial(d_\alpha), \partial(d_\beta) = 0 \quad \forall \beta \neq \alpha$ . Así,  $[c]_\alpha = 0$  y  $i_{\alpha*}$  es un monomorfismo. Esto prueba a).

Observamos que la suma en (\*) es finita, pues si no, habría un número no finito de generadores en la expresión de  $c$ . Así:

$$[c] = \sum_{\alpha \in A} [i_{\alpha\#}(c_\alpha)] = \sum_{\alpha \in A} i_{\alpha*}([c_\alpha]_\alpha).$$

Como  $\partial(c) = 0 \iff \partial(c_\alpha) = 0 \quad \forall \alpha \in A$ , se obtiene directamente b), como consecuencia de a).  $\square$

En particular, para  $H_0(\mathbb{X})$ , por el resultado anterior, se tiene que  $H_0(\mathbb{X})$  es el grupo abeliano libre con tantos generadores como componentes arcoconexas tenga  $\mathbb{X}$ .

#### 1.4 INVARIANZA HOMOTÓPICA DE LA HOMOLOGÍA SINGULAR

En esta sección demostraremos la invarianza de los grupos de homología singular en espacios homotópicamente equivalentes, viendo antes como dos aplicaciones homotópicas inducen el mismo homomorfismo en la homología.

Vamos a comenzar demostrando el llamado Lema de Poincaré.

**Lema 1.12** (Lema de Poincaré)

Sea  $\mathbb{X} \subseteq \mathbb{R}^n$  estrellado desde algún punto. Entonces:

$$H_p(\mathbb{X}) = \begin{cases} \mathbb{Z} & \text{si } p = 0, \\ 0 & \text{si } p \neq 0. \end{cases}$$

*Demostración.* El caso  $p = 0$  está claro, pues un conjunto estrellado es, en particular, arcoconexo, por lo que solo hemos de aplicar el resultado ya demostrado. Suponemos  $p > 0$ .

Para demostrarlo, vamos a construir un homomorfismo  $T : S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{X})$  tal que  $T\partial + \partial T = \text{Id}_{S_p(\mathbb{X})}$  para  $p > 0$ .

De darse la situación anterior, si  $[c] \in H_p(\mathbb{X})$ ,  $c \in Z_p(\mathbb{X})$ , se tiene  $\partial(c) = 0$  y  $c \in S_p(\mathbb{X})$ . En consecuencia,  $c = (\partial T + T\partial)(c) = \partial(T(c))$  con  $T(c) \in S_{p+1}(\mathbb{X})$ . Por tanto,  $c$  es un borde, luego  $[c] = 0$ , y eso implica que  $H_p(\mathbb{X}) = 0 \quad \forall p > 0$ .

Definamos  $T$ :

Sean  $x_0 \in \mathbb{X}$  un punto de estrella,  $\phi : \sigma_p \rightarrow \mathbb{X}$  un  $p$ -símplice singular. Tomemos  $T(\phi) : \sigma_{p+1} \rightarrow \mathbb{X}$  dado por:

$$T(\phi)(t_0, \dots, t_{p+1}) = \begin{cases} x_0 & \text{si } t_0 = 1, \\ (1 - t_0)\phi(\frac{t_1}{1-t_0}, \dots, \frac{t_{p+1}}{1-t_0}) + t_0 x_0 & \text{si } t_0 < 1. \end{cases}$$

Como  $\mathbb{X}$  es estrellado,  $\text{Img}(T(\phi)) \subseteq \mathbb{X}$ .

Veamos que  $T(\phi)$  es continuo.

La continuidad está clara para  $t_0 < 1$ , por lo que la vemos para  $t_0 = 1$ . Se verifica que:

$$\begin{aligned} |T(\phi)(t_0, \dots, t_{p+1}) - x_0| &= |t_0 x_0 + (1 - t_0)\phi(\frac{t_1}{1-t_0}, \dots, \frac{t_{p+1}}{1-t_0}) - x_0| \\ &= (1 - t_0)|\phi(\frac{t_1}{1-t_0}, \dots, \frac{t_{p+1}}{1-t_0}) - x_0| \leq (1 - t_0)(c + |x_0|), \\ &\text{pues } \phi \text{ está acotada.} \end{aligned}$$

Así,  $\lim_{t_0 \rightarrow 1} T(\phi) = x_0$ .

Queda claro que  $\partial_0(T(\phi)) = \phi$ . Si seguimos llamando  $T$  al homomorfismo extensión de  $T$  a  $S_p(\mathbb{X})$ , tenemos construido

$$T: S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{X}) \text{ tal que } \partial_0 \circ T = I.$$

Si  $p = 0 \implies i = 1$  y  $\partial_1(T(\phi)) = x_0$ .

Si  $1 \leq i \leq p+1$ , entonces:

$$\begin{aligned} \partial_i(T(\phi)) &= (t_0, \dots, t_p) = T(\phi)(t_0, \dots, t_{i-1} \circ t_i, \dots, t_p) \\ &= (1 - t_0)\phi\left(\frac{t_1}{1 - t_0}, \dots, \frac{t_{i-1}}{1 - t_0}, \dots, \frac{t_p}{1 - t_0}\right) + t_0 x_0 \\ &= (1 - t_0)\partial_{i-1}\left(\frac{t_1}{1 - t_0}, \dots, \frac{t_p}{1 - t_0}\right) + t_0 x_0 = T(\partial_{i-1}\phi)(t_0, \dots, t_p). \end{aligned}$$

Como consecuencia:

$$\begin{aligned} \partial(T(\phi)) &= \sum_{i=1}^{p+1} (-1)^i \partial_i T(\phi) = \phi + \sum_{i=1}^{p+1} (-1)^i T \partial_{i-1}(\phi) \\ &= \phi - \sum_{j=0}^p (-1)^j T \partial_j(\phi) = \phi - T \partial(\phi). \end{aligned}$$

De donde se obtiene directamente que  $\partial T + T \partial = \text{Id}$ . □

**Teorema 1.13** (Invarianza homotópica de la homología singular)

Sean  $\mathbb{X}, \mathbb{Y}$  espacios topológicos y  $f, g: \mathbb{X} \rightarrow \mathbb{Y}$  aplicaciones continuas. Si  $f$  es homotópica a  $g$ , entonces  $f_* = g_*$ , siendo  $f_*, g_*: H_p(\mathbb{X}) \rightarrow H_p(\mathbb{Y}) \quad \forall p \geq 0$ .

*Demostración.* Supongamos que  $\forall p \geq 0$  podemos definir homomorfismos

$$T: S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{Y})$$

tales que  $T\partial + \partial T = f_{\#} - g_{\#}$ , esto es,  $f$  y  $g$  son homotópicas como aplicaciones de cadenas. Entonces,  $\forall [c] \in H_p(\mathbb{X})$  se tiene:

$$(f_* - g_*)([c]) = [f_{\#}(c) - g_{\#}(c)] = [(T\partial + \partial T)(c)] = [\partial(T(c))] = 0,$$

de donde se deduce  $f_* = g_*$ .

Veamos la definición de dicho  $T$ .

Sea  $H: \mathbb{X} \times I \rightarrow \mathbb{Y}$  la homotopía entre  $f$  y  $g$ , esto es,  $H(x, 0) = f(x)$ ,  $H(x, 1) = g(x)$ , donde  $I$  representa el intervalo  $[0, 1]$ .

Sean  $i_0, i_1: \mathbb{X} \rightarrow \mathbb{X} \times I$  dados por  $i_0(x) = (x, 0)$ ,  $i_1(x) = (x, 1)$ . Es claro que  $f = H \circ i_0$ ,  $g = H \circ i_1$ , de donde se tiene

$$f_{\#} - g_{\#} = H_{\#} \circ i_{0\#} - H_{\#} \circ i_{1\#} = H_{\#} \circ (i_{0\#} - i_{1\#}).$$

Si somos capaces de construir  $\tau: S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{X} \times I)$  tal que  $\partial\tau + \tau\partial = i_{0\#} - i_{1\#}$ , definiendo  $T = H_{\#} \circ \tau$ , se verifica:

$$\partial\tau + \tau\partial = \partial(H_{\#} \circ \tau) + (H_{\#} \circ \tau)\partial = H_{\#}(i_{0\#} - i_{1\#}) = f_{\#} - g_{\#},$$

lo que demostraría el teorema.

Construiremos  $\tau$  por inducción. Para ello, vamos a probar el siguiente resultado:

**Lema 1.14**

Para todo  $p \geq 0$  y para todo espacio topológico  $\mathbb{X}$ , existe un homomorfismo de grupos  $T^{\mathbb{X}}: S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{X} \times I)$  verificando:

- a)  $\partial T^{\mathbb{X}} + T^{\mathbb{X}}\partial = (i_0^{\mathbb{X}})_{\#} - (i_1^{\mathbb{X}})_{\#}$ ,
- b) (Condición de naturalidad) Para todo espacio topológico  $\mathbb{Y}$  y toda aplicación continua  $h: \mathbb{X} \rightarrow \mathbb{Y}$ , el diagrama

$$\begin{array}{ccc} S_p(\mathbb{X}) & \xrightarrow{T^{\mathbb{X}}} & S_{p+1}(\mathbb{X} \times I) \\ \downarrow h_{\#} & & \downarrow (h \times \text{Id})_{\#} \\ S_p(\mathbb{Y}) & \xrightarrow{T^{\mathbb{Y}}} & S_{p+1}(\mathbb{Y} \times I) \end{array} \quad \text{es conmutativo.}$$

Nótese que en b) pedimos más de lo requerido, pero nos sirve para definir  $T^{\mathbb{X}}$ .

Demostremoslo para  $p = 0$ .

Tomamos  $\mathbb{X} = \sigma_0$  e intentamos definir  $T^{\sigma_0}: S_0(\sigma_0) \rightarrow S_1(\sigma_0 \times I)$ .

Si  $\tau_0: \sigma_0 \rightarrow \sigma_0$  es la identidad, entonces  $\tau_0$  genera  $S_0(\sigma_0)$ . Como  $T^{\sigma_0}$  debe verificar  $\partial T^{\sigma_0} + T^{\sigma_0}\partial = (i_0^{\sigma_0})_{\#} - (i_1^{\sigma_0})_{\#}$ , se deduce  $\partial T^{\sigma_0}(\tau_0) = (i_0^{\sigma_0})_{\#}(\tau_0) - (i_1^{\sigma_0})_{\#}(\tau_0)$ .

Definiendo:

$$\begin{aligned} T^{\sigma_0}(\tau_0): \sigma_1 &\rightarrow \sigma_0 \times I \\ (t, 1-t) &\mapsto (\sigma_0, 1-t) \end{aligned}$$

es claro que verifica lo anterior.

Sea ahora  $\mathbb{X}$  espacio topológico, y sea  $\phi \in S_0(\mathbb{X})$  un generador,  $\phi: \sigma_0 \rightarrow \mathbb{X}$ . Entonces, como la construcción debe ser natural, el diagrama:

$$\begin{array}{ccc} S_0(\sigma_0) & \xrightarrow{T^{\sigma_0}} & S_1(\sigma_0 \times I) \\ \downarrow \phi_{\#} & & \downarrow (\phi \times \text{Id})_{\#} \\ S_0(\mathbb{X}) & \xrightarrow{T^{\mathbb{X}}} & S_1(\mathbb{X} \times I) \end{array}$$

debe ser conmutativo, luego  $T^{\mathbb{X}} \circ \phi_{\#} = (\phi \times \text{Id})_{\#} \circ T^{\sigma_0}$ .

Como  $\phi_{\#}(\tau_0) = \phi$ , se tiene  $T^{\mathbb{X}}(\phi) = (\phi \times \text{Id})_{\#} \circ T^{\sigma_0}(\tau_0)$ .

Definimos  $T^X$  de esta manera sobre los generadores, y lo extendemos por linealidad. Veamos que verifica a) y b).

a)

$$\begin{aligned}\partial T^X(\phi) &= \partial(\phi \times \text{Id})_{\#} T^{\sigma_0}(\tau_0) = (\phi \times \text{Id})_{\#} \partial T^{\sigma_0}(\tau_0) \\ &= (\phi \times \text{Id})_{\#} ((i_0)^{\sigma_0}(\tau_0) - (i_1)^{\sigma_0}(\tau_0)) \\ &= ((\phi \times \text{Id})(i_0)^{\sigma_0})_{\#} - ((\phi \times \text{Id})(i_1)^{\sigma_0})_{\#} (\tau_0) \\ &= ((i_0^X \circ \phi)_{\#} - (i_1^X \circ \phi)_{\#})(\tau_0) = ((i_0^X)_{\#} - (i_1^X)_{\#})(\phi),\end{aligned}$$

de donde se obtiene  $\partial T^X + T^X \partial = (i_0^X)_{\#} - (i_1^X)_{\#}$ .

b) Si  $h: X \rightarrow Y$  es continua, entonces:

$$\begin{aligned}(T^Y \circ h_{\#})(\phi) &= T^Y(h \circ \phi) = (h \times \text{Id})_{\#} T^{\sigma_0}(\tau_0) \\ &= ((h \times \text{Id})_{\#} \circ (\phi \times \text{Id})_{\#})(T^{\sigma_0}(\tau_0)) = ((h \times \text{Id})_{\#} \circ T^X)(\phi),\end{aligned}$$

lo que finaliza la demostración para  $p = 0$ .

Suponemos el resultado cierto para  $1, \dots, p-1$  y lo probamos para  $p$ . El razonamiento es similar al anterior.

Tomemos  $X = \sigma_p$ , y vamos a tratar de definir  $T^{\sigma_p}: S_p(\sigma_p) \rightarrow S_{p+1}(\sigma_p \times I)$ . De nuevo, como  $T^{\sigma_p}$  debe verificar a), se tiene que  $\forall \phi \in F_p(\sigma_p)$ ,  $T^{\sigma_p}(\phi)$  ha de satisfacer

$$\partial(T^{\sigma_p}(\phi)) = (i_0^{\sigma_p})_{\#}(\phi) - (i_1^{\sigma_p})_{\#}(\phi) - T^{\sigma_p}(\partial(\phi)),$$

donde  $T^{\sigma_p}(\partial(\phi))$  está bien definida por hipótesis de inducción.

Sea  $d = (i_0^{\sigma_p})_{\#}(\phi) - (i_1^{\sigma_p})_{\#}(\phi) - T^{\sigma_p}(\partial(\phi)) \in S_p(\sigma_p \times I)$ . Entonces:

$$\begin{aligned}\partial(d) &= (i_0^{\sigma_p})_{\#}(\partial(\phi)) - (i_1^{\sigma_p})_{\#}(\partial(\phi)) - \partial T^{\sigma_p}(\partial(\phi)) \\ &\stackrel{\text{h.i.}}{=} (i_0^{\sigma_p})_{\#}(\partial(\phi)) - (i_1^{\sigma_p})_{\#}(\partial(\phi)) - [-T^{\sigma_p}(\partial^2(\phi)) + (i_0^{\sigma_p})_{\#}(\partial(\phi)) - (i_1^{\sigma_p})_{\#}(\partial(\phi))] = 0\end{aligned}$$

y por tanto  $d$  es un ciclo.

Como  $p \geq 1$  y  $\sigma_p \times I$  es convexo, por el lema de Poincaré,  $d$  es un borde, luego existe  $e \in S_{p+1}(\sigma_p \times I)$  tal que  $\partial(e) = d$ . Definimos  $T^{\sigma_p}(\phi) = e$ . Notemos que si bien  $e$  no tiene por qué ser único, basta con hacer una elección entre los existentes.

Extendiendo por linealidad, conseguimos construir  $T^{\sigma_p}$  verificando a).

Sean ahora  $X, \phi: \sigma_p \rightarrow X$  u  $p$ -símplice singular. De nuevo, al tener que verificar b), el diagrama

$$\begin{array}{ccc} S_p(\sigma_p) & \xrightarrow{T^{\sigma_p}} & S_{p+1}(\sigma_p \times I) \\ \downarrow \phi_{\#} & & \downarrow (\phi \times \text{Id})_{\#} \\ S_p(X) & \xrightarrow{T^X} & S_{p+1}(X \times I) \end{array}$$

ha de ser conmutativo.

Si  $\tau_p: \sigma_p \rightarrow \sigma_p$  es la identidad (que es  $p$ -símplice singular en  $\sigma_p$ ),  $\phi_#(\tau_p) = \phi$ , de donde definimos  $T^X(\phi) = (\phi \times \text{Id})_\# T^{\sigma_p}(\tau_p)$ .

Como los  $\phi$  son la base de  $S_p(X)$ ,  $T^X$  se extiende por linealidad a un homomorfismo. Veamos que verifica a) y b).

a)

$$\begin{aligned}
 (\partial T^X + T^X \partial)(\phi) &= \partial((\phi \times \text{Id})_\# T^{\sigma_p}(\tau_p)) + T^X \partial(\phi_# \tau_p) \\
 &= (\phi \times \text{Id})_\# \partial T_p^{\sigma_p}(\tau_p) + T^X \phi_#(\partial(\tau_p)) \\
 &= (\phi \times \text{Id})_\# (-T^{\sigma_p} \partial(\tau_p) + (i_0^{\sigma_p})_\#(\tau_p) - (i_1^{\sigma_p})_\#(\tau_p)) + (\phi \times \text{Id})_\# T^{\sigma_p} \partial(\tau_p) \\
 &= (\phi \times \text{Id})_\# (i_0^{\sigma_p})_\#(\tau_p) - (\phi \times \text{Id})_\# (i_1^{\sigma_p})_\#(\tau_p) \\
 &= (i_0^X)_\# \phi_#(\tau_p) - (i_1^X)_\# \phi_#(\tau_p) = ((i_0^X)_\# - (i_1^X)_\#)(\phi).
 \end{aligned}$$

b) Sea  $h: X \rightarrow Y$  continua y  $\phi: \sigma_p \rightarrow X$  un generador de  $S_p(X)$ . Entonces:

$$\begin{aligned}
 T^Y(h_#(\phi)) &= T^Y(h \circ \phi) = ((h \circ \phi) \times \text{Id})_\# T^{\sigma_p}(\tau_p) \\
 &= (h \times \text{Id})_\# (\phi \times \text{Id})_\# T^{\sigma_p}(\tau_p) = (h \times \text{Id})_\# T^X(\phi).
 \end{aligned}$$

□

### Corolario 1.15

Si  $X$  e  $Y$  son espacios homotópicamente equivalentes, entonces

$$H_p(X) = H_p(Y) \quad \forall p \geq 0.$$

*Demostración.* Sea  $f$  la equivalencia homotópica de  $X$  en  $Y$ . Entonces existe su inversa,  $g$ , que verifica  $g \circ f = \text{Id}_X$ ,  $f \circ g = \text{Id}_Y$ . Así,  $(f \circ g)_* = \text{Id}_{H_p(Y)}$ , y  $(g \circ f)_* = \text{Id}_{H_p(X)}$ . Como  $(g \circ f)_* = g_* \circ f_*$ ,  $f_*$  tiene inverso, y por tanto  $f_*$  es un isomorfismo entre  $H_p(X)$  y  $H_p(Y)$ . □

## 1.5 HOMOLOGÍA SINGULAR DE UN PAR

Sea  $A \subseteq X$ . Vamos a tratar de cuantificar la relación existente entre  $H_*(A)$  y  $H_*(X)$ .

Para ello, vamos a definir la homología singular del par  $(X, A)$ , que será una extensión de la homología singular de un espacio, en el sentido  $H_*(X, \emptyset) = H_*(X)$ .

Consideremos  $i: A \hookrightarrow X$  la inclusión,  $i_\#: S_p(A) \rightarrow S_p(X)$  el homomorfismo inducido.

Entonces  $i_\#(\sum_{j=1}^n n_j \phi_j) = \sum_{j=1}^n n_j i_\#(\phi_j) = \sum_{j=1}^n n_j (i \circ \phi_j) = \sum_{j=1}^n n_j \phi_j$ , luego  $i_\#$  es un monomorfismo.

Definimos  $S_p(X, A) = \frac{S_p(X)}{i_\#(S_p(A))} = \frac{S_p(X)}{S_p(A)} = \{\bar{c} \mid c \in S_p(X)\}$  con  $\bar{c} = c + S_p(A)$ . Este grupo se corresponde con el grupo abeliano libre generado por  $\{\bar{\phi} \mid \phi \in F_p(X)\}$ , ya que  $\bar{c} = \sum_i n_i \bar{\phi}_i = \sum_i n_i \overline{\phi_i}$ ,  $c \in S_p(X)$ .



Llamamos a  $S_p(\mathbb{X}, A)$  el grupo de  $p$ -cadenas singulares del par  $(\mathbb{X}, A)$ . Podemos definir un operador borde, que seguiremos notando  $\partial$ :

$$\begin{aligned}\partial: S_p(\mathbb{X}, A) &\rightarrow S_{p-1}(\mathbb{X}, A) \\ \partial \bar{c} &= \overline{\partial c}.\end{aligned}$$

Este borde está bien definido, pues si  $\bar{c} = \bar{d}$ ; entonces  $d = c + a$ ,  $a \in S_p(A)$  y  $\partial d = \partial c + \partial a \implies \overline{\partial d} = \overline{\partial c} + \overline{\partial a} = \overline{\partial c}$  ya que  $\partial a \in S_{p-1}(A) \implies \overline{\partial a} = 0$ .

Es claro, por la definición que hemos hecho, que el operador borde es un homomorfismo de clases y que  $\partial^2 = 0$ .

Con esto, hemos construido un complejo de cadenas  $\{S_p(\mathbb{X}, A), \partial\}_{p \geq 0}$  al que llamaremos el complejo de cadenas singulares del par  $(\mathbb{X}, A)$ .

Para cada  $p \geq 0$  definimos:

$$Z_p(\mathbb{X}, A) = \{\bar{c} \in S_p(\mathbb{X}, A) \mid \partial \bar{c} = 0\} \text{ p-ciclos,}$$

$$B_p(\mathbb{X}, A) = \{\bar{c} \in S_p(\mathbb{X}, A) \mid \bar{c} = \partial \bar{d}\} \text{ p-bordes.}$$

Así, dado que de nuevo  $B_p(\mathbb{X}, A) \subseteq Z_p(\mathbb{X}, A)$ , podemos considerar el cociente

$$H_p(\mathbb{X}, A) = \frac{Z_p(\mathbb{X}, A)}{B_p(\mathbb{X}, A)} \quad \forall p \geq 0.$$

que llamamos el **p-ésimo grupo de homología singular** del par  $(\mathbb{X}, A)$ .

Dados dos pares  $(\mathbb{X}, A)$  y  $(\mathbb{Y}, B)$ , una aplicación continua  $f: (\mathbb{X}, A) \rightarrow (\mathbb{Y}, B)$  es una aplicación continua  $f: \mathbb{X} \rightarrow \mathbb{Y}$  tal que  $f(A) \subseteq B$ . Llamaremos a  $f$  una aplicación de pares entre  $(\mathbb{X}, A)$  e  $(\mathbb{Y}, B)$ .

Si  $f$  es tal aplicación, definimos:

$$\begin{aligned}f_{\#}: S_p(\mathbb{X}, A) &\rightarrow S_p(\mathbb{Y}, B) \\ f_{\#}(\bar{c}) &= \overline{f_{\#}(c)},\end{aligned}$$

que está bien definida.

Con esto, hemos hecho conmutativo el diagrama:

$$\begin{array}{ccc} S_p(\mathbb{X}) & \xrightarrow{f_{\#}} & S_p(\mathbb{Y}) \\ \downarrow \pi & & \downarrow \pi \\ S_p(\mathbb{X}, A) & \xrightarrow{f_{\#}} & S_p(\mathbb{Y}, B) \end{array} \quad \text{con } \pi(c) = \bar{c}.$$

Es claro que  $\partial f_{\#}(\bar{c}) = \overline{\partial f_{\#}(c)} = \overline{\partial f(c)} = f_{\#} \partial(c) = f_{\#} \partial(\bar{c})$ , de donde se deduce  $\partial f_{\#} = f_{\#} \partial$ .

A la función  $f_{\#}: S_*(\mathbb{X}, A) \rightarrow S_*(\mathbb{Y}, B)$  la llamamos la aplicación de cadenas inducida por la aplicación de pares  $f$ . Nuevamente, se verifica que  $(g \circ f)_{\#} = g_{\#} \circ f_{\#}$  y  $\text{Id}_{\#} = \text{Id}$ .

Definimos  $f_*: H_p(\mathbb{X}, A) \rightarrow H_p(\mathbb{Y}, B)$  por  $f_*([\bar{c}]) = [f_\#(\bar{c})]$ , que está bien definida. Claramente,  $f_*$  es un homomorfismo de grupos, que verifica  $(g \circ f)_* = g_* \circ f_*$  y  $\text{Id}_* = \text{Id}$ .

Así, hemos construido un funtor covariante de la categoría de pares y aplicaciones entre pares en la de grupos graduados y homomorfismos, el funtor de la homología singular.

$$\begin{array}{ccc} (\mathbb{X}, A) & \longrightarrow & \{H_p(\mathbb{X}, A)\}_{p \geq 0} \\ \downarrow f & & \downarrow f_* \\ (\mathbb{Y}, B) & \longrightarrow & \{H_p(\mathbb{Y}, B)\}_{p \geq 0} \end{array}$$

Tenemos un resultado de obtención directa:

**Corolario 1.16**

Si  $h: (\mathbb{X}, A) \rightarrow (\mathbb{Y}, B)$  es un homeomorfismo de  $\mathbb{X}$  sobre  $\mathbb{Y}$  que verifica  $h(A) = B$ , entonces  $h_*: H_p(\mathbb{X}, A) \rightarrow H_p(\mathbb{Y}, B)$  es un isomorfismo de grupos  $\forall p \geq 0$ .

*Demostración.* Basta considerar el inverso de  $h$ , que induce una aplicación inversa en la homología.  $\square$

Vamos a trasladar los resultados que vimos de arcoconexión para el caso de pares.

**Proposición 1.17**

Si  $\mathbb{X}$  es arcoconexo y  $A \neq \emptyset$ , se tiene  $H_0(\mathbb{X}, A) = 0$ .

*Demostración.* Sea  $[\bar{c}] \in H_0(\mathbb{X}, A)$ .  $\partial \bar{c} = 0$ , con  $c = \sum_{i=1}^n n_i x_i$   $x_i \in \mathbb{X}$ .

$\bar{c} = \sum_{i=1}^n n_i \bar{x}_i$  y así, tenemos  $[\bar{c}] = \sum_{i=1}^n n_i [\bar{x}_i]$ .

Veamos que  $\forall x \in \mathbb{X}$ ,  $[\bar{x}] = 0$ .

Como  $A \neq \emptyset$ , sea  $a \in A$ . Por arcoconexión, tomamos  $\alpha: \sigma_1 \rightarrow \mathbb{X}$  verificando  $\alpha(0, 1) = x, \alpha(1, 0) = a$ . Se cumple que  $\bar{\alpha} \in S_1(\mathbb{X}, A)$  y  $\partial \bar{\alpha} = \bar{\partial \alpha} = \overline{x - a} = \bar{x} - \bar{a} = \bar{x}$ , pues  $a \in S_0(A) \implies \bar{a} = 0$ .

De esta forma,  $[\bar{x}] = [\partial \bar{\alpha}] = 0$ , ya que es un borde.  $\square$

**Proposición 1.18**

Sea  $\mathbb{X} = \bigcup_{\alpha \in A} \mathbb{X}_\alpha$  la descomposición de  $\mathbb{X}$  en componentes arcoconexas, y sea  $A \subseteq \mathbb{X}$ . Entonces se verifica:

- a)  $\forall \alpha, i_{\alpha*}: H_p(\mathbb{X}_\alpha, \mathbb{X}_\alpha \cap A) \rightarrow H_p(\mathbb{X}, A)$  es un monomorfismo,
- b)  $H_p(\mathbb{X}, A) = \bigoplus_{\alpha \in A} i_{\alpha*}(H_p(\mathbb{X}_\alpha, \mathbb{X}_\alpha \cap A))$ .

La demostración es similar al caso visto para espacios simples. No la repetimos, pues no aporta nada nuevo.

Tenemos también un resultado equivalente al ya visto para aplicación homotópicas.

**Proposición 1.19**

Sean  $f, g: (\mathbb{X}, A) \rightarrow (\mathbb{Y}, B)$  aplicaciones de pares. Si existe  $H: \mathbb{X} \times I \rightarrow \mathbb{Y}$  continua tal que  $H(\_, 0) = f$ ,  $H(\_, 1) = g$  y  $H(A, t) \subseteq B \forall t \in I$ , entonces  $g_* = f_*$ . En tal caso, se dice que  $f$  y  $g$  son homotópicas.

La demostración es similar a la ya vista, por lo que no se incluye.

Consideremos ahora la proyección  $\pi: S_p(\mathbb{X}) \rightarrow S_p(\mathbb{X}, A) \cong \frac{S_p(\mathbb{X})}{i_*(S_p(A))}$ .

Tenemos que  $\pi(c) = \bar{c}$ , y la aplicación de cadenas inducida por la inclusión,  $i_\#: S_p(A) \rightarrow S_p(\mathbb{X})$ , que es claramente un monomorfismo.

$\pi$  es un epimorfismo, y se verifica que  $\text{Im } i_\# \cong \text{Ker } \pi = i_\#(S_p(A))$  por la definición de  $S_p(\mathbb{X}, A)$ . Esto nos permite definir la sucesión:

$$0 \rightarrow S_p(A) \xrightarrow{i_\#} S_p(\mathbb{X}) \xrightarrow{\pi} S_p(\mathbb{X}, A) \rightarrow 0,$$

que verifica  $\partial i_\# = i_\# \partial$  y  $\partial \pi = \pi \partial$ .

A estas sucesiones, en las que cada triple  $C \xrightarrow{f} D \xrightarrow{g} E$  verifica  $\text{Im } f = \text{Ker } g$ , las llamaremos exactas. Si además tienen solo tres elementos, como en el caso anterior, las llamaremos sucesiones exactas cortas.

Podemos extender el resultado anterior, y obtenemos una sucesión exacta corta de complejos de cadenas:

$$0 \rightarrow S_*(A) \xrightarrow{i_\#} S_*(\mathbb{X}) \xrightarrow{\pi} S_*(\mathbb{X}, A) \rightarrow 0.$$

Sea ahora  $\pi_*: H_*(\mathbb{X}) \rightarrow H_*(\mathbb{X}, A)$  el homomorfismo inducido por  $\pi$ .

$\pi_*([c]) = [\bar{c}]$  que está bien definido, pues si  $[c] = [d] \implies d = c + \partial h \implies \bar{d} = \bar{c} + \partial \bar{h} \implies [\bar{c}] = [\bar{d}]$ .

Veamos que  $\text{Im } i_* = \text{Ker } \pi_*$ .

Se verifica  $\pi_* i_*([a]) = \pi_*[i_\#(a)] = [\pi i_\#(a)] = 0$ , pues  $\pi i_\# = 0$ . Así,  $\text{Im } i_* \subseteq \text{Ker } \pi_*$ .

Si  $[c] \in \text{Ker } \pi_*$ ,  $\pi_*([c]) = [\bar{c}] = 0 \implies \bar{c} = \partial \bar{d}, c = \partial d + i_\#(a)$ , luego  $\partial c = \partial^2 d + \partial i_\#(a) \implies \partial c = \partial^2 d + i_\# \partial a$ , y se deduce que  $\partial a = 0$ .  $a$  es un ciclo y genera  $[a]$ . Como  $c = \partial d + i_\#(a)$ ,  $[c] = [\partial d] + [i_\#(a)] = i_*[a] \implies \text{Ker } \pi_* \subseteq \text{Im } i_*$ , lo que demuestra la igualdad.

Generalmente  $i_*$  no tiene por qué ser un monomorfismo, y  $\pi_*$  no tiene por qué ser un epimorfismo, aunque se verifica el siguiente resultado.

**Proposición 1.20**

$\pi_*: H_0(\mathbb{X}) \rightarrow H_0(\mathbb{X}, A)$  es un epimorfismo.

*Demostración.* Si tomamos  $[\bar{c}] \in H_0(\mathbb{X}, A)$  entonces  $c \in S_0(\mathbb{X}) = Z_0(\mathbb{X})$ , y por tanto  $[\bar{c}] \in H_0(\mathbb{X})$  y  $\pi_*([c]) = [\bar{c}]$ . □

**Nota 1.21**

Para  $p$  mayor que 1 no podemos asegurar que sea un ciclo, por lo que el resultado solo vale para el caso visto.

**Proposición 1.22**

Para todo  $p \geq 1$  existe un homomorfismo  $\Delta: H_p(\mathbb{X}, A) \rightarrow H_{p-1}(A)$  tal que  $\text{Img } \Delta = \text{Ker } i_*$  y  $\text{Ker } \Delta = \text{Img } \pi_*$ .

A la sucesión exacta así construída:

$$\dots H_{p+1}(\mathbb{X}, A) \xrightarrow{\Delta} H_p(\mathbb{X}, A) \xrightarrow{i_*} H_p(\mathbb{X}) \xrightarrow{\pi_*} H_p(\mathbb{X}, A) \xrightarrow{\Delta} H_{p-1}(A) \dots$$

se le llama la **sucesión exacta** del par  $(\mathbb{X}, A)$ .

*Demostración.* En primer lugar, vamos a definir  $\Delta$ .

Sea  $[\bar{c}] \in H_p(\mathbb{X}, A)$ , con  $\bar{c} \in S_p(\mathbb{X}, A)$  y  $\partial \bar{c} = 0$ . Como  $\partial \bar{c} = \overline{\partial c} = 0$ ,  $\partial c \in i_\#(S_{p-1}(A))$ . Por la inyectividad de  $i_\#$ , existe un único  $a \in S_{p-1}(A)$  tal que  $\partial c = i_\#(a)$ . Además,  $i_\# \partial a = \partial i_\#(a) = \partial^2 c = 0$ , y al ser  $i_\#$  monomorfismo,  $\partial a = 0$  luego  $a \in Z_{p-1}(A)$ ,  $[a] \in H_{p-1}(A)$ .

Definimos pues  $\Delta([\bar{c}]) = [a]$ , que está bien definido, ya que:

$$\begin{aligned} [\bar{c}] = [\bar{d}] &\implies \bar{d} = \bar{c} + \partial \bar{h} \implies d = c + \partial h + i_\#(e) \\ &\implies \partial d = \partial c + i_\#(\partial e) = i_\#(a) + i_\#(\partial e) = i_\#(a + \partial e) \\ &\implies \Delta([\bar{d}]) = [a + \partial e] = [a] = \Delta([\bar{c}]). \end{aligned}$$

Veamos que  $\Delta$  es homomorfismo, es decir,  $\Delta([\bar{c}_1] + [\bar{c}_2]) = \Delta([\overline{c_1 + c_2}])$ .

$$\begin{aligned} \Delta([\bar{c}_1]) &= [a_1] \text{ con } \partial c_1 = i_\#(a_1), \\ \Delta([\bar{c}_2]) &= [a_2] \text{ con } \partial c_2 = i_\#(a_2). \end{aligned}$$

De esta forma,  $\partial(c_1 + c_2) = i_\#(a_1 + a_2)$  y  $\Delta([\overline{c_1 + c_2}]) = [a_1 + a_2]$ .

Veamos que  $\text{Img } \Delta = \text{Ker } i_*$ .

$$i_*(\Delta([\bar{c}])) = i_*[a] = [i_\#(a)] = [\partial c] = 0, \text{ de donde } \text{Img } \Delta \subseteq \text{Ker } i_*.$$

Si  $[a] \in H_{p-1}(A)$  verificando  $i_*(a) = 0$ , entonces  $[i_\#(a)] = 0$ , y eso nos dice que  $i_\#(a) = \partial b$ ,  $b \in S_p(\mathbb{X})$ . Además,  $\overline{\partial b} = \overline{i_\#(a)} = 0$ , y como  $\overline{\partial b} = \partial \bar{b}$ ,  $b \in Z_p(\mathbb{X}, A)$  con  $\partial b = i_\#(a)$ . Por definición,  $\Delta([\bar{b}]) = [a]$ . Así se da la igualdad.

Veamos que  $\text{Ker } \Delta = \text{Img } \pi_*$ .

$$\Delta \pi_*([c]) = \Delta([\bar{c}]) = [a], \text{ con } \partial c = i_\#(a).$$

Como  $c$  es un ciclo,  $i_\#(a) = 0$ , y al ser  $i_\#$  un monomorfismo,  $a = 0$ , luego  $[a] = 0$ . Así,  $\text{Ker } \Delta \subseteq \text{Ker } \pi_*$ .

Si  $[\bar{c}] \in \text{Ker } \Delta$ ,  $\Delta[\bar{c}] = [\bar{a}] = 0$ , con  $\partial c = i_\#(a)$ . Como  $[a] = 0$ ,  $a = \partial b$ ,  $b \in S_p(A)$ . Así,  $\partial(c - i_\#(b)) = 0$ , y  $c - i_\#(b) \in Z_p(\mathbb{X})$ , de donde obtenemos  $[c - i_\#(b)]$ .

$$\pi_*([c \cdot i_\#(b)]) = \pi_*([c]) + e = [\bar{c}]. \text{ Con esto, } \text{Img } \pi_* = \text{Ker } \Delta. \quad \square$$

**Proposición 1.23** (Naturalidad de la sucesión exacta)

Sea  $f: (\mathbb{X}, A) \rightarrow (\mathbb{Y}, B)$  una aplicación de pares. Entonces el siguiente diagrama es conmutativo:

$$\begin{array}{ccccccc}
 \dots H_p(A) & \xrightarrow{i_*} & H_p(\mathbb{X}) & \xrightarrow{\pi_*} & H_p(\mathbb{X}, A) & \xrightarrow{\Delta} & H_{p-1}(A) \dots \\
 \downarrow (f|_A)_* & & \downarrow f_* & & \downarrow f_* & & \downarrow (f|_A)_* \\
 \dots H_p(B) & \xrightarrow{i_*} & H_p(Y) & \xrightarrow{\pi_*} & H_p(\mathbb{Y}, B) & \xrightarrow{\Delta} & H_{p-1}(B) \dots
 \end{array}$$

*Demostración.* Nos basta con probar que  $\Delta \circ f_* = f|_A \circ \Delta$ , pues ya conocemos el resto.

Realizamos el cálculo:

$$(\Delta \circ f_*)([c]) = \Delta([f_{\#}(\bar{c})]) = \Delta(\overline{[f_{\#}(c)]}) \stackrel{(*)}{=} [(f|_A)_{\#}(a)] = (f|_A)_*[a] = (f|_A)_*\Delta[\bar{c}].$$

Donde en (\*) estamos usando que  $\partial(f_{\#}(c)) = f_{\#}\partial c = f_{\#}i_{\#}(a) = i_{\#}(f|_A)_{\#}(a)$ , con  $\partial c = i_{\#}(a)$ .  $\square$

## CÁLCULO DE LA HOMOLOGÍA SINGULAR

---

En esta parte, vamos a enunciar y demostrar el teorema de subdivisión baricéntrica, el cual nos permite calcular directa o indirectamente (a través de sus consecuencias) los grupos de homología de ciertos espacios topológicos, como haremos con las esferas  $n$ -dimensionales.

### 2.1 TEOREMA DE SUBDIVISIÓN BARICÉNTRICA

#### Definición 2.1

Sea  $X$  un espacio topológico. Diremos que  $\mathcal{U} \subseteq \mathcal{P}(X)$  es un **recubrimiento** de  $X$  si  $X = \bigcup_{U \in \mathcal{U}} U$ .

Tomemos ahora  $X$  un espacio topológico y  $\mathcal{U}$  un recubrimiento suyo. Para  $p \geq 0$ , podemos definir:

$$F_p^{\mathcal{U}}(X) = \{\phi \in F_p(X) \mid \exists U \in \mathcal{U} \text{ tal que } \text{Im} \phi \subseteq U\}$$

que es, por definición, subconjunto de  $F_p(X)$ .

Si  $S_p^{\mathcal{U}}(X)$  es el grupo abeliano libre generado por  $F_p^{\mathcal{U}}(X)$ , por  $F_p^{\mathcal{U}}(X) \subseteq F_p(X)$ , se tiene que  $S_p^{\mathcal{U}}(X)$  es un subgrupo de  $S_p(X)$ . Si consideramos la inclusión  $i: S_p^{\mathcal{U}}(X) \rightarrow S_p(X)$ , si  $\phi \in F_p^{\mathcal{U}}(X)$ , entonces  $\partial_i(\phi) \in F_{p-1}^{\mathcal{U}}(X)$  y así, por la definición del borde,  $\partial(\phi) \in S_{p-1}^{\mathcal{U}}(X)$ .

En consecuencia, la restricción del borde a  $S_p^{\mathcal{U}}(X)$  define un homomorfismo borde, y así, construimos un complejo de cadenas  $S_*^{\mathcal{U}}(X) = \{S_p^{\mathcal{U}}(X), \partial\}_{p \geq 0}$ .

Es claro que este complejo es un subcomplejo de  $S_*(X)$ . Si tomamos ahora la inclusión  $i: S_*^{\mathcal{U}}(X) \rightarrow S_*(X)$ , esta induce un homomorfismo en la homología,  $i_*: H_p(S_*^{\mathcal{U}}(X)) \rightarrow H_p(X) \forall p \geq 0$ , donde  $H_p(S_*^{\mathcal{U}}(X)) = \frac{\text{Ker } \partial}{\text{Im} \partial}$ , con el borde actuando en  $S_*^{\mathcal{U}}(X)$ .

Sea ahora  $Y$  otro espacio topológico, y  $\mathcal{V}$  un recubrimiento suyo. Diremos que una función continua  $f: X \rightarrow Y$  es compatible con  $\mathcal{U}, \mathcal{V}$  si para todo  $U \in \mathcal{U}$  existe  $V \in \mathcal{V}$  tal que  $f(U) \subseteq V$ . En tal caso, puede definirse  $f_{\#}: S_p^{\mathcal{U}}(X) \rightarrow S_p^{\mathcal{V}}(Y)$ , homomorfismo, dado por la restricción de  $f_{\#}: S_p(X) \rightarrow S_p(Y)$  a  $S_p^{\mathcal{U}}(X)$ .

Esto hace que el siguiente diagrama sea conmutativo:

$$\begin{array}{ccc} S_p^{\mathcal{U}}(X) & \xrightarrow{f_{\#}} & S_p^{\mathcal{V}}(Y) \\ \downarrow i & & \downarrow i \\ S_p(X) & \xrightarrow{f_{\#}} & S_p(Y) \end{array}$$

**Nota 2.2**

El diagrama conmutativo está bien definido, pues si  $\phi \in F_p^{\mathbb{U}}(\mathbb{X})$ , entonces existe  $\mathbb{U} \in \mathbb{U}$  con  $\text{Img } \phi \subseteq \mathbb{U}$ , de donde  $f_{\#}(\phi) = f \circ \phi$  verifica  $\text{Img}(f \circ \phi) \subseteq f(\mathbb{U}) \subseteq \mathbb{V} \in \mathbb{V}$ .

Como  $f_{\#}\partial = \partial f_{\#}$ ,  $f$  induce  $f_*: H_p(S_p^{\mathbb{U}}(\mathbb{X})) \rightarrow H_p(S_p^{\mathbb{V}}(\mathbb{Y}))$ .

Nos encontramos en condiciones de enunciar el teorema de subdivisión baricéntrica, que relaciona complejo de cadenas de un espacio con un recubrimiento con el complejo de cadenas del espacio.

**Teorema 2.3** (Teorema de subdivisión baricéntrica)

Sean  $\mathbb{X}$  un espacio topológico y  $\mathbb{U}$  un recubrimiento de  $\mathbb{X}$  tal que  $\mathring{\mathbb{U}} = \{\mathring{\mathbb{U}} \mid \mathbb{U} \in \mathbb{U}\}$  también recubra a  $\mathbb{X}$ . Entonces la aplicación

$$i_*: H_p(S_p^{\mathbb{U}}(\mathbb{X})) \rightarrow H_p(\mathbb{X})$$

es un isomorfismo,  $\forall p \geq 0$ .

Además, si  $(\mathbb{Y}, \mathbb{V})$  es otro par de las mismas características, y  $f: \mathbb{X} \rightarrow \mathbb{Y}$  es una función continua compatible con  $\mathbb{U}, \mathbb{V}$ , entonces el siguiente diagrama es conmutativo:

$$\begin{array}{ccc} H_p(S_p^{\mathbb{U}}(\mathbb{X})) & \xrightarrow{i_*} & H_p(\mathbb{X}) \\ \downarrow f_* & & \downarrow f_* \\ H_p(S_p^{\mathbb{V}}(\mathbb{Y})) & \xrightarrow{i_*} & H_p(\mathbb{Y}) \end{array}$$

Antes de pasar a la demostración del teorema, necesitaremos introducir nuevos conceptos. La idea es definir la subdivisión de símplexes singulares mediante la subdivisión baricéntrica de símplexes, que nos llevan a poder dar una especie de inversa homotópica de la inclusión.

## 2.2 SUBDIVISIÓN BARICÉNTRICA DE SÍMPLES

Vamos a definir la subdivisión baricéntrica de símplexes, para lo que necesitaremos definir el baricentro de un símplex.

**Definición 2.4**

Sea  $s_p = (a_0, \dots, a_p)$  un  $p$ -símplex. Definimos el **baricentro** de  $s_p$  como

$$b(s_p) = \frac{a_0 + \dots + a_p}{p+1}.$$

Los puntos  $\{b(s_p), a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_p\}$  generan un  $p$ -símplex  $\forall i = 0, \dots, p$ . Definimos la **subdivisión baricéntrica** del símplex  $s_p$ , y la notaremos  $Sd(s_p)$ , por inducción sobre  $p$ :

- $Sd(s_0) = s_0$ .

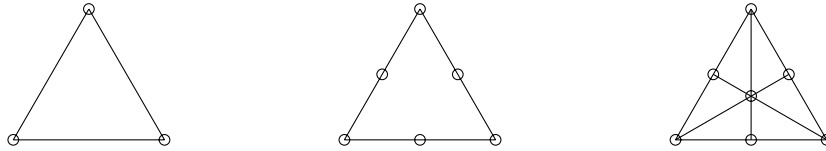
- $Sd(s_p) = (b(s_p), t_0, \dots, t_{p-1}) \mid (t_0, \dots, t_{p-1}) \in Sd(\mathbb{U}_{p-1}), \mathbb{U}_{p-1} \in \overset{\circ}{s}_p$  siendo  $\overset{\circ}{s}_p$  el conjunto de caras de orden  $p-1$  de  $s_p$ .

**Nota 2.5**

Es importante destacar que el orden en el que se ponen los puntos es importante, pues si el baricentro fuera en otro lugar, la orientación del símplexe cambiaría, y afectaría al cálculo del borde.

**Ejemplo 2.6**

Vemos una representación gráfica de la subdivisión baricéntrica, para dos casos distintos.

Figura 2.1:  $Sd(s_1)$ Figura 2.2:  $Sd(s_2)$ **Definición 2.7**

Si  $K$  es un conjunto de símplexes, la subdivisión de  $K$  viene dada por  $Sd(K) = \bigcup_{s_p \in K} Sd(s_p)$ . Se define la  $n$ -ésima subdivisión como

$$Sd^n(s_p) = Sd(Sd^{n-1}(s_p)), \text{ con } Sd^1 = Sd.$$

**Proposición 2.8**

Sea  $t_p \in Sd(s_p)$  un  $p$ -símplexe de la subdivisión. Entonces  $\text{diam}(t_p) \leq \frac{p}{p+1} \text{diam}(s_p)$ .

*Demostración.* Haremos la demostración por inducción sobre  $p$ .

Para  $p = 0$ , se tiene  $0 = 0$ .

Supuesto cierto hasta  $p-1$ , lo probamos para  $p$ .

Sea  $s_p = (a_0, \dots, a_p)$ . Se verifica que  $\text{diam}(s_p) = \max_{i,j} |a_i - a_j|$ .

Como  $t_p \in Sd(s_p)$ ,  $t_p = (b(s_p), t_0, \dots, t_{p-1}) \in Sd(\mathbb{U}_{p-1})$ . Entonces,  $\text{diam}(t_p) = \max\{|b(s_p) - t_i|, |t_i - t_j| \mid i, j = 0, \dots, p-1\}$ .

Por hipótesis de inducción, sabemos que

$$|t_i - t_j| \leq \text{diam}(t_0, \dots, t_{p-1}) \leq \frac{p-1}{p} \text{diam}(\mathbb{U}_{p-1}).$$



Como  $U_{p-1} \subseteq s_p$ , entonces  $\text{diam}(U_{p-1}) \leq \text{diam}(s_p)$ , y como  $\frac{p-1}{p} \leq \frac{p}{p+1}$  se tiene  $|t_i - t_j| \leq \frac{p}{p+1} \text{diam}(s_p)$ .

Además,  $|b(s_p) - t_i| \leq \max_i |b(s_p) - a_i| = \max_i \left| \frac{a_0 + \dots + a_p}{p+1} - a_i \right| = \max_i \frac{\sum_{j \neq i} |a_j - a_i|}{p+1} \leq \frac{p}{p+1} \text{diam}(s_p)$ .  $\square$

### Definición 2.9

Sea  $\varphi$  una colección de subconjuntos acotados de un espacio euclídeo. Definimos la **mall**a de dicha colección como  $\text{mall}\varphi = \sup_{c \in \varphi} \text{diam}(c)$ .

Del resultado anterior se obtienen directamente estos dos corolarios.

### Corolario 2.10

Si  $K$  es una colección de  $p$ -símplices, se verifica

$$\text{mall}(\text{Sd}(K)) \leq \frac{p}{p+1} \text{mall}(K).$$

### Corolario 2.11

Si  $K$  es una colección finita de  $p$ -símplices, entonces para todo  $\epsilon > 0$ , existe  $n \in \mathbb{N}$  tal que  $\text{mall}(\text{Sd}^n(K)) < \epsilon$ .

*Demostración.* Simplemente aplicamos  $\lim_{n \rightarrow \infty} \left(\frac{p}{p+1}\right)^n = 0$ .  $\square$

## 2.3 SUBDIVISIÓN BARICÉNTRICA DE SÍMPLES SINGULARES AFINES

Vamos a tratar ahora la subdivisión de símplices singulares afines. Estos nos permitirán aproximar la subdivisión de cualquier símplice singular, por lo que su estudio nos resulta de gran interés.

Sea  $C$  un subconjunto convexo del espacio euclídeo en el que trabajamos. Para todo  $p \geq 0$ , sea

$$F_p^A(C) = \{\phi \in F_p(C) \mid \phi \text{ es una aplicación afín}\},$$

es decir,  $\phi: \sigma_p \rightarrow C$  verifica

$$\phi(ta + (1-t)b) = t\phi(a) + (1-t)\phi(b) \quad \forall t \in [0, 1].$$

Sea  $A_p(C)$  el grupo abeliano libre generado por  $F_p^A(C)$ , subgrupo libre de  $S_p(C)$ . Si  $\phi \in F_p^A(C)$ ,  $\partial_i \phi = \phi \circ F_p^i$ , y como  $\phi$  es afín y  $F_p^i$  también lo es,  $\partial_i \phi$  es afín. Así:

$$\partial \phi = \sum_{i=0}^p (-1)^i \partial_i \phi \in A_{p-1}(C).$$

En consecuencia, podemos restringir el borde a  $A_p(C)$ , haciendo conmutativo el siguiente diagrama:

$$\begin{array}{ccc} A_p(C) & \xrightarrow{\partial} & A_{p-1}(C) \\ \downarrow i & & \downarrow i \\ S_p(C) & \xrightarrow{\partial} & S_{p-1}(C) \end{array}$$

Tenemos por tanto un complejo de cadenas  $A_*(C) = \{A_p(C), \partial\}_{p \geq 0}$ , válido para todo convexo  $C$ .

Si  $\phi \in F_p^A(C)$ ,  $\phi: \sigma_p \rightarrow C$  afín. Notando  $\sigma_p = (v_0, \dots, v_p)$ , entonces  $x_i = \phi(v_i) \in C$  determinan unívocamente a  $\phi$ , pues si  $v = \sum_{i=0}^p \alpha_i v_i$ , entonces  $\phi(v) = \sum_{i=0}^p \alpha_i \phi(v_i)$  por afinidad. Por tanto, todo elemento de  $\phi \in F_p^A(C)$  se identifica con  $p+1$  puntos de  $C$ , y usaremos ambas representaciones.

$$\phi \in F_p^A(C) \equiv \{x_0, \dots, x_p\} \subseteq C.$$

$$\text{Además, } \partial\phi = \sum_{i=0}^p (-1)^i \partial_i \phi, \text{ con } \partial_i \phi = \{x_0, \dots, \hat{x}_i, \dots, x_p\}.$$

En consecuencia,  $A_p(C)$  es el grupo abeliano libre sobre:

$$\{(x_0, \dots, x_p) \mid x_i \in C\} \equiv C \times \dots \times C \equiv C^{p+1}$$

$$\text{y el operador borde lo da } \partial(\{x_0, \dots, x_p\}) = \sum_{i=0}^p (-1)^i \{x_0, \dots, \hat{x}_i, \dots, x_p\}.$$

Si tenemos  $C, C'$  convexos, y  $f: C \rightarrow C'$  es una aplicación afín (y por tanto continua), al tomar la restricción de  $f_\#$  a  $A_p(C)$ , se tiene que si  $\phi \in F_p^A(C)$  entonces  $f_\#(\phi) = f \circ \phi \in F_p^A(C') \implies f_\#(A_p(C)) \subseteq A_p(C')$ .

En consecuencia,  $\partial f_\# = f_\# \partial$  y  $f_\#: A_*(C) \rightarrow A_*(C')$  es una aplicación de cadenas. Además,  $f_\#(\{x_0, \dots, x_p\}) = \{f(x_0), \dots, f(x_p)\}$ .

### Definición 2.12

Sea  $x \in C$ , con  $C$  un espacio cualquiera. Definimos

$$\begin{aligned} \mathcal{C}_x: A_p(C) &\rightarrow A_{p+1}(C) \\ \{x_0, \dots, x_p\} &\mapsto \{x, x_0, \dots, x_p\} \end{aligned}$$

y la extendemos por linealidad a un homomorfismo, pues tenemos los valores de los generadores.

### Proposición 2.13

Veamos que:

- a)  $\partial \mathcal{C}_x + \mathcal{C}_x \partial = \text{Id}_{A_p(C)} \quad \forall p \geq 1,$
- b)  $f_\# \mathcal{C}_x = \mathcal{C}_{f(x)} f_\# \quad \forall f: C \rightarrow C' \text{ afín.}$

$$\begin{aligned}
\text{Demostración. a) } \partial \mathcal{C}_x(x_0, \dots, x_p) &= \partial(x, x_0, \dots, x_p) = \{x_0, \dots, x_p\} + \sum_{i=1}^{p+1} (-1)^i \partial_i(x, x_0, \dots, x_p) \\
&= \{x_0, \dots, x_p\} + \sum_{j=0}^p (-1)^{j+1} \partial_{j+1}(x, x_0, \dots, x_p) = \{x_0, \dots, x_p\} - \sum_{j=0}^p (-1)^j \mathcal{C}_x(x_0, \dots, \hat{x}_j, \dots, x_p) \\
&= \{x_0, \dots, x_p\} - \mathcal{C}_x \partial(x_0, \dots, x_p).
\end{aligned}$$

$$b) f_{\#} \mathcal{C}_x(x_0, \dots, x_p) = f_{\#}(x, x_0, \dots, x_p) = (f(x), f(x_0), \dots, f(x_p)) = \mathcal{C}_{f(x)} f_{\#}(x_0, \dots, x_p). \quad \square$$

Ahora podemos definir una subdivisión para símlices o cadenas afines. Lo haremos por inducción, definiendo  $Sd': A_p(C) \rightarrow A_p(C)$

- $Sd' = \text{Id}_{A_p(C)}$  si  $p = 0$ ,
- $Sd'(\phi) = \mathcal{C}_{b(\phi)}(Sd' \circ \partial \phi)$ , supuesto definido hasta  $p - 1$ .

donde  $b(\phi) = \frac{x_0 + \dots + x_p}{p+1}$ , siendo  $\phi = \{x_0, \dots, x_p\}$  y extendido linealmente.

Veamos algunas propiedades. Debido a que la definición se hizo por inducción, todas las demostraciones serán por inducción.

$$a) \partial \circ Sd' = Sd' \circ \partial.$$

*Demostración.* Para  $p = 0$  la demostración es trivial, por la definición de  $Sd'$ . Suponemos  $p \geq 1$ .

Sea  $\phi \in F_p^A(C)$ , entonces

$$\begin{aligned}
\partial \circ Sd'(\phi) &= \partial \mathcal{C}_{b(\phi)} Sd' \partial \phi = Sd'(\partial \phi) - \mathcal{C}_{b(\phi)} \partial Sd'(\partial \phi) \\
&= Sd'(\partial \phi) - \mathcal{C}_{b(\phi)} Sd'(\partial^2 \phi) = Sd'(\partial \phi).
\end{aligned}$$

□

$$b) \text{ Si } f: C \rightarrow C' \text{ es afín, entonces } f_{\#} \circ Sd' = Sd' \circ f_{\#}.$$

*Demostración.* Hacemos el cálculo:

$$\begin{aligned}
(f_{\#} Sd')(\phi) &= f_{\#}(\mathcal{C}_{b(\phi)} Sd'(\partial \phi)) = \mathcal{C}_{f(b(\phi))} f_{\#} Sd'(\partial \phi) \\
&= \mathcal{C}_{f(b(\phi))} Sd'(f_{\#}(\partial \phi)) = \mathcal{C}_{f(b(\phi))} Sd'(\partial(f_{\#}(\phi))) = Sd'(f_{\#}(\phi)).
\end{aligned}$$

□

c) Existe un homomorfismo  $\tau': A_p(C) \rightarrow A_{p+1}(C)$  tal que  $\partial \tau' + \tau' \partial = Sd' - \text{Id}_{A_p(C)}$ , es decir,  $Sd'$  es, como aplicación de cadenas, homotópica a la identidad.

*Demostración.* Para  $p = 0$ , tomamos  $\tau' = 0$ .

Supuesto cierto hasta  $p - 1$ , buscamos  $\tau': A_p(C) \rightarrow A_{p+1}(C)$  verificando  $\partial \tau' = Sd' - \text{Id}_{A_p(C)} - \tau' \partial$ .

Como el segundo miembro es conocido, entonces si  $\phi \in F_p^A(C)$ ,  $\tau'(\phi)$  ha de verificar  $\partial(\tau'(\phi)) = Sd'(\phi) - \phi - \tau'(\partial(\phi))$ . Lo calculamos:

$$\begin{aligned}\partial(Sd'(\phi) - \phi - \tau'(\partial(\phi))) &= Sd'(\partial(\phi)) - \partial(\phi) - \partial(\tau'(\partial(\phi))) \\ &= Sd'(\partial(\phi)) - \partial(\phi) + \tau'(\partial^2(\phi)) - Sd'(\partial(\phi)) + \partial(\phi) = 0.\end{aligned}$$

Definiendo  $\tau'(\phi) = \mathcal{C}_{b(\phi)}(Sd'(\phi) - \phi - \tau'(\partial(\phi)))$ , entonces

$$\begin{aligned}\partial(\tau'(\phi)) &= \partial \mathcal{C}_{b(\phi)}(Sd'(\phi) - \phi - \tau'(\partial(\phi))) \\ &= (Sd'(\phi) - \phi - \tau'(\partial(\phi))) - \mathcal{C}_{b(\phi)} \partial(Sd'(\phi) - \phi - \tau'(\partial(\phi))) \\ &= Sd'(\phi) - \phi - \tau'(\partial(\phi)),\end{aligned}$$

lo que demuestra la igualdad que queríamos.  $\square$

d)  $\tau'$  es natural, esto es, si  $f: C \rightarrow C'$  es afín, entonces  $f_{\#}\tau' = \tau'f_{\#}$ .

*Demostración.* Para  $p = 0$  es trivial por definición.

Suponemos cierto hasta  $p - 1$ . Sea  $\phi \in F_p^A(C)$ , entonces

$$\begin{aligned}f_{\#}(\tau'(\phi)) &= f_{\#}(\mathcal{C}_{b(\phi)}(Sd'(\phi) - \phi - \tau'(\partial(\phi)))) \\ &= \mathcal{C}_{f(b(\phi))}f_{\#}(Sd'(\phi) - \phi - \tau'(\partial(\phi))) \\ &= \mathcal{C}_{f(b(\phi))}(Sd'(f_{\#}\phi) - f_{\#}\phi - \tau'(f_{\#}\partial(\phi))) = \tau'(f_{\#}(\phi))\end{aligned}$$

por lo que  $f_{\#}\tau' = \tau'f_{\#}$ .  $\square$

e) Propiedad técnica. Nos permitirá subdividir símlices hasta que estos queden dentro de un recubrimiento.

Sea  $\tau_p: \sigma_p \rightarrow \sigma_p$  la identidad ( $\tau_p \in F_p^A(\sigma_p)$ ).

Si  $Sd'(\tau_p) = \sum_i n_i \phi_i$  (que ocurre siempre), con  $\phi_i: \sigma_p \rightarrow \sigma_p$  afines, entonces  $\forall i \exists t_p^i \in Sd(\sigma_p)$  tal que  $\text{Img}(\phi_i) \subseteq t_p^i$ .

*Demostración.* Para  $p = 0$ ,  $Sd'(\tau_0) = \tau_0 = \sigma_0$ . Siendo  $t_0^i = \sigma_0 = Sd'(\sigma_0)$ .

Supuesto cierto hasta  $p - 1$ , sea  $Sd'(\tau_p) = \mathcal{C}_{b(\tau_p)}Sd'(\partial(\tau_p))$ .

$$\partial(\tau_p) = \sum_{i=0}^p (-1)^i \partial_i(\tau_p) = \sum_{i=0}^p (-1)^i \tau_p \circ F_p^i = (\text{por ser la identidad})$$

$$= \sum_{i=0}^p (-1)^i F_p^i \circ \tau_{p-1} = \sum_{i=0}^p (-1)^i (F_p^i)_{\#}(\tau_{p-1}). \text{ De esta forma:}$$

$$Sd'(\tau_p) = \mathcal{C}_{b(\tau_p)}\left(\sum_{i=0}^p (-1)^i Sd'(F_p^i)_{\#}(\tau_{p-1})\right) = \mathcal{C}_{b(\tau_p)}\left(\sum_{i=0}^p (-1)^i (F_p^i)_{\#}Sd'(\tau_{p-1})\right).$$

Si  $Sd'(\tau_{p-1}) = \sum_j m_j \psi_j$ , entonces, por inducción,  $\forall j$  existe  $t_{p-1}^j \in Sd(\sigma_{p-1})$  tal que

$$\text{Img}(\psi_j) \subseteq t_{p-1}^j. \text{ Por tanto, } Sd'(\tau_p) = \sum_{i=0}^p \sum_j (-1)^i m_j C_{b(\tau_p)}(F_p^i)_\#(\psi_j).$$

Como  $\text{Img}(\psi_j) \subseteq t_{p-1}^j \in Sd(\sigma_{p-1})$   $\text{Img}((F_p^i)_\#(\psi_j)) = \text{Img}(F_p^i \circ \psi_j) \subseteq F_p^i(\text{Img}(\psi_j)) \subseteq F_p^i(t_{p-1}^j) \in Sd(\sigma_p^\circ)$ , luego  $\text{Img}((F_p^i)_\#(\psi_j)) \subseteq U_{p-1}^{i,j} \in Sd(\sigma_p^\circ)$ .

Así,  $\text{Img}(C_{b(\tau_p)}(F_p^i)_\#(\phi_j)) \subseteq C_{b(\tau_p)}(U_{p-1}^{i,j}) = (b(\sigma_p), t_0, \dots, t_{p-1}) \in Sd(\sigma_p)$   
(donde  $U_{p-1}^{i,j} = (t_0, \dots, t_p)$ ). □

#### 2.4 SUBDIVISIÓN BARICÉNTRICA DE SÍMPLES SINGULARES

Utilizando el caso afín ya estudiado, definiremos una subdivisión de símlices singulares de forma general sobre un espacio topológico  $\mathbb{X}$ . Demostraremos una serie de propiedades, equivalentes a las vistas anteriormente.

##### Proposición 2.14

Sea  $\mathbb{X}$  un espacio topológico. Entonces, para todo  $p \geq 0$  existe un homomorfismo

$$Sd: S_p(\mathbb{X}) \rightarrow S_p(\mathbb{X})$$

verificando:

- a) Si  $\mathbb{X}$  es un convexo de un espacio euclídeo,  $Sd|_{A_p(\mathbb{X})} = Sd'$ ,
- b)  $Sd \circ \partial = \partial \circ Sd$ ,
- c)  $Sd \circ f_\# = f_\# \circ Sd \quad \forall f: \mathbb{X} \rightarrow \mathbb{Y}$  continua,
- d) Existe un homomorfismo  $\tau: S_p(\mathbb{X}) \rightarrow S_{p+1}(\mathbb{X})$  tal que  $\partial\tau + \tau\partial = Sd - \text{Id}_{S_p(\mathbb{X})} \quad \forall p \geq 0$   
y  $\tau f_\# = f_\# \tau \quad \forall f: \mathbb{X} \rightarrow \mathbb{Y}$  continua,
- e) Si  $\phi \in F_p(\mathbb{X})$  y  $Sd(\phi) = \sum_i n_i \phi_i$ , entonces  $\forall i \exists t_p^i \in Sd(\sigma_p)$  tal que  $\text{Img}(\phi_i) \subseteq \phi(t_p^i)$ .

*Demostración.* Sea  $\phi \in F_p(\mathbb{X})$ . Entonces, si  $\tau_p: \sigma_p \rightarrow \sigma_p$  es el  $p$ -símplexe identidad, se tiene que  $\phi = \phi_\#(\tau_p)$ . En consecuencia,  $Sd(\phi)$  debe verificar  $Sd(\phi_\#)(\tau_p) = \phi_\#(Sd(\tau_p)) = \phi_\#Sd'(\tau_p)$ .

Definimos  $Sd(\phi) = \phi_\#(Sd'(\tau_p))$  y lo extendemos por linealidad.

Veamos ahora las propiedades.

- a) Si  $\mathbb{X}$  es convexo,  $\phi \in F_p^A(\mathbb{X})$ , y entonces,  $Sd(\phi) = \phi_\#Sd'(\tau_p) = (Sd' \circ \phi_\#)(\tau_p) = Sd'(\phi)$  pues ya lo hemos probado en este caso.

b)

$$\begin{aligned}
(\partial \circ \text{Sd})(\phi) &= \partial \phi_{\#} \text{Sd}'(\tau_p) = \phi_{\#} \partial \text{Sd}'(\tau_p) = \phi_{\#} \text{Sd}'(\partial \tau_p) \\
&= \phi_{\#} (\text{Sd}'(\sum_{i=0}^p (-1)^i (F_p^i)_{\#}(\tau_{p-1}))) = \phi_{\#} (\sum_{i=0}^p (-1)^i \text{Sd}' \circ (F_p^i)_{\#}(\tau_{p-1})) \\
&= (\text{caso afín}) = \phi_{\#} (\sum_{i=0}^p (-1)^i (F_p^i)_{\#} \text{Sd}'(\tau_{p-1})) = \sum_{i=0}^p (-1)^i (\phi \circ F_p^i)_{\#} \text{Sd}'(\tau_{p-1}) \\
&= (\partial_i \phi)_{\#} (\sum_{i=0}^p (-1)^i \text{Sd}'(\tau_{p-1})) = \sum_{i=0}^p (-1)^i \text{Sd}(\partial_i \phi) = \text{Sd}(\sum_{i=0}^p (-1)^i \partial_i \phi) \\
&= (\text{Sd} \circ \partial)(\phi).
\end{aligned}$$

$$\text{c) } (f_{\#} \circ \text{Sd})(\phi) = f_{\#} \phi_{\#} \text{Sd}'(\tau_p) = (f \circ \phi)_{\#} \text{Sd}'(\tau_p) = \text{Sd}(f_{\#}(\phi)).$$

d) Definimos  $\tau(\phi) = \phi_{\#}(\tau'(\tau_p))$  con  $\phi_{\#}: S_{p+1}(\sigma_p) \rightarrow S_{p+1}(\mathbb{X})$ .

Entonces, se tiene:

$$\begin{aligned}
(\partial \tau + \tau \partial)(\phi) &= \partial(\phi_{\#} \tau'(\tau_p)) + \tau(\sum_{i=0}^p (-1)^i \partial_i \phi) \\
&= \phi_{\#}(\partial \tau'(\tau_p)) + \sum_{i=0}^p (-1)^i ((\partial_i \phi)_{\#}(\tau'(\tau_{p-1}))) \\
&= \phi_{\#}(\partial \tau'(\tau_p)) + \phi_{\#}(\sum_{i=0}^p (-1)^i (F_p^i)_{\#}(\tau'(\tau_{p-1}))) = (\text{caso afín}) \\
&= \phi_{\#}(\partial \tau'(\tau_p)) + \phi_{\#}(\tau'(\sum_{i=0}^p (-1)^i (F_p^i)_{\#}(\tau_{p-1}))) \\
&= \phi_{\#}(\partial \tau'(\tau_p)) + \phi_{\#}(\tau' \partial(\tau_p)) = \phi_{\#}(\partial \tau' + \tau' \partial)(\tau_p) = (\text{caso afín}) \\
&= \phi_{\#}(\text{Sd}' - \text{Id})(\tau_p) = \text{Sd}(\phi) - \phi.
\end{aligned}$$

e) Como  $\text{Sd}(\phi) = \phi_{\#}(\text{Sd}'(\tau_p))$ , entonces si  $\text{Sd}'(\tau_p) = \sum_i n_i \psi_i$ ,  $\phi_i = \phi_{\#}(\psi_i)$ .

Como existe  $t_p^i \in \text{Sd}(\sigma_p)$  tal que  $\text{Img}(\psi_i) \subseteq t_p^i$ , entonces

$$\text{Img}(\phi_i) = \text{Img}(\phi_{\#}(\psi_i)) = \phi(\text{Img}(\psi_i)) \subseteq \phi(t_p^i).$$

**Nota 2.15**

Probar que si  $\phi \in F_p(\mathbb{X})$  y  $\text{Sd}^n(\phi) = \sum_i n_i \phi_i \implies \forall i \exists t_p^i \in \text{Sd}^n(\sigma_p)$  tal que  $\text{Img}(\phi_i) \subseteq \phi(t_p^i)$ , consiste en reiterar la propiedad e) ya demostrada.

□

## 2.5 DEMOSTRACIÓN DEL TEOREMA

Demostremos ya el teorema de subdivisión baricéntrica.

**Teorema 2.16** (Teorema de subdivisión baricéntrica)

Sean  $X$  un espacio topológico y  $\mathcal{U}$  un recubrimiento de  $X$  tal que  $\mathring{\mathcal{U}} = \{\mathring{U} \mid U \in \mathcal{U}\}$  también recubra a  $X$ . Entonces la aplicación

$$i_*: H_p(S_*^{\mathcal{U}}(X)) \rightarrow H_p(X)$$

es un isomorfismo, para todo  $p \geq 0$ .

Además, si  $(Y, V)$  es otro par de las mismas características, y  $f: X \rightarrow Y$  es una función continua compatible con  $\mathcal{U}, V$ , entonces el diagrama que obtenemos considerando  $f_*$  e  $i_*$  es conmutativo.

*Demostración.* Sea  $X$  un espacio topológico, y sea  $\mathcal{U}$  un recubrimiento de  $X$  tal que  $\mathring{\mathcal{U}}$  también recubre a  $X$ .

Si  $\phi \in F_p(X)$ , esto es,  $\phi: \sigma_p \rightarrow X$  es continua, sea  $V = \{\phi^{-1}(U) \mid U \in \mathcal{U}\}$ , recubrimiento abierto de  $\sigma_p$  como espacio métrico compacto. Sea  $\epsilon(\phi)$  el número de Lebesgue de dicho recubrimiento. Así, si  $A \subseteq \sigma_p$  y  $\text{diam}(A) < \epsilon(\phi)$  entonces existe  $U \in \mathcal{U}$  tal que  $A \subseteq \phi^{-1}(U) \subseteq \phi^{-1}(U)$ , y entonces  $\phi(A) \subseteq U$ .

En consecuencia,  $\forall \phi \in F_p(X) \exists \epsilon(\phi) > 0$  tal que si  $A \subseteq \sigma_p$ ,  $\text{diam}(A) < \epsilon(\phi)$ , entonces existe  $U \in \mathcal{U}$  tal que  $\phi(A) \subseteq U$ .

Por un corolario ya visto, dado  $\epsilon(\phi)$ , existe  $n(\phi)$  tal que  $\text{malla}(\text{Sd}^{n(\phi)}(\sigma_p)) < \epsilon(\phi)$ . Si  $t_p \in \text{Sd}^{n(\phi)}(\sigma_p)$ ,  $\text{diam}(t_p) < \epsilon(\phi)$ , luego existe  $U \in \mathcal{U}$  con  $\phi(t_p) \subseteq U$ . En consecuencia, si  $\text{Sd}^{n(\phi)}(\phi) = \sum_i n_i \phi_i$ , entonces, por una propiedad vista,  $\text{Img}(\phi_i) \subseteq \phi(t_p^i)$  con  $t_p^i \in \text{Sd}^{n(\phi)}(\sigma_p)$ , por lo que existirá  $U_i \in \mathcal{U}$  con  $\text{Img}(\phi_i) \subseteq \phi(t_p^i) \subseteq U_i$ . Así,  $\text{Sd}^{n(\phi)}(\phi) \in S_p^{\mathcal{U}}(X)$ .

Sea  $m(\phi) = \inf\{n(\phi) \mid \text{Sd}^{n(\phi)}(\phi) \in S_p^{\mathcal{U}}(X)\}$ , que puede ser 0, si, por ejemplo,  $\phi \in S_p^{\mathcal{U}}(X)$ .

Definimos  $\psi: S_p(X) \rightarrow S_p^{\mathcal{U}}(X)$  por

$$\psi(\phi) = \text{Sd}^{m(\phi)}(\phi) - \sum_{i=0}^p (-1)^i \tau(\text{Sd}^{m(\partial_i(\phi))} + \dots + \text{Sd}^{m(\phi)-1}) \partial_i \phi,$$

y si  $m(\partial_i \phi) = m(\phi)$ , el sumando  $i$  no aparece en la expresión de  $\psi$ .

Hay que ver que está bien definido, para lo que comprobamos:

$$\text{i) } m(\partial_i(\phi)) \leq m(\phi),$$

$$\text{ii) } \tau(S_{p-1}^{\mathcal{U}}(X)) \subseteq S_p^{\mathcal{U}}(X).$$

$$\text{i) } \text{Sd}^n(\partial_i(\phi)) = \sum_{\alpha} n_{\alpha} \psi_{\alpha}, \text{ con } \text{Img}(\psi_{\alpha}) \subseteq (\partial_i \circ \phi)(t_{p-1}^{\alpha}), t_{p-1}^{\alpha} \in \text{Sd}^n(\sigma_{p-1}).$$

Como  $\partial_i \phi = \phi \circ F_p^i$ , se tiene:

$$\text{Img}(\phi_{\alpha}) \subseteq (\phi \circ F_p^i)(t_{p-1}^i) \subseteq \phi(U_{p-1}^{\alpha}) \subseteq \phi((b(\sigma_p), U_{p-1}^{\alpha})) = \phi(S_p^{\alpha}),$$

con  $S_p^{\alpha} \in \text{Sd}^n(\sigma_p)$ .

Si  $Sd^n(\phi) \in S_p^U(\mathbb{X})$ , entonces  $\phi(S_p^\alpha) \subseteq U_\alpha \in \mathbb{U}$ , luego se verifica

$$\text{Img}(\psi_\alpha) \subseteq U_\alpha \in \mathbb{U} \implies Sd^n(\partial_i \phi) \in S_{p-1}^U(\mathbb{X}).$$

De esta forma,  $m(\partial_i \phi) = \inf\{n(\phi) \mid Sd^n(\partial_i \phi) \in S_{p-1}^U(\mathbb{X})\} \leq m(\phi)$ .

ii) Como  $\tau(\phi) = \phi_\#(\tau'(\tau_p))$ , sea  $\phi: \sigma_{p-1} \rightarrow \mathbb{X}$  continua con  $\text{Img}(\phi) \subseteq U \in \mathbb{U}$ , esto es, un generador de  $S_{p-1}^U(\mathbb{X})$ . Entonces:

$$\tau(\phi) = \phi_\#(\tau'(\tau_p)) = \phi_\#(\sum_i n_i \phi_i) = \sum_i n_i \phi_\#(\phi_i).$$

Como  $\text{Img}(\phi_\#(\phi_i)) \subseteq \text{Img}(\phi) \subseteq U \in \mathbb{U}$ ,  $\tau(\phi) \in S_p^U(\mathbb{X})$ , lo que concluye la demostración de ii).

Una vez probada la buena definición de  $\psi$ , definimos:

$$\begin{aligned} T: S_p(\mathbb{X}) &\rightarrow S_{p+1}(\mathbb{X}) \\ T(\phi) &= \tau(Sd^0 + Sd + Sd^2 + \dots + Sd^{m(\phi)-1}). \end{aligned}$$

Vamos a ver que:

- a)  $\psi \circ i = \text{Id}_{S_p^U(\mathbb{X})}$ ,
- b)  $i \circ \psi - \text{Id}_{S_p(\mathbb{X})} = \partial T - T\partial$ .

En tal caso,  $i \circ \psi$  y la identidad serían aplicaciones de cadenas homotópicas, e inducirían los mismos homomorfismos en la homología. Esto, junto a que  $\psi \circ i = \text{Id}$ , nos dice que  $i_*: H_p^U(\mathbb{X}) \rightarrow H_p(\mathbb{X})$  es un isomorfismo para todo  $p$ , con inverso  $\psi_*$ . Concluimos la demostración probando a) y b).

a)  $(\psi \circ i)(\phi) = \psi(\phi)$  con  $\phi \in F_p^U(\mathbb{X})$ . Como  $m(\phi) = 0$  en este caso,  $\psi(\phi) = \phi - 0 = \phi$ .

b) Sabemos que  $\partial\tau + \tau\partial = Sd - \text{Id}$ , por lo que:

$$\partial\tau Sd - \tau Sd\partial = \partial\tau Sd - \tau\partial Sd = Sd^2 - Sd.$$

Repitiendo este proceso sucesivamente, se tiene:

$$\partial\tau Sd^{m-1} - \tau Sd^{m-1}\partial = Sd^m - Sd^{m-1}.$$

Si los sumamos todos,  $\partial\tau(I + Sd + \dots + Sd^{m-1}) - \tau(I + Sd + \dots + Sd^{m-1})\partial = Sd^m - \text{Id}$ . De esta forma:



$$\begin{aligned}
(\partial T + T\partial)(\phi) &= \partial\tau(I + Sd + \dots + Sd^{m-1})\phi + T\left(\sum_{i=0}^p (-1)^i \partial_i(\phi)\right) \\
&= \partial\tau(I + Sd + \dots + Sd^{m(\phi)-1})\phi + \sum_{i=0}^p (-1)^i \tau(I + \dots + Sd^{m(\partial_i(\phi))-1}) \partial_i \phi \\
&= Sd^{m(\phi)}(\phi) - \phi - \tau(I + Sd + \dots + Sd^{m(\phi)-1})(\partial\phi) \\
&\quad + \sum_{i=0}^p (-1)^i \tau(I + \dots + Sd^{m(\partial_i(\phi))-1}) \partial_i \phi \\
&= Sd^{m(\phi)}(\phi) - \phi - \sum_{i=0}^p (-1)^i \tau(I + \dots + Sd^{m(\phi)-1}) \partial_i \phi \\
&\quad + \sum_{i=0}^p (-1)^i \tau(I + \dots + Sd^{m(\partial_i(\phi))-1}) \partial_i \phi \\
&= Sd^{m(\phi)}(\phi) - \phi - \sum_{i=0}^p (-1)^i \tau(Sd^{m(\partial_i(\phi))} + \dots + Sd^{m(\phi)-1}) \partial_i \phi = \psi(\phi) - \phi,
\end{aligned}$$

de donde se sigue b). □

Veamos que la hipótesis de que  $\mathring{U}$  sea recubrimiento es esencial.

Sea  $\mathbb{X} = S^1$ ,  $\mathbb{U} = \{\{N\}, S^1 - \{N\}\}$  donde  $N$  representa el punto  $(0, 1)$ . Es claro que  $\mathbb{U}$  recubre a  $\mathbb{X}$ , pero no su interior.

Veamos que  $i_*: H_1(S_*^{\mathbb{U}}(S^1)) \rightarrow H_1(S^1)$  no es un isomorfismo.

Sea  $[c] \in H_1(S_*^{\mathbb{U}}(S^1))$ ,  $c \in S_1^{\mathbb{U}}(S^1)$ ,  $\partial c = 0$ . Entonces,  $c = c_1 + c_2$ , y si  $\phi: \sigma_1 \rightarrow \{N\}$  es la función (única) continua, entonces,  $c_1 = n\phi$ ,  $n \in \mathbb{Z}$ . Así,  $\partial = \partial c = n\partial\phi + \partial c_2 = \partial c_2$ , con  $c_2$  una 1-cadena en  $S^1 - \{N\} \cong \mathbb{R}$ , que es un ciclo.

Como  $H_1(\mathbb{R}) = 0$ , existe  $d_2$ , una 2-cadena en  $S^1 - \{N\}$  tal que  $\partial d_2 = c_2$ . Así,  $d_2$  es también cadena en  $S_2^{\mathbb{U}}(S^1)$ , y se tiene  $c = n\phi + \partial d_2 = n\partial\phi + \partial d_2 = \partial(n\phi d_2) \implies [c] = 0$  por ser un borde.

Por tanto,  $H_1(S_*^{\mathbb{U}}(S^1)) = 0$  y  $H_1(S^1) \cong \mathbb{Z}$ , que no son isomorfos.

## 2.6 CONSECUENCIAS DEL TEOREMA DE SUBDIVISIÓN BARICÉNTRICA

Vamos a tratar ahora algunas de las principales consecuencias del teorema de subdivisión baricéntrica.

**Teorema 2.17** (Sucesión de Mayer-Vietoris)

Sea  $\mathbb{X}$  un espacio topológico y  $\{U, V\}$  un recubrimiento de  $\mathbb{X}$  tal que  $\mathring{U} \cup \mathring{V} = \mathbb{X}$ . Consideramos las inclusiones:

$$\begin{array}{ll} i: U \cap V \rightarrow U & k: U \rightarrow \mathbb{X} \\ j: U \cap V \rightarrow V & l: V \rightarrow \mathbb{X} \end{array}$$

En este caso, existe un homomorfismo  $\Delta: H_p(\mathbb{X}) \rightarrow H_{p-1}(U \cap V)$  tal que la siguiente sucesión es exacta para todo  $p \geq 1$ :

$$\cdots \rightarrow H_p(U \cap V) \xrightarrow{G} H_p(U) \oplus H_p(V) \xrightarrow{H} H_p(\mathbb{X}) \xrightarrow{\Delta} H_{p-1}(U \cap V) \rightarrow \cdots$$

donde  $G([c]) = (i_*[c], j_*[c])$ ,  $H([c_1], [c_2]) = k_*([c_1]) - l_*([c_2])$ .

A esta sucesión la llamaremos la **sucesión de Mayer-Vietoris**. Nos referiremos a  $\Delta$  como el **homomorfismo de conexión** de la sucesión de Mayer-Vietoris.

Además, esta construcción es natural, es decir, si  $(\mathbb{Y}, \{R, S\})$  está en las condiciones del teorema y  $f: \mathbb{X} \rightarrow \mathbb{Y}$  es continua con  $f(U) \subseteq R, f(V) \subseteq S$ , entonces el diagrama resultante es conmutativo.

$$\begin{array}{ccccccc} H_p(U \cap V) & \xrightarrow{G} & H_p(U) \oplus H_p(V) & \xrightarrow{H} & H_p(\mathbb{X}) & \xrightarrow{\Delta} & H_{p-1}(U \cap V) \\ \downarrow f_* & & \downarrow f_* \oplus f_* & & \downarrow f_* & & \downarrow f_* \\ H_p(R \cap S) & \xrightarrow{G'} & H_p(R) \oplus H_p(S) & \xrightarrow{H'} & H_p(\mathbb{Y}) & \xrightarrow{\Delta'} & H_{p-1}(R \cap S) \end{array}$$

*Demostración.* Consideramos la siguiente sucesión:

$$\begin{aligned} 0 \rightarrow S_p(U \cap V) &\rightarrow S_p(U) \oplus S_p(V) \rightarrow S_p^U(\mathbb{X}) \rightarrow 0 \\ c &\mapsto (i_\#(c), j_\#(c)); (c_1, c_2) \mapsto k_\#(c_1) - l_\#(c_2) \end{aligned}$$

Veamos que es exacta.

Claramente,  $(i_\#, j_\#)$  es inyectiva, pues las aplicaciones  $i_\#, j_\#$  lo son.

Si  $d \in S_p^U(\mathbb{X})$ , podemos expresar  $d$  como una suma,  $d = d_1 + d_2$ , con  $d_1 = \sum_i n_i \phi_i$ ,  $d_2 = \sum_j m_j \psi_j$ ,  $\text{Im} \phi_i \subseteq U, \text{Im} \psi_j \subseteq V$ .

De esta forma,  $d_1 = k_\#(d_1), -d_2 = l_\#(d_2)$ , y  $d = k_\#(d_1) - l_\#(-d_2)$ . Es claro que  $k_\#(i_\#(c)) - l_\#(j_\#(c)) = 0$ , y, por último, si  $k_\#(c_1) - l_\#(c_2) = 0$ , y si  $c_1 = \sum_i n_i \phi_i, c_2 = \sum_j m_j \psi_j$ , entonces  $\text{Im}(\phi_i) \subseteq U \cap V, \text{Im}(\psi_j) \subseteq U \cap V$ , y así  $c_1 = i_\#(c), c_2 = j_\#(c)$ , de donde se sigue la exactitud.

Tenemos una sucesión exacta corta de complejos de cadenas:

$$0 \rightarrow S_*(U \cap V) \rightarrow S_*(U) \oplus S_*(V) \rightarrow S_*(\mathbb{X}) \rightarrow 0$$

donde el borde en  $S_*(U) \oplus S_*(V)$  lo da  $\partial \oplus \partial$ .

Usando el teorema de homología (que veremos más adelante), y teniendo en cuenta que  $H_p(S_*(U) \oplus S_*(V)) = H_p(U) \oplus H_p(V)$ , entonces existe un homomorfismo  $\bar{\Delta}: H_p(S_*^U(\mathbb{X})) \rightarrow H_{p-1}(U \cap V)$  tal que la siguiente sucesión es exacta:

$$\cdots \rightarrow H_p(U \cap V) \xrightarrow{G} H_p(U) \oplus H_p(V) \xrightarrow{\bar{H}} H_p(S_*^U(\mathbb{X})) \xrightarrow{\bar{\Delta}} \cdots$$

con  $\bar{H}([c_1], [c_2]) = [k_*(c_1) - l_*(c_2)]$ .

Aplicando el teorema de subdivisión baricéntrica,  $i_*: H_p(S_*^U(\mathbb{X})) \rightarrow H_p(\mathbb{X})$  es un isomorfismo.

Así, si  $\Delta = \bar{\Delta} \circ i_*^{-1}$ , entonces, como  $i_* \circ \bar{H} = H$ , la sucesión anterior se convierte en la sucesión de Mayer-Vietoris.

Como la sucesión exacta larga asociada a una corta es natural, se sigue directamente la naturalidad de la sucesión de Mayer-Vietoris.  $\square$

La sucesión de Mayer-Vietoris nos permite simplificar el cálculo de la homología singular de un espacio topológico  $\mathbb{X}$ , siempre que conozcamos los grupos de homología de los elementos de la descomposición. Esta cualidad nos resultará muy útil a la hora de calcular la homología de ciertos espacios, y en particular, la utilizaremos para el cálculo de la homología de las esferas.

Veamos ahora el teorema de homología, que utilizamos previamente para la demostración de la existencia de la sucesión de Mayer-Vietoris.

**Teorema 2.18** (Teorema de homología)

Sean  $\mathcal{E} = \{E_p, \partial\}$ ,  $\mathcal{F} = \{F_p, \partial\}$ ,  $\mathcal{G} = \{G_p, \partial\}$  complejos de cadenas.

Si  $0 \rightarrow \mathcal{E} \xrightarrow{f} \mathcal{F} \xrightarrow{g} \mathcal{G} \rightarrow 0$  es una sucesión exacta corta de complejos de cadenas, entonces existe un homomorfismo  $\Delta: H_p(\mathcal{G}) \rightarrow H_{p-1}(\mathcal{E})$  tal que la siguiente sucesión es exacta:

$$\cdots H_p(\mathcal{E}) \xrightarrow{f_*} H_p(\mathcal{F}) \xrightarrow{g_*} H_p(\mathcal{G}) \xrightarrow{\Delta} H_{p-1}(\mathcal{E}) \cdots$$

donde  $f_*([e]) = [f(e)]$ ,  $g_*([c]) = [g(c)]$ .

Además, la construcción así hecha es natural, esto es, si  $h_1, h_2$  y  $h_3$  son aplicaciones de cadenas tales que  $h_2 \circ f = f' \circ h_1$ ,  $h_3 \circ g = g' \circ h_2$ , esto es, el siguiente diagrama es conmutativo

$$\begin{array}{ccccccc} 0 & \longrightarrow & \mathcal{E} & \xrightarrow{f} & \mathcal{F} & \xrightarrow{g} & \mathcal{G} \longrightarrow 0 \\ & & \downarrow h_1 & & \downarrow h_2 & & \downarrow h_3 \\ 0 & \longrightarrow & \mathcal{E}' & \xrightarrow{f'} & \mathcal{F}' & \xrightarrow{g'} & \mathcal{G}' \longrightarrow 0 \end{array}$$

entonces el diagrama inducido en la homología es también conmutativo.

*Demostración.* Vamos a definir  $\Delta$ .

Sea  $[c] \in H_p(\mathcal{G})$  con  $c \in \mathcal{G}_p$  y  $\partial c = 0$ . Como  $g$  es sobreyectiva,  $c = g(d)$ ,  $d \in \mathcal{F}_p$ . Además,  $0 = \partial c = \partial g(d) = g(\partial d)$ , y así,  $\partial d \in \text{Ker } g = \text{Im } f$ , luego existe un único  $e \in \mathcal{E}_{p+1}$  tal

que  $f(e) = \partial d$ , con  $g(d) = c$ .

Definimos  $\Delta([c]) = [e]$ . Veamos que está bien definido.

Si  $[c] = [c']$ , existe  $d'$  con  $g(d') = c'$  y existe  $e'$  con  $f(e') = \partial d'$ . Como  $c - c' = \partial h$ ,  $h \in \mathcal{G}_{p+1}$ , entonces existe  $h' \in \mathcal{F}_{p+1}$  tal que  $g(h') = h$ . De esta forma:

$$g(d') - g(d) = \partial g(h') \implies g(d' - d - \partial h') = 0.$$

Por tanto, existe  $e'$  tal que  $f(e') = d' - d - \partial h$ . Así,  $\partial f(e') = f(d') - f(d) = f(e') - f(e) \implies e' - e = \partial e' \implies [e'] = [e]$ .

Comprobar la exactitud y la naturalidad es repetir el mismo proceso que hemos hecho en otras ocasiones.  $\square$

Demostrado el teorema de homología, vamos a enunciar y demostrar el teorema de escisión, un resultado que nos permitirá simplificar el cálculo de la homología de un par.

**Teorema 2.19** (Teorema de escisión)

Sea  $(X, A)$  un par,  $U \subseteq A$  verificando  $\bar{U} \subseteq \mathring{A}$ . Entonces la inclusión

$$k: (X - U, A - U) \rightarrow (X, A)$$

induce un isomorfismo en la homología, esto es,  $k_*: H_p(X - U, A - U) \rightarrow H_p(X, A)$  es un isomorfismo para todo  $p \geq 0$ .

*Demostración.* Sean  $U = \{X - U, A\}$ ,  $U' = \{A - U, \mathring{A}\}$ . Se verifica que  $X \overset{\circ}{-} U \cup \mathring{A} = X - \bar{U} \cup \mathring{A} \supseteq X - \mathring{A} \cup \mathring{A} = X$ , y, análogamente,  $(A \overset{\circ}{-} U)_A \cup \mathring{A}_A = A - \bar{U}_A \cup \mathring{A} = A - (\bar{U} \cap A) \cup \mathring{A} \supseteq A$ .

Así, por el teorema de subdivisión baricéntrica, las inclusiones

$$i: S_*^U(X) \rightarrow S_*(X)$$

$$i': S_*^{U'}(X) \rightarrow S_*(X)$$

inducen isomorfismos en la homología.

Consideremos ahora el siguiente diagrama:

$$\begin{array}{ccccccc} 0 & \longrightarrow & S_*^{U'}(A) & \longrightarrow & S_*^U(X) & \longrightarrow & \frac{S_*^U(X)}{S_*^{U'}(A)} \longrightarrow 0 \\ & & \downarrow i' & & \downarrow i & & \downarrow i'' \\ 0 & \longrightarrow & S_*(A) & \longrightarrow & S_*(X) & \longrightarrow & S_*(X, A) \longrightarrow 0 \end{array}$$

donde  $i''(c) = \overline{i(c)}$ , que está bien definido.

Del teorema de homología se sigue la conmutatividad del diagrama que induce el anterior en la homología:

$$\begin{array}{ccccccc}
 H_p(S_*^{\mathbb{U}'}(A)) & \longrightarrow & H_p(S_*^{\mathbb{U}}(\mathbb{X})) & \longrightarrow & H_p\left(\frac{S_*^{\mathbb{U}}(\mathbb{X})}{S_*^{\mathbb{U}'}(A)}\right) & \longrightarrow & H_{p-1}(S_*^{\mathbb{U}'}(A)) \dots \\
 \downarrow i'_* & & \downarrow i_* & & \downarrow i''_* & & \downarrow i'_* \\
 H_p(S_*(A)) & \longrightarrow & H_p(S_*(\mathbb{X})) & \longrightarrow & H_p(S_*(\mathbb{X}, A)) & \longrightarrow & H_{p-1}(S_*(A)) \dots
 \end{array}$$

Como  $i_*$ ,  $i'_*$  son isomorfismos, se deduce directamente que  $i''_*$  también lo es.

Sean ahora:

$j: S_*(\mathbb{X} - \mathbb{U}) \hookrightarrow S_*^{\mathbb{U}}(\mathbb{X})$  la inclusión, y

$j'': S_*(\mathbb{X} - \mathbb{U}, A - \mathbb{U}) \rightarrow \frac{S_*^{\mathbb{U}}(\mathbb{X})}{S_*^{\mathbb{U}'}(A)}$  el homomorfismo en el cociente.

$j''$  puede definirse ya que  $j(S_*(A - \mathbb{U})) \subseteq S_*^{\mathbb{U}'}(A)$ . Veamos que es isomorfismo.

Como  $\frac{S_*^{\mathbb{U}}(\mathbb{X})}{S_*^{\mathbb{U}'}(A)} = \frac{S_*(\mathbb{X} - \mathbb{U}) + S_*(\overset{\circ}{A})}{S_*(A - \mathbb{U}) + S_*(\overset{\circ}{A})} = \frac{S_*(\mathbb{X} - \mathbb{U})}{S_*(A - \mathbb{U})}$ , se tiene directamente el resultado, y así,  $j''$  es un isomorfismo.

Finalmente, debido a que  $k_{\#}: S_*(\mathbb{X} - \mathbb{U}, A - \mathbb{U}) \rightarrow S_*(\mathbb{X}, A)$  es dada por  $k_{\#} = i''_* \circ j''$ ,  $k_* = i''_* \circ j''_*$  es un isomorfismo.  $\square$

### 2.6.1 Homología de las esferas

Como última parte de este capítulo, vamos a calcular los grupos de homología de las esferas  $n$ -dimensionales. Lo haremos de dos formas: en la primera de ellas, utilizaremos resultados sobre pares y el teorema de escisión, mientras que en la segunda nos haremos valer de la sucesión de Mayer-Vietoris, llegando en ambas a las mismas conclusiones.

**Teorema 2.20** (Grupos de homología de las esferas)

Sea  $S^n$  la  $n$ -esfera en  $\mathbb{R}^{n+1}$ , de la que tomamos su representación asociada a la norma:

$$S^n = \{x = (x_1, \dots, x_{n+1}) \in \mathbb{R}^{n+1} \mid |x| = 1\}.$$

$$\text{Entonces se verifica que } H_p(S^n) = \begin{cases} \mathbb{Z} & \text{si } p = 0, n, \\ 0 & \text{en otro caso.} \end{cases}$$

*Demostración.* Ya conocemos el resultado sobre  $H_0$  por arcoconexión, por lo que nos centraremos en el resto.

Consideremos los dos hemisferios de las esferas:

$$E_+^n = \{x \in S^n \mid x_{n+1} \geq 0\}, \quad E_-^n = \{x \in S^n \mid x_{n+1} \leq 0\}.$$

Cada hemisferio es homeomorfo al disco cerrado  $D^n$  mediante la proyección al plano  $\{x \in \mathbb{R}^{n+1} \mid x_{n+1} = 0\}$ , por lo que es contráctil, y sus grupos de homología son los mismos que los de  $\mathbb{R}^n$ . Además, la intersección de ambos,  $E_+^n \cap E_-^n = \{x \in S^n \mid x_{n+1} = 0\}$  es homeomorfa a  $S^{n-1}$ .

Consideremos el par  $(S^n, E_+^n)$  y  $U = \{x \in S^n \mid x_{n+1} > \frac{1}{2}\} \subseteq S^n$  abierto. Como  $\bar{U} = \{x \in S^n \mid x_{n+1} \geq \frac{1}{2}\} \subseteq E_+^n$ , estamos en condiciones de aplicar el teorema de escisión, de donde se deduce:

$$H_*(S^n, E_+^n) \cong H_*(S^n - U, E_+^n - U).$$

Es posible construir un retracts de  $(S^n - U, E_+^n - U)$  a  $(E_-^n, S^{n-1})$ , por lo que sus homologías son isomorfas. Usando la equivalencia anterior, tenemos que  $H_*(S^n, E_+^n) \cong H_*(E_-^n, S^{n-1})$ .

Consideramos las sucesiones de los pares:

$$\begin{aligned} \cdots \rightarrow 0 &= H_p(E_+^n) \rightarrow H_p(S^n) \rightarrow H_p(S^n, E_+^n) \rightarrow H_{p-1}(E_+^n) = 0 \rightarrow \cdots \\ \cdots \rightarrow 0 &= H_1(E_+^n) \rightarrow H_1(S^n) \rightarrow H_1(S^n, E_+^n) \rightarrow H_0(E_+^n) = \mathbb{Z} \rightarrow \cdots \end{aligned}$$

Como  $H_p(E_+^n) = \begin{cases} 0 & \text{si } p > 0 \\ \mathbb{Z} & \text{si } p = 0 \end{cases}$ , utilizando que la sucesión es exacta, se tiene que

$$H_p(S^n) \cong H_p(S^n, E_+^n) \quad \forall p \geq 1.$$

Tomando la otra sucesión:

$$\begin{aligned} \cdots 0 &= H_p(E_-^n) \rightarrow H_p(E_-^n, S^{n-1}) \rightarrow H_{p-1}(S^{n-1}) \rightarrow H_{p-1}(E_-^n) = 0 \rightarrow \cdots \\ \cdots 0 &= H_1(E_-^n) \rightarrow H_1(E_-^n, S^{n-1}) \rightarrow H_0(S^{n-1}) \rightarrow H_0(E_-^n) \rightarrow \cdots \end{aligned}$$

$$\text{Con } H_0(S^{n-1}) = \begin{cases} \mathbb{Z} & \text{si } n \geq 2, \\ \mathbb{Z} \oplus \mathbb{Z} & \text{si } n = 1. \end{cases}$$

De esta forma, si  $p \geq 2$ ,  $H_p(E_-^n, S^{n-1}) \cong H_{p-1}(S^{n-1})$

Si  $p = 1$ , distinguimos dos casos:

- Si  $n = 1$ ,  $H_1(E_-^n, S^{n-1}) \cong \mathbb{Z}$ .
- Si  $n \geq 2$ ,  $H_1(E_-^n, S^{n-1}) = 0$ .

Como  $H_p(S^n, E_+^n) \cong H_{p-1}(E_-^n, S^{n-1})$ , se sigue que:

$$H_p(S^n) \cong H_{p-1}(S^{n-1}) \quad \forall p \geq 2$$

$$H_1(S^n) \cong H_1(E_-^n, S^{n-1}) = \begin{cases} 0 & \text{si } n \geq 2, \\ \mathbb{Z} & \text{si } n = 1. \end{cases}$$

Por tanto, distinguimos tres casos:

$$p = n \quad H_p(S^n) \cong H_{p-1}(S^{n-1}) \cong \dots \cong H_1(S^1) \cong \mathbb{Z}.$$

$$p > n \quad H_p(S^n) \cong \dots \cong H_{p-n}(S^0) = 0.$$

$$p < n \quad H_p(S^n) \cong \dots \cong H_1(S^{n-p+1}) = 0.$$

lo que demuestra el resultado.  $\square$

Veamos ahora una deducción alternativa utilizando la sucesión de Mayer-Vietoris.

*Demostración.* Tomemos nuevamente la  $n$ -esfera  $S^n$ , y sean  $U = S^n - (0, 0, \dots, 0, 1)$ ,  $V = S^n - (0, 0, \dots, 0, -1)$  abiertos de  $S^n$ .  $U$  y  $V$  forman un recubrimiento abierto de  $S^n$ , y son homeomorfos a  $\mathbb{R}^n$  mediante la proyección estereográfica, por lo que son contráctiles, y conocemos sus grupos de homología.

Si consideramos  $U \cap V$ , puede retraerse a  $S^{n-1} = \{x \in S^n \mid x_{n+1} = 0\}$ . Por tanto, se verifica:

$$\dots H_p(U) \oplus H_p(V) \rightarrow H_p(S^n) \rightarrow H_{p-1}(S^{n-1}) \rightarrow H_{p-1}(U) \oplus H_{p-1}(V) \dots$$

Si  $p \geq 2$ , sabemos que  $H_p(U), H_p(V) = 0$ . Por la exactitud de la sucesión, se tiene que  $H_p(S^n) \cong H_{p-1}(S^{n-1})$ .

Tomemos  $n \geq 2$ . Entonces, tenemos la sucesión exacta:

$$H_1(S^n) \xrightarrow{\Delta} \mathbb{Z} \xrightarrow{G} \mathbb{Z} \oplus \mathbb{Z} \xrightarrow{H} \mathbb{Z} \rightarrow 0$$

Utilizando la exactitud, deducimos  $\mathbb{Z} \cong \frac{\mathbb{Z} \oplus \mathbb{Z}}{\text{Ker } H} = \frac{\mathbb{Z} \oplus \mathbb{Z}}{\text{Img } G} \cong \mathbb{Z}$ .

$\text{Ker } G = 0 = \text{Img } \Delta$ , y así,  $H_1(S^n) \cong \text{Img } \Delta = 0$ .

Para  $n = 1$ :

$$0 \rightarrow H_1(S^1) \xrightarrow{\Delta} H_0(U \cap V) \xrightarrow{G} H_0(U) \oplus H_0(V) \xrightarrow{H} H_0(S^1) \rightarrow 0$$

Nuevamente, por exactitud, deducimos que  $H_1(S^1) \cong \text{Img } \Delta = \text{Ker } G$ .

Como  $U \cap V = S^1 - \{(0, 1), (0, -1)\}$ ,  $H_0(U \cap V) \cong \mathbb{Z} \oplus \mathbb{Z}$ , y sus elementos son de la forma  $[ax + by]$ , con  $a, b \in \mathbb{Z}$ ,  $x, y$  puntos en cada componente conexa de  $U \cap V$ . Se tiene que:

$$G([ax + by]) = (i_*(ax + by), j_*(ax + by)) = (0, 0), \text{ de donde obtenemos } [ax + by]_U = [ax + by]_V = 0.$$

Además,  $[ax + by]_U = [(a + b)x]_U$ , ya que  $[x]_U = [y]_U$  por ser  $U$  arcoconexo. Por tanto,  $a = -b$ .

Esto implica que  $\text{Ker } G = \{[ax - ay] \mid a \in \mathbb{Z}\} \cong \mathbb{Z}$ , y  $H_1(S^1) \cong \mathbb{Z}$ . Usando esto y que  $H_p(S^n) \cong H_{p-1}(S^{n-1})$ , se calcula la homología de  $S^n$  para todo  $n$ .  $\square$

## ALGUNOS RESULTADOS CLÁSICOS

---

En este capítulo, vamos a utilizar las herramientas del capítulo anterior y el cálculo de la homología de las esferas para obtener resultados de carácter topológico: la imposibilidad de retraer un disco en su borde, el teorema del punto fijo de Brower, el teorema de separación de Jordan-Brower y los teoremas de invarianza. Finalmente, introduciremos el concepto de **grado** para demostrar resultados sobre campos definidos en esferas de dimensión par.

### 3.1 TEOREMA DEL PUNTO FIJO DE BROWER

Comenzamos con un resultado consecuencia directa del cálculo de la homología de las esferas.

**Proposición 3.1**

a) *Esferas de distinta dimensión no tienen el mismo tipo de homotopía. En particular, no son homeomorfas.*

b) *Espacios euclídeos de distinta dimensión no son homeomorfos.*

*Demostración.* a) Sean  $S^n, S^m$  con  $n \neq m$ . Si  $S^n \cong S^m$ , entonces  $H_n(S^n) \cong H_n(S^m)$ , pero hemos visto que eso no sucede.

b) Si  $n \neq m$ , y  $h: \mathbb{R}^n \rightarrow \mathbb{R}^m$  es un homeomorfismo, entonces su restricción a  $\mathbb{R}^n - \{0\}$  también lo es. Como  $\mathbb{R}^n - \{0\} \cong S^n$ , y  $\mathbb{R}^m - \{h(0)\} \cong S^m$ , llegamos de nuevo a una contradicción.  $\square$

**Nota 3.2**

$\mathbb{R}^n$  y  $\mathbb{R}^m$  tienen el mismo tipo de homotopía. Si  $n \leq m$ , podemos ver  $\mathbb{R}^m$  como  $\mathbb{R}^m = \{(x, y) \mid x \in \mathbb{R}^n, y \in \mathbb{R}^{m-n}\}$ , y se define

$$H((x, y), t) = (1 - t)(x, y) + t(x, 0),$$

homotopía entre ambos espacios, siendo  $\mathbb{R}^n$  un retracto de  $\mathbb{R}^m$ .

**Lema 3.3**

No existe ninguna retracción de  $D^n$  a  $S^{n-1}$ .



*Demostración.* Sean  $D^n = \{x \in \mathbb{R}^n \mid |x| \leq 1\}$ ,  $S^{n-1} = \{x \in \mathbb{R}^n \mid |x| = 1\} \subseteq D^n$ . Si existiera dicha retracción,  $r: D^n \rightarrow S^{n-1}$ , tenemos el siguiente diagrama conmutativo:

$$\begin{array}{ccc} S^{n-1} & \xrightarrow{\text{Id}} & S^{n-1} \\ & \searrow i \quad \nearrow r & \\ & D^n & \end{array}$$

de donde se obtiene un diagrama, también conmutativo, en la homología:

$$\begin{array}{ccc} Z \cong H_{n-1}(S^{n-1}) & \xrightarrow{\text{Id}_*} & H_{n-1}(S^{n-1}) \\ & \searrow i_* \quad \nearrow r_* & \\ & H_{n-1}(D^n) & \end{array}$$

lo cual es imposible, pues  $H_{n-1}(D^n) \cong 0$ , y tendríamos una factorización de la identidad que incluye a la aplicación nula, que en este caso sería  $i_*$ .  $\square$

#### Corolario 3.4

Sea  $f: D^n \rightarrow \mathbb{R}^n$  continua, tal que la restricción de  $f$  a  $S^{n-1}$  sea la identidad. Entonces existe  $x \in D^n$  tal que  $f(x) = 0$ .

*Demostración.* Supongamos que no existe dicho  $x$ . En tal caso, es posible definir  $h(x) = \frac{f(x)}{|f(x)|}$  continua, que contradice el resultado anterior.  $\square$

Vamos a demostrar un teorema que nos dará como consecuencia directa un resultado clásico, el teorema del punto fijo de Brower.

#### Teorema 3.5

Sea  $f: D^n \rightarrow \mathbb{R}^n$  continua. Entonces o bien existe  $y \in D^n$  tal que  $f(y) = 0$ , o existe  $z \in S^{n-1}$  con  $f(z) = \lambda z$ ,  $\lambda < 0$ .

*Demostración.* Sea  $h: D^n \rightarrow \mathbb{R}^n$ , definida por:

$$h(x) = \begin{cases} f(2x) & \text{si } |x| \leq \frac{1}{2}, \\ (2 - 2|x|)f\left(\frac{x}{|x|}\right) + (2|x| - 1)x & \text{si } |x| \geq \frac{1}{2}, \end{cases}$$

que es continua, ya que en  $\frac{1}{2}$  ambas definiciones coinciden.

Si  $|x| = 1$ ,  $h(x) = x$ , y aplicando el corolario anterior, existe  $y \in D^n$  tal que  $h(y) = 0$ . Distinguimos dos casos:

Si  $|y| \leq \frac{1}{2}$ ,  $0 = f(2y)$ , y estamos en el primer caso.

Si  $|y| > \frac{1}{2}$ ,  $f\left(\frac{x}{|x|}\right) = \frac{(-2|x|+1)x}{2-2|x|}$ . Si llamamos  $z = \frac{x}{|x|}$ ,  $f(z) = \frac{(-2|x|+1)|x|}{2-2|x|}z$ , por lo que estamos en el segundo caso del teorema.  $\square$

**Corolario 3.6**

Sea  $f: D^n \rightarrow \mathbb{R}^n$  continua. Entonces existe  $y \in D^n$  tal que  $f(y) = y$  o existe  $z \in S^{n-1}$  con  $f(z) = \mu z$ ,  $\mu > 1$ .

*Demostración.* Aplicamos el teorema anterior a  $-f + \text{Id}$ . □

**Corolario 3.7** (Teorema del punto fijo de Brower)

Toda aplicación continua de  $D^n$  en sí mismo tiene un punto fijo.

*Demostración.* Basta ver que no puede darse la segunda opción del corolario anterior, pues la imagen de la función caería fuera de  $D^n$ . □

## 3.2 TEOREMAS DE INVARIANZA

Definamos el grupo de homología local, herramienta que nos servirá para demostrar varios resultados de invarianza.

**Definición 3.8**

Sean  $X$  un espacio topológico,  $x \in X$ . A  $H_p(X, X - \{x\})$  lo llamaremos el **grupo de homología local** de  $X$  en  $x$ .

**Proposición 3.9**

Sea  $x \in X$ , y supongamos que  $\{x\}$  es un cerrado. Entonces, para todo entorno  $V$  de  $x$ , se verifica:

$$H_p(X, X - \{x\}) \cong H_p(V, V - \{x\}) \quad \forall p \geq 0.$$

*Demostración.* Sea  $U = X - V \subseteq X - \{x\}$ . Si tomamos el cierre, se tiene

$$\bar{U} = \overline{X - V} = X - \overset{\circ}{V} \subseteq X - x = X - \bar{x} = (X - x)^\circ.$$

Aplicando el teorema de escisión,  $i: (X - U, (X - \{x\}) - U) \rightarrow (X, X - \{x\})$  induce un isomorfismo en la homología. Como  $X - U = V$ ,  $(X - \{x\}) - U = V - \{x\}$ , obtenemos el resultado. □

**Proposición 3.10**

Sean  $X, Y$  espacios topológicos,  $x \in X, y \in Y$  cerrados. Si existen entornos  $x \in V, y \in W$  tales que  $(V, \{x\}) \cong (W, \{y\})$ , entonces  $H_*(X, X - \{x\}) \cong H_*(Y, Y - \{y\})$ .

*Demostración.* Sea  $h: V \rightarrow W$  el homeomorfismo entre  $V$  y  $W$ , con  $h(x) = y$ . La restricción  $h: V - \{x\} \rightarrow W - \{y\}$  también es un homeomorfismo, por lo que  $H_*(V, V - \{x\}) \cong H_*(W, W - \{y\})$ . Basta aplicar la proposición anterior para obtener el resultado. □

Vamos a calcular la homología local de  $\mathbb{R}^n$ .

**Proposición 3.11**

$$H_p(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \cong \begin{cases} 0 & \text{si } n \neq p, \\ \mathbb{Z} & \text{si } n = p. \end{cases}$$

*Demostración.* Si  $p = 0$ , ya sabemos que  $H_0(\mathbb{R}^n, \mathbb{R}^n - \{x\}) = 0$  por la [Proposición 1.17](#). Supondremos  $p \geq 1$ .

Consideramos la sucesión de homología del par, teniendo en cuenta que  $\mathbb{R}^n - \{x\}$  y  $S^{n-1}$  tienen el mismo tipo de homotopía.

Si  $p = 1$ :

$$0 = H_1(\mathbb{R}^n) \xrightarrow{\pi_*} H_1(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \xrightarrow{\Delta} H_0(\mathbb{R}^n - \{x\}) \xrightarrow{i_*} H_0(\mathbb{R}^n) = \mathbb{Z} \rightarrow 0.$$

- Si  $n \geq 2$ ,  $H_0(\mathbb{R}^n - \{x\}) \cong \mathbb{Z}$ . Como  $\text{Im} \Delta = \text{Ker } i_* = 0$ ,  $\text{Ker } \Delta = \text{Im } \pi_* = 0$ , entonces se tiene que  $H_1(\mathbb{R}^n, \mathbb{R}^n - \{x\}) = 0$ .
- Si  $n = 1$ ,  $H_0(\mathbb{R}^n - \{x\}) \cong \mathbb{Z} \oplus \mathbb{Z}$ , luego  $H_1(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \cong \mathbb{Z}$ .

Si  $p \geq 2$ :

$$0 = H_p(\mathbb{R}^n) \xrightarrow{\pi_*} H_p(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \xrightarrow{\Delta} H_{p-1}(\mathbb{R}^n - \{x\}) \xrightarrow{i_*} H_{p-1}(\mathbb{R}^n) = 0.$$

- Si  $n \neq p$ ,  $H_{p-1}(\mathbb{R}^n - \{x\}) = 0$ , luego  $H_p(\mathbb{R}^n, \mathbb{R}^n - \{x\}) = 0$ .
- Si  $n = p$ ,  $H_{p-1}(\mathbb{R}^n - \{x\}) \cong \mathbb{Z}$ , luego  $H_p(\mathbb{R}^n, \mathbb{R}^n - \{x\}) = \mathbb{Z}$ .

□

**Teorema 3.12** (Invarianza de la dimensión)

Sean  $O_1 \subseteq \mathbb{R}^n, O_2 \subseteq \mathbb{R}^m$  abiertos homeomorfos. Entonces  $n = m$ .

*Demostración.* Sea  $\phi: O_1 \rightarrow O_2$  un homeomorfismo entre los abiertos, y sea  $x \in O_1$ . Como  $O_1, O_2$  son entornos de  $x, \phi(x)$ , por la proposición anterior:

$$H_*(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \cong H_*(\mathbb{R}^m, \mathbb{R}^m - \{\phi(x)\}),$$

y en particular

$$\mathbb{Z} \cong H_n(\mathbb{R}^n, \mathbb{R}^n - \{x\}) \cong H_n(\mathbb{R}^m, \mathbb{R}^m - \{\phi(x)\}),$$

de donde se deduce  $n = m$ .

□

**Corolario 3.13**

Si  $M$  es un espacio topológico  $T_2$ , conexo, y verificando que para cada  $p \in M$  existen  $V_p$  entorno de  $p$ , un entero  $n(p) \geq 1$  y un homeomorfismo  $\phi_p: V_p \rightarrow O_p \subseteq \mathbb{R}^{n(p)}$  con  $O_p$  abierto, entonces  $n(p) = n$  constante y  $M$  es una variedad topológica  $n$ -dimensional.

*Demostración.* Sea  $p_0 \in M$  fijo. Definimos el conjunto de puntos con el mismo valor de  $n$  que  $p_0$ ,  $A = \{p \in M \mid n(p) = n(p_0)\}$ , que es no vacío, pues  $p_0 \in A$ . Es claro que si  $q \in V_p$ ,  $n(q) = n(p)$ , pues si no fuera así, se contradice el teorema de invarianza de la dimensión. Así, si  $p \in A$ ,  $V_p \subseteq A$  y por tanto  $A$  es un abierto en  $M$ .

Si  $p \in \bar{A}$ , existe  $q \in A \cap V_p$  con  $n(q) = n(p_0)$  por  $q \in A$ , y  $n(q) = n(p)$  por  $q \in V_p$ . Por tanto,  $n(p) = n(p_0)$ ,  $p \in A$ , y  $A$  es cerrado en  $M$ . En consecuencia,  $A = M$ . □

Pasemos ahora a un resultado que nos permitirá demostrar el teorema de invarianza del borde.

**Lema 3.14**

Sean  $H^n = \{x \in \mathbb{R}^n \mid x_n \geq 0\}$  semiespacio de  $\mathbb{R}^n$ ,  $\partial H^n = \text{Fr} H^n = \{x \in \mathbb{R}^n \mid x_n = 0\}$  el borde de dicho semiespacio, y  $\overset{\circ}{H}^n = \{x \in \mathbb{R}^n \mid x_n > 0\}$  su interior. Entonces, se verifica

$$H_p(H^n, H^n - \{x\}) = \begin{cases} \mathbb{Z} & \text{si } p = n, x \in \overset{\circ}{H}^n, \\ 0 & \text{si } p \neq n, x \in \overset{\circ}{H}^n, \\ 0 & \text{si } x \in \partial H^n. \end{cases}$$

*Demostración.* Si  $x \in \overset{\circ}{H}^n$ , por una proposición anterior, sabemos que  $H_p(H^n, H^n - \{x\}) \cong H_p(\overset{\circ}{H}^n, \overset{\circ}{H}^n - \{x\}) \cong H_p(\mathbb{R}^n, \mathbb{R}^n - \{x\})$ , pues  $\overset{\circ}{H}^n \cong \mathbb{R}^n$ . Con esto obtenemos los valores para  $x \in \overset{\circ}{H}^n$ .

Si  $x \in \partial H^n$ ,  $H^n - \{x\}$  es estrellado desde un punto. Usando esto y la sucesión de homología del par  $(H^n, H^n - \{x\})$ , se obtiene la última parte.  $\square$

**Corolario 3.15 (Invarianza del borde)**

Sean  $O_1 \subseteq H^n, O_2 \subseteq H^m$  abiertos homeomorfos. Entonces

- a) Si  $\phi: O_1 \rightarrow O_2$  es un homeomorfismo,  $\phi(O_1 \cap \partial H^n) = O_2 \cap \partial H^m$ ,
- b)  $n = m$ .

*Demostración.* Si  $x \in O_1 \cap \partial H^n$ , por  $x \in O_1$ , se tiene  $H_*(H^n, H^n - \{x\}) = 0$ .  $H_*(H^n, H^n - \{x\}) \cong H_*(O_1, O_1 - \{x\}) \cong H_*(O_2, O_2 - \{\phi(x)\}) \cong H_*(H^m, H^m - \{\phi(x)\})$ . Así,  $x \in O_1 \cap \partial H^n \iff \phi(x) \in O_2 \cap \partial H^m$ , de donde se deduce a).

Para demostrar b), solo hemos de aplicar a) para obtener que  $\phi(O_1 \cap \overset{\circ}{H}^n) = O_2 \cap \overset{\circ}{H}^m$ , y como  $\overset{\circ}{H}^n, \overset{\circ}{H}^m$  son abiertos en  $\mathbb{R}^n, \mathbb{R}^m$  respectivamente, el teorema de invarianza de la dimensión nos dice que  $n = m$ .  $\square$

**Definición 3.16**

Una **variedad topológica con borde** es un espacio topológico Hausdorff  $M$  verificando que para todo  $p \in M$  existen  $(V_p, \phi_p), n(p) \geq 1$  con  $V_p$  entorno abierto de  $p$ ,  $\phi_p: V_p \rightarrow O_p \subseteq H^{n(p)}$  homeomorfismo sobre un abierto  $O_p$  de  $H^{n(p)}$ .

**Proposición 3.17 (Consecuencias)**

Se verifican los siguientes resultados sobre variedades topológicas con borde:

- a) En cada componente conexa de  $M$ ,  $n(p)$  es constante, y la llamaremos la dimensión de dicha componente,
- b) Si  $\partial M = \{p \in M \mid H_*(M, M - \{p\}) = 0\}$ , entonces

$$\partial M = \{p \in M \mid \forall (V_p, \phi_p), \phi_p(p) \in \partial H^n \cap O_p\},$$

- c) Si tomamos  $\overset{\circ}{M} = M - \partial M$ , entonces  $\overset{\circ}{M}$  tiene estructura de variedad topológica (sin borde)  $n$ -dimensional, y  $\partial M$  tiene estructura de variedad topológica con borde  $(n-1)$ -dimensional. Dicha estructura es la única que convierte a las inclusiones  $\overset{\circ}{M} \hookrightarrow M$ ,  $\partial M \hookrightarrow M$  en embebimientos. Además,  $\partial M$  es cerrado de  $M$  y, por tanto,  $\overset{\circ}{M}$  es abierto de  $M$ .

*Demostración.* a) es trivial por una proposición ya vista.

- b) Si  $(V_p, \phi_p)$  es una carta en  $p$ , entonces  $H_*(M, M - \{p\}) \cong H_*(V_p, V_p - \{p\}) \cong H_*(O_p, O_p - \{\phi_p(p)\}) \cong H_*(H^n, H^n - \{\phi_p(p)\})$ , que vale 0 sí y solo sí  $\phi_p(p) \in \partial H^n$ .

- c) Si  $\phi: V \rightarrow O \subseteq H^n$  es una carta de  $M$ , entonces  $\phi(V \cap \partial M) = O \cap \partial H^n$ , y se tiene que  $\phi(V \cap \overset{\circ}{M}) = O \cap \overset{\circ}{H^n}$ , abierto en  $O$ , luego  $V \cap \overset{\circ}{M}$  es abierto en  $V$ , y por tanto en  $M$ . Así,  $\overset{\circ}{M}$  es abierto en  $M$ .

Las estructuras se definen por:

$$\begin{aligned} \overset{\circ}{M}: \{ (V \cap \overset{\circ}{M}, \phi|_{V \cap \overset{\circ}{M}}) \mid (V, \phi) \text{ es una carta en } M \}, \\ \partial M: \{ (V \cap \partial M, \phi|_{V \cap \partial M}) \mid (V, \phi) \text{ es una carta en } M \}. \end{aligned}$$

El resto es trivial. □

Vamos a definir el concepto de variedad diferenciable.

### Definición 3.18

Una *variedad diferenciable con borde*  $n$ -dimensional es una variedad topológica con borde  $n$ -dimensional en la que los cambios de coordenadas son diferenciables.

### Nota 3.19

Podemos extender la proposición anterior al caso de variedades diferenciables. Para ello, únicamente hemos de extender c) al caso diferenciable.

## 3.3 SEPARACIÓN DE JORDAN-BROWER. INVARIANZA DEL DOMINIO

### Proposición 3.20

Sea  $A \subseteq S^n$  homeomorfo a  $I^k$ , con  $0 \leq k \leq n$ . Entonces:

$$H_p(S^n - A) \cong \begin{cases} 0 & \text{si } p > 0, \\ \mathbb{Z} & \text{si } p = 0. \end{cases}$$

*Demostración.* Lo demostraremos por inducción sobre  $k$ .

Si  $k = 0$ ,  $I^k = \{p\}$ , luego  $S^n - A \cong \mathbb{R}^n$ , y se tiene el resultado.

Supongamos el resultado cierto hasta  $k-1$ , y probémoslo para  $k$ .

Sea  $\phi: I^k \rightarrow A$  homeomorfismo, y definamos

$$I_+^k = \{x \in I^k \mid x_1 \geq \frac{1}{2}\}, \quad I_-^k = \{x \in I^k \mid x_1 \leq \frac{1}{2}\}.$$

Sean  $A^+, A^-$  las imágenes por  $\phi$  de  $I_+^k, I_-^k$  respectivamente. En  $S^n - (A^+ \cap A^-)$  tomamos la descomposición  $U = S^n - A^+, V = S^n - A^-$ . Así,  $U \cap V = S^n - A$ , y por Mayer-Vietoris, se tiene:

$$H_p(S^n - (A^+ \cap A^-)) \rightarrow H_{p-1}(S^n - A) \xrightarrow{(i_+^*, i_-^*)} H_{p-1}(S^n - A^+) \oplus H_{p-1}(S^n - A^-) \rightarrow \dots$$

Como  $A^+ \cap A^- \cong I^{k-1}$ , por hipótesis de inducción

$$H_p(S^n - (A^+ \cap A^-)) \cong \begin{cases} 0 & \text{si } p > 0, \\ \mathbb{Z} & \text{si } p = 0. \end{cases}$$

Así,  $(i_+^*, i_-^*)$  es un isomorfismo si  $p > 0$ , y un monomorfismo si  $p = 0$ .

Supongamos ahora que el resultado no es cierto. Entonces, para  $p > 0$  existe  $[\alpha] \in H_p(S^n - A)$  con  $[\alpha] \neq 0$ , y si  $p = 0$ , existen  $x, y \in S^n - A$  tales que  $[x - y] \neq 0$ .

Para tratar ambos casos a la vez llamamos  $\beta$  a  $(x - y)$  o  $\alpha$ , indistintamente.

Por lo que sabemos de  $(i_+^*, i_-^*)$ , existe  $A_1 \subseteq A, A_1 \cong I^k$  tal que  $i_{1*}(\beta) \neq 0$ , con  $i_1: S^n - A \hookrightarrow S^n - A_1$ .

Nos basta con tomar  $A_1 = A^+$  o  $A_1 = A^-$ , dependiendo de si  $i_+^*(\beta) = 0$  o  $i_-^*(\beta) = 0$ .

Aplicamos el mismo razonamiento a  $A_1$ , con lo que obtenemos una sucesión

$$A \supset A_1 \supset \dots \supset A_k \supset \dots \text{ con } A_i \cong I^k \forall i$$

tal que  $\bigcap_{i=1}^{\infty} A_i \cong I^{k-1}$ , y tal que si  $i_m: S^n - A \hookrightarrow S^n - A_m$  es la inclusión, entonces  $i_{m*}(\beta) \neq 0$ .

Consideramos el diagrama de inclusiones:

$$\begin{array}{ccc} S^n - A & \xrightarrow{i_m} & S^n - A_m \\ & \searrow i & \swarrow j_m \\ & S^n - \bigcap_{i=1}^{\infty} A_i & \end{array}$$

Como  $H_p(S^n - \bigcap_{i=1}^{\infty} A_i) = \begin{cases} 0 & \text{si } p > 0, \\ \mathbb{Z} & \text{si } p = 0 \end{cases}$  por inducción, entonces  $0 = i_*(\beta) = [i_{\#}(\beta)]$ ,

luego existe  $d \in S_{p+1}(S^n - \bigcap_{i=1}^{\infty} A_i)$  con  $\partial d = i_{\#}(\beta)$ .

Si  $d = \sum_{i=1}^r m_i \phi_i$ , entonces  $\bigcup_{i=1}^r \text{Img } \phi_i$  es un compacto en  $S^n - \bigcap_{i=1}^{\infty} A_i = \bigcup_{i=1}^{\infty} (S^n - A_i)$ , luego por el teorema de Heine-Borel:

$$\bigcup_{i=1}^r \text{Img } \phi_i \subseteq (S^n - A_{i_1}) \cup \dots \cup (S^n - A_{i_k}) = S^n - A_m,$$

con  $m = \max\{i_1, \dots, i_k\}$ .

Así, existe  $e \in S_{p+1}(S^n - A_m)$  con  $j_{m\#}(e) = d$ . Por tanto,  $j_{m\#} \circ i_{m\#}(\beta) = i_{\#}(\beta) = \partial d = \partial j_{m\#}(e) = j_{m\#}(\partial e)$ .

Como  $j_{m\#}$  es un monomorfismo,  $i_{m\#}(\beta) = \partial e$ , entonces  $i_{m\#}(\beta) = 0$ , lo que contradice lo afirmado.  $\square$

### Teorema 3.21

Sea  $A \subseteq S^n$ ,  $n \geq 2$ ,  $A$  homeomorfo a  $S^k$  con  $0 \leq k < n$ . Entonces:

$$\text{Si } k \leq n-2, \quad H_p(S^n - A) = \begin{cases} 0 & \text{si } p \neq 0, n-k-1, \\ \mathbb{Z} & \text{si } p = 0, p = n-k-1. \end{cases}$$

$$\text{Si } k = n-1, \quad H_p(S^n - A) = \begin{cases} 0 & \text{si } p > 0, \\ \mathbb{Z} \oplus \mathbb{Z} & \text{si } p = 0. \end{cases}$$

*Demostración.* Lo probamos por inducción sobre  $k$ .

Si  $k = 0$ ,  $S^0$  son dos puntos, luego  $A$  es homeomorfo a dos puntos, y  $S^n - A \cong \mathbb{R}^n - \{x\} \cong S^{n-1}$ . Se comprueba que el resultado coincide con el que conocemos para el grupo de homología de las esferas.

Supuesto cierto hasta  $k-1$ , lo probamos para  $k$ .

Sea  $S^k = E_+^k \cap E_-^k$ , con  $E_+^k = \{x \in S^k \mid x_{k+1} \geq 0\}$ ,  $E_-^k = \{x \in S^k \mid x_{k+1} \leq 0\}$ . Entonces,  $E_+^k$  y  $E_-^k$  son homeomorfos a  $I^k$ , y  $E_+^k \cap E_-^k \cong S^{k-1}$ .

Del resultado anterior y de la hipótesis de inducción, conocemos  $H_*(S^k, E_+^k)$ ,  $H_*(S^k, E_-^k)$  y  $H_*(S^n - (E_+^k \cap E_-^k))$ . Tomando la sucesión de Mayer-Vietoris de  $S^n - (E_+^k \cap E_-^k)$ , con  $U = S^n - E_+^k$ ,  $V = S^n - E_-^k$ , se tiene:

$$\begin{aligned} H_p(S^n - E_+^k) \oplus H_p(S^n - E_-^k) &\rightarrow H_p(S^n - (E_+^k \cap E_-^k)) \rightarrow H_{p-1}(S^n - S^k) \\ &\rightarrow H_{p-1}(S^n - E_+^k) \oplus H_{p-1}(S^n - E_-^k). \end{aligned}$$

Si  $p \geq 2$ , se verifica  $H_{p-1}(S^n - S^k) \cong H_p(S^n - (E_+^k \cap E_-^k)) \cong H_p(S^n - S^{k-1})$ , que por

$$\text{inducción es } \begin{cases} 0 & \text{si } p \neq 0, n-k, \\ \mathbb{Z} & \text{si } p = 0, n-k. \end{cases}$$

Llamando  $q = p-1$ , si  $k = n-1$ , como  $p \geq 2$ ,  $H_q(S^n - A) = 0 \quad \forall q \geq 1$ .

$$\text{Si } k \leq n-2, n-k \geq 2, \text{ y } H_q(S^n - A) = \begin{cases} \mathbb{Z} & q = n-k-1, \\ 0 & \text{en otro caso.} \end{cases}$$

Para  $p = 1$  ( $q = 0$ ), tomamos la sucesión exacta

$$\begin{aligned} 0 \rightarrow H_1(S^n - (E_+^k \cap E_-^k)) &\rightarrow H_0(S^n - S^k) \rightarrow H_0(S^n - E_+^k) \oplus H_0(S^n - E_-^k) \rightarrow \\ &\rightarrow H_0(S^n - (E_+^k \cap E_-^k)) \rightarrow 0. \end{aligned}$$

Como  $H_0(S^n - E_+^k)$ ,  $H_0(S^n - E_-^k)$  y  $H_0(S^n - (E_+^k \cap E_-^k))$  son todos isomorfos a  $\mathbb{Z}$ , se deduce que  $H_0(S^n - S^k) \cong \mathbb{Z} \oplus H_1(S^n - (E_+^k \cap E_-^k)) = \begin{cases} \mathbb{Z} \oplus \mathbb{Z} & k = n-1, \\ \mathbb{Z} & k < n-1. \end{cases} \quad \square$

**Teorema 3.22** (Separación de Jordan-Brower)

Sea  $f: S^{n-1} \rightarrow S^n$  un embebimiento. Entonces  $S^n - f(S^{n-1})$  tiene dos componentes conexas, cuya frontera común es  $f(S^{n-1})$ .

*Demostración.* Sea  $A = f(S^{n-1})$ .  $H_0(S^n - A) \cong \mathbb{Z} \oplus \mathbb{Z}$ , por lo que  $S^n - A$  tiene dos componentes arcoconexas, a las que llamamos  $\Omega_1$  y  $\Omega_2$ .

Como  $S^n - A$  es un abierto de  $S^n$ , es localmente arcoconexo, luego sus componentes arcoconexas y conexas coinciden, y por tanto  $\Omega_1$  y  $\Omega_2$  son las componentes conexas de  $S^n - A$ .

Como  $S^n - A$  es localmente arcoconexo, sus componentes conexas son abiertos en  $S^n - A$ , y por tanto en  $S^n$ . De esta forma,  $\Omega_i \cup A = S^n - \Omega_j$ ,  $i \neq j$  es un cerrado en  $S^n$ , de donde se obtiene  $\overline{\Omega_i} \subseteq \Omega_i \cup A$ , y por tanto

$$\text{Fr}\Omega_i = \overline{\Omega_i} \cap \overline{S^n - \Omega_i} \subseteq (\Omega_i \cup A) \cap (S^n - \Omega_i) = (\Omega_i \cup A) \cap (\Omega_j \cup A) = A.$$

Así,  $\text{Fr}\Omega_i \subseteq A$ .

Veamos que  $A \subseteq \text{Fr}\Omega_i$ , lo que completa la demostración.

Sean  $a \in A$ ,  $V$  entorno abierto de  $a$  en  $S^n$ . Usando que  $S^n$  es una variedad  $n$ -dimensional, se puede descomponer  $A = A^1 \cup A^2$ , con  $a \in A^2 \subseteq V \cap A$ , y se verifica  $A^1, A^2 \cong I^{n-1}$ .

Sean  $p_i \in \Omega_i$ ,  $p_i \in S^n - A \subseteq S^n - A^1$ . Como  $H_0(S^n - A^1) \cong \mathbb{Z}$ ,  $S^n - A^1$  es arcoconexo, luego existe un arco  $\gamma: [0, 1] \rightarrow S^n - A^1$  que va desde  $p_1$  a  $p_2$ .

Por tanto, existe un  $t \in [0, 1]$  con  $\gamma(t) \in A^2$ , pues si no, la curva estaría en  $S^n - (A^1 \cup A^2) = S^n - A$  y uniría dos componentes conexas distintas.

Como  $\gamma^{-1}(A^2) \neq \emptyset$  y es cerrado, es un compacto de  $[0, 1]$ , del que podemos tomar mínimo y máximo, a los que llamamos  $t_m$  y  $t_M$  respectivamente. Tomando  $\epsilon$  suficientemente pequeño,  $\gamma(t_m - \epsilon) \in \Omega_1 \cap V$ , y  $\gamma(t_M + \epsilon) \in \Omega_2 \cap V$ , luego  $a \in \overline{\Omega_1} \cap \overline{\Omega_2}$ ,  $a \in \text{Fr}\Omega_i$ .  $\square$

Este teorema nos permite obtener directamente un resultado que puede verse como una generalización del teorema de la curva de Jordan a dimensiones superiores.

**Corolario 3.23**

Sea  $f: S^{n-1} \rightarrow \mathbb{R}^n$  un embebimiento. Entonces,  $\mathbb{R}^n - f(S^{n-1})$  tiene dos componentes conexas, y  $f(S^{n-1})$  es la frontera común de ambas.



**Teorema 3.24** (Invarianza del dominio)

Sean  $A_1, A_2 \subseteq S^n$  subconjuntos de  $S^n$  homeomorfos por  $h: A_1 \rightarrow A_2$ . Entonces,  $A_1$  es abierto de  $S^n \iff A_2$  es abierto de  $S^n$ .

**Nota 3.25**

Nótese que este hecho no es trivial. Lo sería en caso de que  $A_1, A_2$  fueran cerrados, o si  $h$  estuviera definida en todo  $S^n$  y fuera un homeomorfismo entre ambos conjuntos.

*Demostración.* Supongamos que  $A_1$  es abierto, y veamos que  $A_2$  también lo es. Sea  $a_2 \in A_2$ , y sea  $a_1 \in A_1$  tal que  $h(a_1) = a_2$ , que es único, puesto que  $A_1$  y  $A_2$  son homeomorfos. Existe un entorno de  $a_1$ , al que notamos  $V_1 \subseteq S^n$ , tal que  $V_1 \subseteq A_1$  y  $V_1 \cong D^n$ . Sean  $V_2 = h(V_1)$ ,  $W_2 = h(\partial V_1)$ , con  $\partial V_1 \cong S^{n-1}$ . Se verifica que  $a_2 \in V_2 - W_2 \subseteq A_2$ .

Por el teorema de separación de Jordan-Brower,  $S^n - W_2$  tiene dos componentes conexas, y al ser  $V_2 \cong D^n$ ,  $S^n - V_2$  es conexo. Además,  $S^n - W_2 = (V_2 - W_2) \cup (S^n - V_2)$ , siendo esta unión disjunta, luego  $V_2 - W_2$  y  $S^n - V_2$  son las componentes conexas de  $S^n - W_2$ , y por tanto abiertos en  $S^n - W_2$  (y en  $S^n$ ).

Como  $a_2 \in V_2 - W_2 \subseteq A_2$ ,  $A_2$  es abierto.

Para demostrar la otra implicación basta con intercambiar los papeles de  $A_1$  y  $A_2$  y considerar el homeomorfismo inverso.  $\square$

## 3.4 GRADO DE UNA APLICACIÓN CONTINUA

Vamos a definir el grado de una aplicación continua de una esfera en sí misma, aprovechando lo que sabemos de los grupos de homología de las esferas.

Sea  $f: S^n \rightarrow S^n$  una aplicación continua. Los únicos homomorfismos que induce  $f$  y que son no triviales son  $f_*: H_0(S^n) \rightarrow H_0(S^n)$  y  $f_*: H_n(S^n) \rightarrow H_n(S^n)$ .

Se cumple que el primero de ellos es siempre la identidad. En el segundo caso, como  $H_n(S^n) \cong \mathbb{Z}$ , si  $[\alpha]$  es un generador de  $H_n(S^n)$ , necesariamente se tiene que  $f_*([\alpha]) = n \times [\alpha]$ ,  $n \in \mathbb{Z}$ .

Definimos como el **grado** de  $f$  a dicho entero  $n$ , y lo notamos  $\text{Deg } f$ . Por tanto, se cumple que  $f_*([\alpha]) = \text{Deg } f \times [\alpha]$ .

Veamos algunas de sus propiedades.

**Proposición 3.26** (Propiedades)

Se verifican las siguientes propiedades:

- a)  $\text{Deg Id} = 1$ ,
- b)  $\text{Deg}(g \circ f) = \text{Deg } g \times \text{Deg } f$ ,
- c)  $f \simeq g \implies \text{Deg } f = \text{Deg } g$ ,

d) Si  $f$  es una equivalencia homotópica,  $\text{Deg } f = \pm 1$  (en particular, es cierto si  $f$  es homeomorfismo),

e) Si  $\text{Deg } f \neq 0 \implies f$  es sobreyectiva.

La implicación contraria en la propiedad c) también es cierta, el llamado Teorema de Hopf, que no demostraremos.

*Demostración.* Las demostraciones son sencillas usando la definición. Probamos únicamente e).

Supongamos que  $f$  no es sobreyectiva, es decir, existe  $x \in S^n$  con  $x \notin \text{Img } f$ . Así, tenemos el diagrama

$$\begin{array}{ccc} S^n & \xrightarrow{f} & S^n \\ & \searrow f' & \nearrow i \\ & S^n - \{x\} & \end{array}$$

que induce el siguiente diagrama en la homología:

$$\begin{array}{ccc} H_n(S^n) & \xrightarrow{f_*} & H_n(S^n) \\ & \searrow f'_* & \nearrow i_* \\ & H_n(S^n - \{x\}) \cong 0 & \end{array}$$

dándonos una descomposición de  $f_*$  que tiene a la función nula como factor, lo cual es imposible.  $\square$

### Lema 3.27

Si  $\phi: S^n \rightarrow S^n$  es una isometría, entonces  $\text{Deg } \phi = \text{Det } \phi$ , viendo  $\phi$  como una matriz cuadrada de orden  $n+1$ .

*Demostración.* Por el teorema de Cartan-Dieudonné, sabemos que toda isometría en  $\mathbb{R}^m$  puede expresarse como composición de  $k$  simetrías, con  $k \leq m$ . Por tanto, podemos tomar  $\phi = \sigma_i \circ \dots \circ \sigma_k$ ,  $k \leq n+1$ .

Como  $\text{Deg } \phi = \prod_{i=1}^k \text{Deg } \sigma_i$ , y  $\text{Det } \phi = \prod_{i=1}^k \text{Det } \sigma_i$ , basta con probar el resultado para simetrías.

Sea  $\sigma: S^n \rightarrow S^n$  una simetría respecto al hiperplano  $H$ , y sea  $A: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}$  la isometría que lleva  $H$  en el hiperplano  $\{x \in \mathbb{R}^{n+1} \mid x_1 = 0\}$ . Entonces, si tomamos la aplicación que cambia el signo a la primera coordenada,  $\tau(x_1, \dots, x_{n+1}) = (-x_1, \dots, x_{n+1})$ , se verifica que  $\sigma = A^{-1} \circ \tau \circ A$ , luego  $\text{Deg } \sigma = \text{Deg } A^{-1} \times \text{Deg } \tau \times \text{Deg } A = \text{Deg } \tau$ .

Por tanto, basta probar que  $\text{Deg } \tau = \text{Det } \tau = -1$ . Lo hacemos por inducción.

Si  $n = 1$ , tomamos  $U = S^1 - (0, 1)$ ,  $V = S^1 - (0, -1)$ . Entonces, el homomorfismo de conexión de la sucesión de Mayer-Vietoris asociada (que notábamos  $\Delta$ ) está dado por

$\Delta([\alpha]) = [x - y]$ , siendo  $[\alpha]$  un generador de  $H_1(S^1)$ . Por la naturalidad de dicha sucesión respecto a  $\tau$ , se tiene la conmutatividad del siguiente diagrama:

$$\begin{array}{ccc} H_1(S^1) & \xrightarrow{\Delta} & H_0(U \cap V) \\ \downarrow \tau_* & & \downarrow \tau_* \\ H_1(S^1) & \xrightarrow{\Delta} & H_0(U \cap V) \end{array}$$

Así, tenemos las dos siguientes cadenas de igualdades:

$$\tau_*(\Delta([\alpha])) = \Delta(\tau_*([\alpha])) = \Delta(\text{Deg } \tau \times [\alpha]) = \text{Deg } \tau \times [x - y],$$

$$\tau_*(\Delta([\alpha])) = \tau_*([x - y]) = [\tau_*(x) - \tau_*(y)] = [y - x],$$

de donde se deduce  $\text{Deg } \tau = -1$ .

Supongamos ahora que el resultado es cierto hasta  $n - 1$ , y lo probamos para  $n$ . Denotaremos como  $\tau_{n-1}$  y  $\tau_n$  las correspondientes aplicaciones en cada espacio, para evitar confusión.

Considerando en  $S^n$  la misma descomposición anterior, se tiene:

$$\begin{array}{ccccc} H_n(S^n) & \xrightarrow{\Delta} & H_{n-1}(U \cap V) & \xleftarrow{i_*} & H_{n-1}(S^{n-1}) \\ \downarrow \tau_{n*} & & \downarrow \tau_{n*} & & \downarrow \tau_{n-1*} \\ H_n(S^n) & \xrightarrow{\Delta} & H_{n-1}(U \cap V) & \xleftarrow{i_*} & H_{n-1}(S^{n-1}) \end{array}$$

$i_*$  es un isomorfismo, pues  $S^{n-1}$  es un retracto de  $U \cap V$ .

Así,  $\tau_{n*}([\alpha]) = \text{Deg } \tau_n \times [\alpha]$  y además:

$$\tau_{n*}([\alpha]) = \Delta^{-1} \tau_{n*} \Delta([\alpha]) = (\Delta^{-1} i_* \tau_{n-1*} i_*^{-1} \Delta)([\alpha]) = (\Delta^{-1} i_* i_*^{-1} \Delta)(-[\alpha]) = -[\alpha].$$

Por tanto,  $\text{Deg } \tau_n = -1$ . □

Como consecuencia directa de este hecho, si consideramos la aplicación  $-\text{Id}: S^n \rightarrow S^n$ , que lleva cada punto en su antípoda, se verifica  $\text{Deg}(-\text{Id}) = \text{Det}(-\text{Id}) = (-1)^{n+1}$ .

Veamos un lema que nos permitirá demostrar dos resultados de aplicaciones sobre esferas de dimensión par, con los que terminaremos el capítulo.

### Lema 3.28

Sean  $f, g: S^n \rightarrow S^n$  aplicaciones tales que  $f(x) \neq g(x) \forall x \in S^n$ . Entonces  $f \simeq -g$ .

*Demostración.* Basta ver que  $H(x, t) = \frac{tf(x) - (1-t)g(x)}{|tf(x) - (1-t)g(x)|}$  es la homotopía buscada. □

### Corolario 3.29

Si  $f: S^{2n} \rightarrow S^{2n}$  es continua, entonces existe  $x \in S^{2n}$  tal que  $f(x) = x$  o  $f(x) = -x$ .

*Demostración.* Supongamos que existe  $f$  tal que para todo  $x \in S^{2n}$ ,  $f(x) \neq x$  y  $f(x) \neq -x$ . Entonces, por el lema anterior,  $f \simeq -\text{Id}$  y  $f \simeq \text{Id}$ , luego  $\text{Id} \simeq -\text{Id}$  y  $\text{Deg}(\text{Id}) = \text{Deg}(-\text{Id})$ , lo que implica  $1 = -1$ , y llegamos a una contradicción.  $\square$

### Corolario 3.30

*No existen campos continuos sin ceros en una esfera de dimensión par.*

*Demostración.* Recordemos que el tangente a  $x \in S^{2n}$  es  $\langle x \rangle^\perp$ . Por tanto, un campo continuo en  $S^{2n}$  es una aplicación continua  $f: S^{2n} \rightarrow \mathbb{R}^{2n+1}$  tal que  $\langle f(x), x \rangle = 0 \forall x \in S^{2n}$ .

Si dicho campo no tiene ceros, sea  $g: S^{2n} \rightarrow S^{2n}$  dada por  $g(x) = \frac{f(x)}{|f(x)|}$ . Como  $g$  es continua, existe  $x$  con  $g(x) = x$  o  $g(x) = -x$ . Sin embargo,  $\langle g(x), x \rangle = 0$  para todo  $x$ , lo cual nos lleva a una contradicción, pues los elementos de  $S^{2n}$  no pueden tener norma nula.  $\square$

## Parte II

### ANÁLISIS Y RESOLUCIÓN DEL PROBLEMA DEL CLIQUE MÁXIMO

En esta parte, analizaremos y resolveremos mediante distintas técnicas el problema del clique máximo, comparando sus resultados. Introduciremos el trabajo realizado y los recursos utilizados, pasando a una descripción de los algoritmos. Continuaremos con detalles específicos de la implementación, y expondremos los resultados obtenidos, realizando un análisis individual y conjunto de las técnicas incluidas.

## INTRODUCCIÓN

---

### 4.1 INTRODUCCIÓN AL PROBLEMA

Dado un grafo  $G = (V, E)$ , donde  $V$  representa el conjunto de vértices y  $E$  el conjunto de aristas, llamaremos clique a un subgrafo de  $G$  en el que todos sus vértices están conectados entre sí. Llamaremos a los cliques **maximales** si no son subgrafo de un clique de mayor tamaño, y **máximos** si no existe ningún clique de mayor tamaño en el grafo. Es claro que todos los cliques máximos son maximales, pero no al contrario.

El problema del clique máximo, o *maximum clique problem* en inglés (MCP), consiste en encontrar el clique de mayor número de vértices dentro de un grafo  $G$  cualquiera. Este problema es equivalente a otros dos también muy conocidos, que son el problema del conjunto independiente, en el que se intenta buscar el mayor conjunto de nodos no conectados entre sí, y el problema del recubrimiento por vértices, en el que buscamos el mínimo conjunto de nodos que cubran todas la aristas.

El problema del clique máximo lleva siendo estudiado desde segunda mitad del siglo XX, y fue sujeto de estudio en el segundo desafío de implementación de DIMACS, que tuvo lugar entre 1992 y 1993, junto con el problema del coloreo de grafos y de satisfabilidad booleana (SAT). A día de hoy, se encuentran disponibles de forma libre las instancias utilizadas y los datos de los mejores cliques conocidos hasta la fecha.

En 1979, Garey y Johnson [4] demostraron que el problema es NP-difícil. En 1999, Hastad [10] probó que, salvo que  $NP = ZPP$ , no existe ningún algoritmo capaz de aproximar MCP con factor  $n^{1-\epsilon} \quad \forall \epsilon$ , donde la clase ZPP (zero-error probabilistic polynomial time) es la clase de complejidad de los problemas reconocibles por algoritmos aleatorizados con complejidad polinómica. Debido a esto, no es posible obtener algoritmos exactos ni que den una solución aproximada al problema, de ahí la necesidad del uso de algoritmos heurísticos para resolverlo.

### 4.2 INTRODUCCIÓN A LAS METAHEURÍSTICAS

En Inteligencia Artificial, se emplea el término **heurística** para referirnos a una técnica, método o procedimiento de realizar una tarea, que no es producto de un análisis formal riguroso, sino del conocimiento experto sobre la tarea [15].

El uso de heurísticas es una parte fundamental en la resolución de problemas de optimización, pues usualmente estos serán lo bastante complejos como para no poder resolverlos de forma exacta en un tiempo razonable. Con ellas, seremos capaces de crear

procedimientos que obtengan soluciones cercanas al óptimo en una fracción mínima del tiempo que le tomaría a un algoritmo exacto [20].

Las heurísticas aplicables a la hora de resolver un problema pueden ser más generales o más específicas que otras. Los métodos más específicos deben ser diseñados de forma distinta para cada problema, utilizando toda la información y el análisis teórico disponible sobre él. Bien diseñados, suelen tener un rendimiento más alto que las heurísticas generales. Por el contrario, las heurísticas más generales son más sencillas, adaptables y robustas, pues no dependen de forma tan fuerte del problema que se resuelve.

Las metaheurísticas pueden ser vistas como estrategias de diseño generales para procedimientos heurísticos de alto rendimiento. Estas estrategias quedan por encima de las heurísticas convencionales, representando un nivel superior de abstracción. El término metaheurística apareció por primera vez en un artículo de Fred Glover [6], y desde entonces han surgido numerosas propuestas, mediante la aparición de pautas para diseñar procedimientos para resolver distintos problemas, los cuales, al ser extendidos a otros ámbitos, han adquirido la denominación de metaheurísticas.

Podemos clasificar las metaheurísticas en los siguientes tipos [15]:

- Las metaheurísticas de relajación utilizan un modelo relajado del problema original, que lo hace más fácil de resolver, y cuya solución facilita la solución del problema original.
- Las metaheurísticas de búsqueda siguen movimientos o transformaciones guiadas para explorar el espacio de soluciones, aprovechando las estructuras de entornos asociadas.
- Las metaheurísticas constructivas tratan de obtener una solución a través de un proceso de análisis y selección de sus componentes.
- Las metaheurísticas evolutivas trabajan con un conjunto de soluciones que va cambiando a lo largo del tiempo mediante el intercambio de información.

Las metaheurísticas pueden ser de uno o varios tipos, lo que daría lugar a metaheurísticas híbridas. También es posible encontrar metaheurísticas que no se engloben en estos cuatro tipos, como las basadas en redes neuronales. Esta clasificación queda abierta a la interpretación, pudiendo ser modificada, y no debe tomarse nunca como definitiva o infalible.

### 4.3 EL REPOSITORIO DIMACS

Las siglas DIMACS hacen referencia al centro de matemática discreta y computación teórica (del inglés, Center for Discrete Mathematics and Theoretical Computer Science). Es una colaboración entre dos universidades estadounidenses, las de Rutgers y Princeton, además de las secciones de investigación de las empresas AT&T, Bell Labs, Applied Communication Science y NEC. Fue fundada en el año 1989, y desde 1990 convoca desafíos de implementación para determinados problemas de interés en el ámbito científico, con el objetivo de conseguir algoritmos de calidad.

El problema del clique máximo fue, junto con el coloreo de grafos y el problema de satisfacibilidad de cláusulas (SAT), protagonista del segundo desafío DIMACS, que tuvo lugar entre 1990 y 1991. Para el problema, se proporcionó un conjunto de instancias de grafos, que son de acceso público y las que utiliza la comunidad científica para evaluar nuevos algoritmos. Debido a esto, he decidido utilizar estas instancias para evaluar los algoritmos considerados en este trabajo. Dichas instancias pueden encontrarse en [http://iridia.ulb.ac.be/~fmascia/maximum\\_clique/DIMACS-benchmark](http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark).

En dicho repositorio podemos encontrar un total de 37 grafos, de los cuales he tomado 35. Ha quedado fuera los grafos *MANN\_a81* y *MANN\_a45*, debido a que por su tamaño los tiempos de ejecución eran muy elevados. Para ejecutar sobre ellos los algoritmos con los mismos parámetros que sobre el resto de grafos, se hubiera requerido un tiempo estimado de más de una semana, lo que suponía una cifra demasiado alta en comparación con el resto.

Los grafos están subdivididos en distintos grupos, que hacen referencia al método de generación de los mismos. Debido a esto, es posible que algunos algoritmos muestren un buen comportamiento en ciertos tipos de grafos, y den peores resultados en grafos que correspondan a otra familia. Los métodos de generación de cada familia de grafos se pueden consultar con detalle en la web del repositorio.

En ciertas instancias se ha confirmado el valor del óptimo global, mientras que en otras solo se conoce una cota inferior a la mejor solución posible. Estos valores serán tomados como referencia a la hora de evaluar cada algoritmo implementado y determinar la calidad de las soluciones.

#### 4.4 OBJETIVOS

Los objetivos de este trabajo son los siguientes:

- Recopilación de bibliografía sobre el problema del clique máximo y sobre las posibles metaheurísticas a aplicar al problema. Con esto, queremos aprovechar el conocimiento existente sobre el problema para tener una base sobre la que trabajar, junto a un conjunto de técnicas que se pueden utilizar para resolver el problema. Así, no solo se conocerá mejor el problema, sino que será posible abordarlo de forma más eficiente.
- Estudio y selección de las técnicas a utilizar. Buscaremos, de entre todas las técnicas aplicables, aquellas que sean de interés a la hora de realizar un posterior análisis.
- Diseño e implementación de los algoritmos seleccionados. Tomaremos las características de los algoritmos que nos interesen, y las modificaremos para adaptarlas a la representación de problema, aprovechando la funcionalidad del lenguaje de programación escogido.
- Experimentación y estudio comparativo de los algoritmos implementados. Una vez obtengamos los datos experimentales, compararemos los algoritmos para com-



probar empíricamente su funcionamiento, basándonos en la calidad de las soluciones obtenidas por cada uno de ellos. Veremos cómo actúan sobre el conjunto de instancias del problema y discutiremos su viabilidad a la hora de resolverlo.

El primer objetivo se ha cumplido en cierta medida, pues, si bien se ha recopilado bibliografía referente al problema del clique máximo, esta se centraba en su gran mayoría en los métodos de resolución del problema, y no tanto en el estudio teórico del mismo.

El segundo objetivo se ha cumplido en un grado alto, como se demuestra en la recopilación de algoritmos implementados que veremos en el [Capítulo 5](#).

El tercer objetivo se ha completado satisfactoriamente, como veremos en el [Capítulo 6](#).

El cuarto objetivo también se ha completado de forma satisfactoria, y queda reflejado en el [Capítulo 7](#).

#### 4.5 METODOLOGÍA Y TRABAJO REALIZADO

Este trabajo se ha desarrollado siguiendo una metodología ágil, creando un proyecto desde cero y tratando de agregar distintas componentes de forma que la funcionalidad fuese aumentando paulatinamente. Así, cada vez que se añadía un nuevo componente, se buscaban los puntos comunes con el trabajo ya existente, y en caso de conflicto, se solucionaba adaptando las partes más antiguas a los cambios, de forma que el funcionamiento de ellas fuese igual que antes de realizar dichos cambios.

De esta forma, fallos de diseño, o partes que se hubieran obviado en un principio, son arreglados en cuanto se detectan. Así, el proyecto es funcional desde el comienzo hasta que se termina, lo que ayuda a encontrar fallos de forma temprana y a comprobar que las soluciones a estos fallos cumplen con su cometido, pues siempre se puede testear el trabajo que esté finalizado.

El desarrollo del trabajo ha sido el siguiente:

El primer paso fue buscar información existente sobre el problema, lo que me llevó al repositorio DIMACS y a numerosos artículos publicados. Con esto, pude saber las líneas de trabajo que se siguen para su resolución y adquirir una base sobre la que empezar a trabajar, además de establecer un modelo del problema.

Analizando la literatura disponible, pude comprobar la variedad de algoritmos que ya existían. De entre todos los estudiados, elegí los que me parecieron de mayor interés para profundizar sobre ellos, tomándolos total o parcialmente.

Una vez seleccionados los algoritmos, tuve que ver cómo se adaptaban al modelo del problema considerado. Introduje los cambios necesarios para que estos funcionasen de la forma más similar posible a los originales.

Después, pasé a la creación del programa encargado de la ejecución de los algoritmos. Implementé, siguiendo el modelo considerado, los algoritmos elegidos y todo lo relacio-

nado con la inicialización del programa, la obtención de soluciones y la toma de tiempos. Una vez implementado el programa, hice pruebas sobre el conjunto de datos para comprobar que funcionase correctamente y corregir todos los errores que surgieran. Finalmente, pasé a la toma de datos ejecutando todos los algoritmos sobre las instancias consideradas, recopilando los datos necesarios para su posterior análisis.

#### 4.6 ORGANIZACIÓN

El resto de la memoria está organizada de la siguiente forma:

Primero, introduciremos teóricamente los algoritmos considerados en este trabajo. Para ello, comenzaremos con un pequeño apartado donde se tratan elementos generales a todos ellos, seguido de un desglose de los algoritmos divididos según la metaheurística a la que pertenezcan, consistente en una introducción a la metaheurística correspondiente, seguida de cómo esta se adapta al problema del clique máximo, y concluyendo con los detalles de cada algoritmo.

Posteriormente, trataremos la parte de implementación, en la que estudiaremos todo lo referente a la parte de programación de este trabajo: la organización de archivos, la lectura de datos y la implementación de las técnicas descritas con anterioridad, teniendo en cuenta el lenguaje de programación utilizado.

Una vez introducidas y explicadas las técnicas que se han utilizado y su implementación, pasaremos a la sección de resultados, donde se expondrán los cálculos que cada algoritmo ha realizado sobre el conjunto de instancias. Seguidamente, se realizará un análisis de dichas técnicas, en los que evaluaremos si han tenido el comportamiento esperado y las causas de los resultados obtenidos. Compararemos entre sí todas las técnicas implementadas, tratando de clasificarlas según su funcionamiento. Finalizaremos la memoria con un apartado de conclusiones, que incluirá cómo este trabajo podría ser ampliado en el futuro.

## ALGORITMOS

---

En este capítulo se tratarán los diferentes algoritmos considerados en el trabajo, hablando de la metaheurística que los engloba, viendo cómo esta se adapta al problema, y finalmente explicando las técnicas consideradas.

### 5.1 CONSIDERACIONES INICIALES

Antes de describir cada algoritmo, vamos a ver consideraciones más generales sobre el problema, que se utilizarán en los algoritmos implementados. Discutiremos la representación de las soluciones, el operador de vecindad, la función objetivo y la inicialización del programa.

Cada solución del problema será un clique que se encuentre dentro del grafo, y todas ellas forman el **espacio de soluciones**. Estos cliques estarán representados por los nodos que contienen, de forma que las soluciones serán conjuntos  $\{x_0, \dots, x_p\}$ , donde cada elemento  $x_i$  representa un nodo del grafo, que está identificado de forma única por un índice.

La función para medir la calidad de una solución, o **función objetivo**, será el tamaño de cada clique, si bien se considerarán algunas variantes en ciertos algoritmos. En estas variantes preservaremos el tamaño de los cliques como factor principal, pues queremos maximizarlo, y agregaremos otras componentes.

Como **operador de vecino** tenemos tres distintos, los cuales aplicados sobre cada clique nos permitirán obtener su entorno. Los vemos con más detalle:

#### 5.1.1 Operador *add*

El primer operador, al que llamaremos operador *add*, consiste en añadir al clique un nuevo nodo que conserve la estructura de clique, esto es, que esté conectado con todos los nodos del clique. Usualmente tendremos calculado el conjunto de nodos que podemos añadir a cada clique con el que estemos trabajando. Llamaremos a dicho conjunto  $C_0$  o posibles adiciones.

Podemos ver el operador *add* como una función encargada de añadir un nodo a un clique. Supondemos que el nodo  $x$  que añadimos está en el conjunto  $C_0$ , por lo que el grafo resultante sigue siendo un clique.

---

**Algorithm 1** Operador *add*

---

```

function ADD(clique, x)
  clique = clique  $\cup$  x
  return clique

```

---

5.1.2 Operador *swap*

El segundo operador, al que nos referiremos como *swap*, consiste en intercambiar un nodo del clique con uno de fuera, de forma que se siga manteniendo la estructura. Para ello, el nodo que no pertenece al clique debe estar conectado con todos salvo uno de sus nodos, que será el nodo que saldrá del clique. Nos referiremos a este conjunto de nodos como posibles intercambios o  $C_1$ . Al igual que el anterior, también tendremos calculado dicho conjunto de nodos para cada clique con el que trabajemos.

Al igual que el anterior, podemos ver el operador como una función que actúa sobre un grafo y dos nodos, in, dentro del clique, y out, fuera de él. Nuevamente, suponemos que se mantiene la estructura de clique tras realizar las operaciones.

---

**Algorithm 2** Operador *swap*

---

```

function SWAP(clique, in, out)
  clique = clique - in
  clique = clique  $\cup$  out
  return clique

```

---

5.1.3 Operador *drop*

Finalmente, el tercer operador de vecino será el operador *drop*, que consiste simplemente en quitar un nodo del clique. En este caso no tenemos que preocuparnos por que el grafo resultante sea un clique, pues siempre que eliminemos nodos de un clique el resultado seguirá siendo otro clique.

Viéndolo como una función, el funcionamiento sería el siguiente:

---

**Algorithm 3** Operador *drop*

---

```

function DROP(clique, x)
  clique = clique - x
  return clique

```

---

#### 5.1.4 Entorno

Estos serán los tres operadores que utilizaremos, que definirán el entorno de una solución, compuesto por aquellas a las que se puede llegar mediante el uso de cualquiera de los tres. No siempre usaremos los tres, sino que dependerá del algoritmo. El operador *add* es esencial, pues nos permite ampliar el tamaño de un clique, por lo que siempre lo utilizaremos. Los otros dos operadores se usarán dependiendo del algoritmo, siendo el operador *swap* bastante frecuente, y el *drop* menos frecuente. Debido a esto, el entorno variará dependiendo de aquellos considerados.

Pasamos ya a ver los algoritmos que se han incluido en este trabajo.

### 5.2 ALGORITMOS *greedy*

Los algoritmos voraces, o algoritmos *greedy*, son aquellos que resuelven problemas tomando las decisiones óptimas en cada instante. Esto no lleva necesariamente a un óptimo global, pero puede aproximarlos de forma razonable en un periodo de tiempo generalmente muy pequeño. Este tipo de algoritmos basan sus decisiones inmediatas en las que ya se han tomado, pero nunca en lo que podría suceder más adelante.

Por lo general, en los algoritmos *greedy* tendremos un conjunto de candidatos, una función de selección, encargada de elegir al mejor candidato en cada momento, una función objetivo, que da un valor a cada solución, una función de factibilidad, que elige los candidatos que se pueden incluir en una solución, y una función que nos dice si tenemos una solución. Suelen ser algoritmos irreversibles, esto es, no pueden deshacer elecciones ya hechas, si bien existen tipos de algoritmos *greedy*, en los que esta condición se relaja. Así, distinguiremos entre algoritmos puros, relajados y ortogonales. En este trabajo se han implementado dos algoritmos *greedy*, uno puro y otro relajado.

#### 5.2.1 *Greedy en MCP*

En el problema del clique máximo, los algoritmos *greedy* tienen dos enfoques principales: o bien se parte de un conjunto vacío y se añaden vértices, respetando la estructura de clique, hasta que no sea posible añadir más, o partimos del conjunto total de vértices y vamos eliminando nodos hasta que el grafo obtenido sea un clique. Los dos algoritmos *greedy* que he implementado siguen el primero de ellos, y se diferencian principalmente en la elección del vértice que se añade al clique y en la estructura de entornos.

A continuación se detallan en más profundidad los dos algoritmos implementados.

5.2.2 *Greedy básico*

Este algoritmo es una implementación propia, que trata de crear un algoritmo lo más básico posible usando los conceptos esenciales de los algoritmos *greedy*. Comenzando desde un clique vacío, añade nodos al clique, respetando la estructura, hasta que no se pueda añadir ninguno más. Para elegir el vértice que añadir, toma, de aquellos que pueden añadirse, el que tiene mayor número de adyacencias, usando el operador *add*. Otras posibles opciones serían elegir un nodo aleatorio entre los disponibles o tomar aquel que maximice el tamaño del conjunto  $C_0$  al añadirse al clique. Este algoritmo es irreversible, pues los nodos que se añadan al clique no saldrán de él.

Vemos a continuación el pseudocódigo del algoritmo.

**Algorithm 4** Greedy

---

```

function GREEDY
  Clique = []
   $C_0$  = Vértices
  repeat
    Añadir  $v \in C_0$  con más adyacencias
    Actualizar  $C_0$ 
  until  $C_0 = \emptyset$ 
  return clique

```

---

5.2.3 *Greedy adaptativo*

En este algoritmo, se han seguido las ideas de Grosso, Locatelli y Della Croce [7], usando un algoritmo que nos permite deshacer, en cierta forma, decisiones tomadas anteriormente. La idea es no solo considerar aquellos vértices que pueden ser añadidos al clique, sino también tomar aquellos que pueden ser intercambiados por uno del clique, es decir, usar operadores *add* y *swap*.

El criterio para elegir un nodo será tomar el que maximice el número de conexiones con el conjunto  $C_0$ . Así, al añadir el nodo al grafo, el tamaño del nuevo conjunto  $C_0$  será el máximo de los posibles. Como dicho tamaño es una cota superior para el número de nodos que se pueden añadir al clique, nos interesa que sea lo más elevado posible.

Antes de empezar a considerar intercambios, queremos que el clique tenga un tamaño mínimo, por lo que al principio consideraremos solo movimientos *add* hasta alcanzar dicho tamaño. Se ha fijado el valor del tamaño mínimo en 4 para todos los grafos, a partir del cual introducimos los intercambios en el entorno. Detendremos el algoritmo una vez llegue a un clique maximal, esto es, cuando el conjunto  $C_0$  sea vacío. No obstante, se ha fijado un valor máximo de intercambios para evitar que el algoritmo se prolongue durante mucho tiempo. Si este límite se alcanza, ampliaremos el clique con movimientos *add* antes de finalizar, pues es posible que el clique que tengamos no sea maximal.

Para evitar repetición de intercambios, al hacer uno guardaremos el nodo que sale del clique, para no tenerlo en cuenta en la siguiente iteración. De no hacer esto, cabe la posibilidad de entrar en un bucle, en el que se intercambien dos nodos de forma indefinida hasta que se alcance el criterio de parada. Este nodo permanecerá como nodo tabú hasta que se haga otro intercambio.

Vemos a continuación el pseudocódigo de la parte adaptativa del algoritmo.

---

**Algorithm 5** Greedy adaptativo
 

---

```

function GREEDY(límite)
  clique = []
   $C_0 = \text{Vértices}$ ,  $C_1 = []$ 
   $i = 0$ 
  Swap = {}
  repeat
    Tomar  $v \in C_0 \cup C_1 - \text{Swap}$  que maximice adyacencias con  $C_0$ 
    if  $v \in C_0$  then
      Añadir  $v$  a clique
    else
      Hacer intercambio
      Actualizar Swap,  $i++ = 1$ 
    Actualizar  $C_0, C_1$ 
  until  $C_0 = \emptyset$  o  $i = \text{límite}$ 
  return clique
  
```

---

### 5.3 BÚSQUEDA LOCAL

Los algoritmos de búsqueda local son aquellos que se centran en el entorno de una solución para buscar nuevas soluciones que sean mejores que la anterior. Dicho entorno viene dado por aquellas soluciones a las que se puede llegar desde la solución de partida mediante un **operador de vecindad**, las llamadas **soluciones vecinas**.

Las distintas formas de recorrer el entorno y de aceptar soluciones vecinas dan lugar a los distintos algoritmos de búsqueda local. Utilizaremos un criterio de aceptación que nos conduzca a soluciones cada vez mejores, para finalizar en un óptimo local. Así, obtenemos la búsqueda local básica, que se mueve a soluciones que mejoren la actual explorando el entorno, hasta que no sea posible mejorar más.

Habrà ocasiones en las que la monotonía del criterio de aceptación no se exija de forma obligada, para permitir al algoritmo salir de óptimos locales. Debido a esto, existen algoritmos como el enfriamiento simulado o la búsqueda tabú, que son más flexibles en esta condición.

La estructura de entornos también puede ser variable dentro del propio algoritmo, para guiarlo hacia ciertas zonas del espacio de soluciones de forma dinámica. Técnicas co-

mo la búsqueda en entornos variables (VNS) o, nuevamente, la búsqueda tabú, entran dentro de estas.

Las técnicas de búsqueda local se utilizan no solo como heurísticas para resolver problemas, sino como complementos de otras. Por ejemplo, pueden ser utilizadas junto a los algoritmos genéticos para mejorar el conjunto de soluciones, dando lugar a los algoritmos meméticos, o aplicarse después de técnicas *greedy* para explorar el espacio de soluciones de la solución generada. También se usan en técnicas multiarranque como ILS o GRASP, que explicaremos más adelante.

En este trabajo, se han implementado dos técnicas de búsqueda local, que difieren en la estructura de entornos y en el método de elección de candidatos. Asimismo, se han implementado dos algoritmos de enfriamiento simulado, y se han incorporado varias de las técnicas mencionadas anteriormente en otros algoritmos. Estas serán detalladas en cada caso.

#### 5.3.1 Búsqueda local en MCP

En la búsqueda local, las diferencias que hacen a los algoritmos distintos están en la estructura de entornos y en elección de una solución vecina a la que desplazarse. Estas diferencias son también las que existen entre los dos algoritmos implementados en este trabajo. Nos valdremos de los operadores *add*, *swap* y *drop* como operadores de vecindad, si bien no siempre los tendremos todos en cuenta.

En el primer algoritmo se consideran los tres movimientos para calcular el entorno. Si bien esto rompe con la filosofía de que la búsqueda local debe ir a mejores soluciones, la posibilidad de eliminar elementos del clique se incluye para evitar que estos algoritmos sean similares a los *greedy*, buscando favorecer una búsqueda en el entorno.

En el segundo algoritmo, solo se consideran movimientos *add* y *swap*. En un principio, este algoritmo puede parecer similar al *greedy* adaptativo detallado anteriormente, aunque veremos que tiene varias diferencias fundamentales con respecto a este.

Veámoslos en más detalle.

#### 5.3.2 1LS

Este algoritmo se basa en las ideas de Katayama, Hamamoto y Narihisa [12], siendo una versión simplificada de su *k-opt local search* para el caso  $k = 1$ . El algoritmo *k-opt* consiste en tomar todas las posibles combinaciones de  $k$  movimientos para generar el entorno de un clique, y explorar este entorno siguiendo diversas técnicas (en este caso, una búsqueda local). La versión considerada en este trabajo toma 1 como valor de  $k$ , por lo que nos referiremos al algoritmo por **1LS**.

Para crear el entorno, se han considerado los tres operadores de vecino, *add*, *swap* y *drop*. Daremos más prioridad a *add*, pues nuestro objetivo es construir un clique del mayor ta-



maño posible, y tomaremos el nodo que tenga más conexiones con el conjunto  $C_0$ . Si no es posible añadir un nodo intentaremos hacer un *swap*, siempre que este intercambio haga que podamos añadir más nodos al clique resultante. Buscaremos posibles intercambios, y efectuaremos el primero que satisfaga esta condición. De no haber ninguno, eliminaremos del clique el nodo que tenga menor número de adyacencias

Para evitar que los nodos que eliminemos sean añadidos inmediatamente al clique, los iremos acumulando en una lista tabú. Los nodos de esta lista no se tendrán en cuenta a la hora de hacer un *add*, lo que evita que añadamos un nodo justo tras sacarlo del clique. Esta lista tabú será vaciada en el momento en el que se realice un movimiento que no sea quitar un nodo del clique, ya que nos habremos asegurado de que el clique incluye al menos un nodo que no contenía antes de comenzar a eliminar nodos.

Este algoritmo no tiene un criterio de parada inducido por el entorno, pues siempre podremos hacer alguno de los tres movimientos. Por tanto, es necesario establecer un límite, que este caso, ha sido establecer un límite de swaps y drops. Una vez alcanzado este límite, el algoritmo finaliza, devolviendo el mejor clique encontrado. Para evitar que el límite sea una constante común para todas las instancias, se ha establecido en proporción al número de vértices del grafo, haciendo que el algoritmo se adapte al tamaño de cada problema.

Es posible combinar este algoritmo con otros, ya sea para tratar de mejorar soluciones como la de un *greedy*, o hibridarlo con otras técnicas como algoritmos genéticos. Para evaluar sus prestaciones por separado, partirá desde un clique vacío.

Vemos el pseudocódigo del algoritmo:

---

**Algorithm 6** 1LS
 

---

```

function 1LS(clique, límite)
   $i = 0$ , tabú =  $\emptyset$ 
  Calcular  $C_0$  y  $C_1$ .
  repeat
    if  $C_0 \neq \emptyset$  then
      Añadir el nodo de  $C_0$  que tenga más conexiones con  $C_0$ .
      tabú =  $\emptyset$ 
    else
      Buscar en  $C_1$  el primer intercambio que nos da un  $C_0$  no vacío.
      if El intercambio existe then
        Hacer intercambio
        tabú =  $\emptyset$ 
         $i = i + 1$ 
      else
        Quitar el nodo del clique con menos adyacencias y añadirlo a tabú.
         $i = i + 1$ 
      Actualizar  $C_0, C_1$ , teniendo en cuenta tabú.
  until  $i = \text{límite}$ 
  return Mejor clique encontrado
  
```

---

### 5.3.3 *Dynamic Local Search*

Este algoritmo está basado en la búsqueda local dinámica o DLS (del inglés, *Dynamic Local Search*), propuesta por W. Pullan y H. Hoos [16], y en concreto, se basa en una adaptación hecha por A. Grosso, M. Locatelli y W. Pullan para un algoritmo de búsqueda local iterada [8], que veremos más adelante. Sigue una filosofía más simple que el anterior, pues solo considera movimientos *add* y *swap*, a los que volveremos a ordenar por prioridad. Además, incluye una lista tabú con los nodos que han salido del clique mediante un *swap*, evitando que vuelvan a entrar al clique.

El entorno consistirá en el conjunto de posibles adiciones ( $C_0$ ) y el conjunto de posibles intercambios ( $C_1$ ), al que quitaremos la lista tabú. Es posible que el entorno sea vacío, y en ese caso, el algoritmo finalizaría. No obstante, se ha establecido un límite de intercambios, pues en la práctica el entorno no tiene por qué ser vacío.

La prioridad entre movimientos es la siguiente: primero intentaremos añadir un elemento de  $C_0$  que no esté en la lista tabú, tomando uno aleatorio en caso de existir varios. De no existir ninguno, tomaremos el conjunto  $C_1$  y le quitaremos la lista tabú. En caso de quedar algún nodo, haremos el intercambio, que volverá a ser aleatorio si tenemos varios candidatos, y añadiremos el nodo que sale del clique a la lista tabú. Finalmente, si el conjunto  $C_1$  menos la lista tabú fuera vacío, nos queda la posibilidad de añadir un elemento de la lista tabú que esté en  $C_0$ . Nuevamente, elegiremos uno aleatorio entre los existentes. Este proceso se repetirá hasta que se satisfaga uno de los dos criterios de parada. Una vez acabe el algoritmo, devolveremos el mejor clique que haya encontrado.

Al igual que el anterior, es posible utilizar este algoritmo como complemento a otras técnicas, bien sea mediante hibridaciones o para mejorar las soluciones obtenidas por otros algoritmos. Para evaluar su funcionamiento por separado, volveremos a tomar como clique inicial el clique vacío, para poder comparar los dos algoritmos bajo las mismas condiciones de partida.

Pasamos a ver el pseudocódigo del algoritmo:

---

**Algorithm 7** DLS

---

```

function DLS(clique, límite)
   $i = 0, \text{tabú} = \emptyset$ 
  Calcular  $C_0$  y  $C_1$ .
  repeat
    if  $C_0 - \text{tabú} \neq \emptyset$  then
      Añadir un nodo aleatorio de  $C_0 - \text{tabú}$ 
    else if  $C_1 - \text{tabú} \neq \emptyset$  then
      Hacer un cambio con un nodo aleatorio de  $C_1 - \text{tabú}$ .
      Añadir el nodo que sale del clique a tabú.
       $i = i + 1$ 
    else if  $C_0 \neq \emptyset$  then
      Añadir un nodo aleatorio de  $C_0$ .
  Actualizar  $C_0, C_1$ .
  entorno =  $C_0 \cup (C_1 - \text{tabú})$ 
until  $i = \text{límite}$  o entorno =  $\emptyset$ 
return Mejor clique encontrado

```

---

## 5.4 ENFRIAMIENTO SIMULADO

El enfriamiento simulado [13] es un algoritmo de búsqueda en entornos, que incluye un criterio probabilístico de aceptación de soluciones basado en la termodinámica, el algoritmo de Metrópolis. Su principal objetivo es evitar que la búsqueda en el entorno finalice en un óptimo local, cosa que ocurre con la búsqueda local. Para esto, permitirá que algunos movimientos vayan a soluciones peores, usando una función de probabilidad que decida si se acepta una solución de menos calidad. Esta probabilidad vendrá dada por el algoritmo de Metrópolis, e irá descendiendo a medida que avanza el algoritmo. Así, conseguimos una diversificación de soluciones al principio, mientras que al final del algoritmo intensificamos la búsqueda en una región concreta del espacio de soluciones.

El algoritmo de Metrópolis utiliza dos valores para calcular la probabilidad. Uno de ellos es la diferencia entre los valores de la función objetivo de la solución del entorno y la solución actual. El otro es un valor llamado temperatura, en referencia a su origen termodinámico, que desciende en cada iteración de la búsqueda, lo que permitirá que la probabilidad descienda con el valor.

Su forma de actuar es la siguiente: si nos movemos a una solución que mejore el resultado anterior, la aceptamos directamente. De no ser así, calculamos la diferencia del valor de la función objetivo, que notamos  $\Delta F$ , y tomamos como probabilidad de aceptación  $e^{\Delta F/T}$ , donde  $T$  representa la temperatura. Como  $\Delta F \leq 0$ , este valor estará siempre entre 0 y 1, siendo más cercano a 0 cuanto menor sean la temperatura y  $\Delta F$ .

El decrecimiento de la temperatura nos proporciona también un criterio de parada. Fijando una temperatura final, usualmente cercana a 0, el algoritmo se ejecutará hasta que

la temperatura alcance dicho valor. Así, conseguimos que la probabilidad de aceptación de soluciones peores sea mayor al principio, favoreciendo la exploración, mientras que en las fases finales la probabilidad de aceptación será baja, y el algoritmo intensificará la búsqueda en la región del espacio de soluciones que se encuentre. Este decrecimiento puede seguir distintos esquemas, aunque el más usual es el geométrico, consistente en multiplicar la temperatura por una constante menor que 1 en cada iteración, si bien cualquier función estrictamente decreciente nos resultaría útil para realizarlo.

#### 5.4.1 *Enfriamiento simulado en MCP*

El principal problema a la hora de elaborar un algoritmo de enfriamiento simulado en MCP es la elección de una función objetivo que permita una exploración amplia del entorno. Las funciones objetivo usuales, como el tamaño del clique, no aportan diversidad de soluciones, pues una vez lleguemos a un óptimo local, normalmente repetirá movimientos de intercambio de forma indefinida, ya que estos no cambian el valor de la función objetivo, y son aceptados por el criterio de Metrópolis.

Establecer una prioridad entre los tres operadores de vecindad daría como resultado un algoritmo muy similar a la búsqueda local, que seguiría teniendo los mismos problemas. Por lo tanto, evitaremos hacerlo en este caso, y centraremos el esfuerzo en buscar una función objetivo que permita una mayor exploración del entorno. No obstante, hemos de tener en cuenta que el algoritmo debe seguir convergiendo a un óptimo local, por lo que no nos vale cualquier función.

Es posible encontrar algunos trabajos sobre MCP que involucran el enfriamiento simulado. Sin embargo, se suelen basar en una representación y una estructura diferentes a las consideradas en este problema, siendo imposible trasladarlos completamente a este caso, aunque sí pueden traerse algunas de las ideas principales. En el caso de este trabajo, se ha implementado un algoritmo que utiliza las nociones básicas de enfriamiento simulado para ver sus diferencias con la búsqueda local, y un segundo algoritmo que recoge algunas ideas de un artículo sobre el problema.

#### 5.4.2 *Enfriamiento simulado básico*

Este algoritmo es una implementación propia, que contiene una adaptación de una búsqueda local a un enfriamiento simulado, utilizando una función objetivo basada en el tamaño de los cliques. En concreto, el valor de cada clique es su tamaño más el número de vértices que contiene el conjunto  $C_0$  dividido por el número de vértices totales, que notaremos  $N$ .

$$f(C) = |C| + \frac{|C_0|}{N}$$

De esta forma, conseguimos que cliques de mayor tamaño tengan un valor mayor, lo que garantiza la convergencia a un óptimo local. En caso de empate, utilizamos el

tamaño del conjunto  $C_0$  para determinar el mejor, pues, como hemos discutido con anterioridad, cuanto mayor sea  $C_0$  más puede crecer el clique.

Para el entorno, he considerado movimientos *add*, *swap* y *drop*, sin priorizar ninguno de ellos. Se tomarán en orden aleatorio hasta encontrar uno que mejore la función objetivo o satisfaga el criterio de Metrópolis.

Los valores de temperatura han sido inicializados a  $T_{\text{inicial}} = 1$ ,  $T_{\text{final}} = 0,001$ , con un mecanismo de enfriamiento geométrico, multiplicando la temperatura por una constante  $\beta$  en cada iteración, fijada a  $\beta = 0,99$ . Una vez se alcance una temperatura menor que la final, completamos el mejor clique encontrado añadiendo tantos nodos como sea posible (aunque lo más probable es que no se añada ninguno), eligiendo dichos nodos por mayor número de adyacencias. Finalmente, el algoritmo devuelve el clique y finaliza.

El algoritmo trabaja partiendo siempre del clique vacío. No obstante, puede ser adaptado fácilmente para que comience la exploración desde cualquier clique, lo que nos permitiría combinarlo con otros algoritmos para obtener técnicas híbridas.

Vemos una descripción del algoritmo en pseudocódigo.

---

**Algorithm 8** Enfriamiento Simulado

---

```

function SA
   $T = 1, T_{\text{final}} = 0,001, \beta = 0,99$ 
  Calcular  $C_0$  y  $C_1$ .
  repeat
    Entorno =  $C_0 \cup C_1 \cup \text{Drops}$ 
    repeat
      Sacar un clique del entorno, calcular  $\Delta F$ .
      if  $\Delta F \geq 0$  then
        Aceptar solución.
        Salir del bucle interno.
      else
        Aceptar solución con probabilidad  $e^{\Delta F/T}$ .
        Salir del bucle interno si se acepta la solución.
    until Entorno =  $\emptyset$ .
     $T = T \times \beta$ 
  until  $T < T_{\text{final}}$ 
  Ampliar mejor clique encontrado.
  return mejor clique

```

---

### 5.4.3 Enfriamiento simulado adaptado

En el segundo algoritmo, se han adaptado las ideas de X. Geng, J. Xu, J. Xiao y L. Pan [5], que propusieron un algoritmo de enfriamiento simulado para MCP. Dicho algoritmo sigue un enfoque distinto al de este problema, pues intenta buscar cliques de

un tamaño predeterminado, explorando el espacio de soluciones hasta que lo encuentra, o hasta llegar al criterio de parada. Se basa en una representación por permutaciones del vector de vértices, y va realizando intercambios de dos vértices, de tal forma que uno sale del subgrafo y otro entra en él, siempre que la función objetivo mejore o se satisfaga el criterio de Metrópolis. En este caso, en lugar de ir moviéndose entre cliques, se consideran grafos de tamaño fijo, que se modificarán con el objetivo de alcanzar un clique, lo que permite definir como función objetivo el número de aristas que le faltan al grafo para ser un clique. Para poder adaptarme a este enfoque, he tenido que diseñar un algoritmo que se moviera entre grafos que no fueran necesariamente cliques, para lo que he utilizado los mismos tres operadores, *add*, *swap* y *drop*, pero en este caso, he permitido añadir cualquier nodo al grafo o intercambiar dos nodos cualesquiera, sin necesidad de que el resultado sea un clique. La exploración del nuevo entorno se hace igual que en el caso anterior, de forma aleatoria.

Al principio tomé la misma función objetivo que la propuesta por Geng et al., consistente en el número de aristas necesarias para que el grafo sea un clique. Sin embargo, esta función no asegura una convergencia a soluciones de calidad; en particular, es ajena al tamaño de los cliques. Por esto, he usado la longitud de los grafos para corregir esta, restándola al número de aristas restante. Así, tenemos una función objetivo que es menor cuanto mayor es el grafo, y cuanto más se parece a un clique.

Una vez definida la función objetivo, el algoritmo explora el entorno y toma un elemento aleatorio hasta que la función objetivo es menor, o hasta que se acepta el criterio de Metrópolis. Como ahora no trabajamos con cliques, el grafo que obtenemos será reducido a un clique, para lo que eliminaremos el nodo que menos adyacencias tenga hasta obtener uno. El clique obtenido no sustituye al grafo que tengamos, sino que se compara con el mejor clique obtenido hasta el momento, reemplazándolo si su longitud fuera mayor. Después del algoritmo continua explorando el entorno del grafo, repitiendo el método descrito. La temperatura vuelve a seguir un esquema de descenso geométrico.

He vuelto a tomar los valores  $T_{\text{inicial}} = 1$ ,  $T_{\text{final}} = 0,001$  y  $\beta = 0,99$  para los parámetros del algoritmo. Una vez finaliza la fase de enfriamiento, vuelvo a añadir tantos nodos como sea posible al mejor clique encontrado, mediante un algoritmo *greedy* que prioriza los nodos con más adyacencias, devolviendo el clique resultante. Dado que las principales diferencias con el anterior algoritmo son el espacio de búsqueda, y la función objetivo, no se incluye pseudocódigo de este algoritmo, por ser similar al anterior.

## 5.5 BÚSQUEDAS MULTIARRANQUE

Las búsquedas multiarranque son un tipo de búsqueda local, que intentan salir de óptimos locales reiniciando la búsqueda para tomar otra solución de partida, para así explorar una región distinta del espacio de soluciones. Dicha reinicialización de la búsqueda será la clave para distinguir entre búsquedas multiarranque, y podemos clasificarlas como sigue:

- Búsqueda multiarranque básica, que genera una solución inicial aleatoria y explora su entorno mediante búsqueda local, repitiendo el proceso hasta que se satisfaga una condición de parada. Es el método más sencillo, que no utiliza ningún tipo de información del problema, y equivale a aplicar una búsqueda local múltiples veces sobre distintas soluciones aleatorias de partida.
- Métodos constructivos de la solución inicial, donde esta varía en cada iteración. En este caso, la solución inicial se obtiene mediante una construcción, intentando conseguir soluciones de partida que sean de calidad. Intentaremos que las soluciones sean distintas en cada iteración, por lo que el proceso de construcción será usualmente aleatorizado. Una vez se tenga la solución inicial, se aplica búsqueda local, repitiendo hasta satisfacer una condición de parada, normalmente un número de iteraciones fijo de antemano.
- Métodos basados en la modificación del óptimo encontrado, en los que el óptimo local se somete a una perturbación, dando lugar a una nueva solución de partida. Con esto utilizamos una solución ya calculada para proporcionar una nueva solución de partida, con la que intentaremos mejorar la calidad de la solución anterior. Nuevamente, repetiremos el proceso hasta alcanzar una condición, que habitualmente será un número de repeticiones dado.

En este trabajo he considerado dos tipos de algoritmos multiarranque: algoritmos GRASP (*Greedy Randomized Adaptive Search Procedure*), basados en la construcción de la solución inicial, y algoritmos ILS (*Iterated Local Search*), basados en modificaciones de la solución encontrada. Vamos a detallarlos.

#### 5.5.1 GRASP

El algoritmo GRASP [3], o *greedy* aleatorizado, es un algoritmo multiarranque que consiste en repetir una fase de construcción de una solución, seguida de una búsqueda local en el entorno de dicha solución, hasta que se alcance una condición, que suele ser un límite de iteraciones o de tiempo de ejecución. Es un algoritmo sencillo de programar, y su capacidad para obtener buenas soluciones está ligada a si los algoritmos *greedy* y la búsqueda local ofrecen buenos resultados sobre el problema, pues está fundamentado en ellos. Fue usado por primera vez en 1989 por T. A. Feo y M. G. C. Resende [2], aunque la idea había sido descrita en 1987 por J. P. Hart y A. W. Shogan [9].

La construcción de la solución inicial sigue un procedimiento *greedy*, en el cual, en lugar de proporcionar un único candidato como se haría en un algoritmo *greedy* clásico, se proporciona una lista restringida de candidatos (en inglés, *restricted candidate list*, o RCL), que es un conjunto de los mejores candidatos, tomados de entre todos los posibles, y de entre estos, elegiremos uno al azar. El tamaño de RCL es un parámetro, que dependiendo de su valor hace a la construcción más similar a un *greedy* o a un algoritmo aleatorio, pues notemos que si el número de candidatos es 1, el algoritmo será un *greedy* puro, mientras que si es el tamaño de la lista de candidatos, el algoritmo será aleatorio [20].

Esta técnica se utiliza para proporcionar diversidad, pues esta se pierde al tomar soluciones iniciales generadas por algoritmos voraces. Así, seguimos obteniendo buenas soluciones de partida, a la vez que estas difieren lo suficiente como para explorar una región amplia del espacio de soluciones.

Una vez construída la solución inicial, se explora su entorno siguiendo una búsqueda local, hasta alcanzar un óptimo local. Cuando es alcanzado, se compara la solución con la mejor obtenida hasta el momento, se actualiza si es necesario, y se comienza de nuevo el algoritmo. Se repetirá el proceso hasta satisfacer una condición de parada.

#### 5.5.1.1 GRASP en MCP

El aspecto fundamental de este algoritmo es la construcción de la solución inicial. Podemos valorar los elementos del conjunto de candidatos de distintas formas, lo que dará lugar a distintas soluciones de partida. Por lo demás, el algoritmo se basa en una búsqueda local, que hemos discutido con anterioridad.

#### 5.5.1.2 Algoritmo implementado

Para generar la solución inicial he tomado la lista de candidatos,  $C_0$ , y la he ordenado por número de adyacencias en el grafo, en orden descendente. Una vez ordenada, tomo la mitad superior como conjunto de candidatos, y hago una elección aleatoria entre los nodos, en la que todos tienen la misma probabilidad de ser elegidos. El proceso se repetirá hasta que  $C_0$  sea vacío.

La fase de búsqueda local utiliza los dos métodos ya implementados, 1LS y DLS, lo que da lugar a dos versiones del algoritmo, en las que solo varía esta fase. Como estos métodos no se detienen en un óptimo, sino que necesitan un número de iteraciones máximo, he establecido como límite el número de vértices del grafo, para que se adapte al tamaño de cada problema. Este proceso se repetirá un número determinado de iteraciones, que se pasará como parámetro, y que en este trabajo se ha fijado a 20 en todas las instancias disponibles. Almacenaremos la mejor solución obtenida, que será la que el algoritmo devuelva al final.

Vemos el pseudocódigo del algoritmo y de la generación de soluciones iniciales.

---

**Algorithm 9** GRASP

---

```
function GRASP(límite)
   $i = 0$ 
  repeat
    Obtener solución inicial.
    Aplicar búsqueda local a la solución inicial.
    Actualizar mejor solución.
     $i = i + 1$ 
  until  $i = \text{límite}$ 
  return mejor clique
```

---



---

**Algorithm 10** Generación de soluciones aleatorias

---

```
clique =  $\emptyset$ 
 $C_0$  = vértices
repeat
  Ordenar  $C_0$  por adyacencias.
  Tomar la mitad superior de  $C_0$ .
  Añadir un elemento aleatorio a clique.
  Recalcular  $C_0$ .
until  $C_0 = \emptyset$ 
return clique
```

---

### 5.5.2 ILS

La búsqueda local iterada, o ILS (del inglés, *Iterated Local Search*), es un algoritmo de búsqueda local multiarranque basado en la perturbación de soluciones como método para obtener diversidad. Fue propuesta en 1998 por Thomas Stützle en su tesis doctoral [17], y desde entonces se utiliza en la resolución de distintos problemas, entre los que se encuentra el problema del clique máximo.

El algoritmo ILS requiere de cuatro componentes principales: una solución inicial, un algoritmo de mutación de soluciones, un método de búsqueda local, y un criterio de aceptación. Definiendo estos cuatro componentes en un problema, estaremos en condiciones de aplicar el algoritmo.

El funcionamiento general de este tipo de algoritmos es el siguiente: primero, se toma la solución inicial y se le aplica búsqueda local, hasta llegar a un óptimo local. Seguidamente, se le aplica una mutación a la solución obtenida, que nos permite crear una nueva solución inicial a la que volver a aplicar búsqueda local. Una vez finalizada esta segunda búsqueda local, se usa el criterio de aceptación para ver si reemplazamos la solución anterior con la obtenida. Este proceso mutación-búsqueda local-aceptación se repetirá hasta satisfacer un criterio de parada.

Este modelo de ILS es el más básico que existe, pero no es el único. Podemos utilizar un modelo basado en poblaciones, donde a cada elemento se le aplica ILS y estos se combinan de alguna forma para una mayor exploración del espacio de soluciones, o el modelo  $(\nu + \lambda)$ , donde, partiendo de  $\nu$  soluciones iniciales, generamos  $\lambda$  hijos a partir de ellas mediante mutación, que compiten para pasar a la siguiente fase. También es posible establecer diferencias en el criterio de aceptación. Si bien el método más usual es tener una función objetivo, pueden utilizarse otros, como una selección aleatoria, una selección basada en combinaciones, o una basada en probabilidades, en la que podría utilizarse el algoritmo de Metrópolis, por ejemplo.

La magnitud de la perturbación es muy importante, pues una perturbación demasiado débil podría hacernos caer en los mismos óptimos locales de forma sucesiva, mientras que una perturbación demasiado fuerte nos haría perder información de la solución y actuaría de forma similar a un reinicio aleatorio [20].

### 5.5.2.1 ILS en MCP

Es sencillo construir una búsqueda local iterada para el problema del clique máximo, partiendo de un algoritmo de búsqueda local existente. Para ello, son necesarios un criterio de aceptación, una solución inicial y una función de mutación.

### 5.5.2.2 Algoritmo implementado

Se ha implementado un algoritmo que sigue la filosofía de la búsqueda local iterada básica basado en el propuesto por A. Grosso, M. Locatelli y W. Pullan [8], que usa el algoritmo de búsqueda local DLS [16], si bien se propusieron algunas modificaciones que aquí no se han tenido en cuenta. Además de DLS he utilizado 1LS, aprovechando que ya estaba implementado. Esto nos da dos algoritmos ILS distintos, que difieren únicamente en la búsqueda local.

El algoritmo para mutar un clique consiste en tomar un nodo que no se encuentre en él, eliminar del clique todos los nodos que no estén conectados a él, y finalmente añadir dicho nodo al clique. Con este método conseguimos una solución de partida que se encuentra en una zona distinta del espacio de soluciones, a la vez que usamos información de la iteración anterior para movernos por soluciones que serán potencialmente de calidad.

La función de aceptación se queda siempre con el último clique obtenido tras la búsqueda local. Tras valorar si quedarme con la última o con la mejor encontrada, decidí hacer lo primero, para favorecer la diversidad en las soluciones. No obstante, tendremos almacenada la mejor solución, al ser el valor de salida del programa.

Dado que las dos búsquedas locales implementadas requieren de un límite como criterio de parada, he fijado dicho límite al número de vértices del grafo, para que se adapte al tamaño de cada problema. Además, el criterio de parada del algoritmo también es un número de iteraciones, que ha sido 20 para todas las instancias.

Vemos el pseudocódigo del algoritmo:

---

#### Algorithm 11 ILS

---

```

function ILS
  Generar solución inicial.
  Aplicar búsqueda local.
  repeat
    Modificar solución.
    Aplicar búsqueda local a la solución obtenida.
    Actualizar la mejor solución, si es necesario.
  until Se alcance el número de iteraciones.
  return mejor solución obtenida

```

---

## 5.6 ALGORITMOS DE COLONIA DE HORMIGAS

Los **algoritmos de colonia de hormigas** [1] o ACO (del inglés, *Ant Colony Optimization*) son una técnica de adaptación social, que se basa en el comportamiento de las colonias de hormigas para tratar de encontrar caminos en grafos. Para ello, utilizarán un recurso que emula a las feromonas que emiten las hormigas cuando se desplazan.

Las hormigas utilizan este rastro de feromonas para guiarse, pues poseen unos receptores capaces de captar la intensidad de dicho rastro. Dado su pésimo sentido de la vista, el uso de las feromonas es vital para que las hormigas se desplacen y sean capaces de localizar fuentes de alimento, volver al hormiguero, y que el resto de la colonia también sea capaz de llegar a la comida, siguiendo el rastro que las propias hormigas dejan. Este rastro de feromonas no permanece de forma indefinida en el ambiente, sino que sufre un proceso de evaporación constante. De esta forma, caminos más cortos tenderán a una mayor concentración de feromonas, lo que incrementará el tránsito de hormigas, mientras que los más largos se verán más afectados por la evaporación, y así, tendrán un menor nivel de feromonas, lo que a la larga hará que dejen de ser visitados.

La elección por parte de una hormiga del camino que recorrer se fundamenta en un criterio probabilístico. La hormiga tiende a recorrer caminos en los que el rastro de feromonas es mayor, aunque podrá ir por otros donde el rastro no sea tan intenso.

Su fundamentación teórica hace que sea un algoritmo muy eficaz para problemas que requieren de construcción de caminos, como el problema del viajante de comercio. En general, son algoritmos que funcionan bien cuando trabajan sobre estructuras similares a los grafos.

Este comportamiento será trasladado a la optimización por colonia de hormigas, mediante la construcción de hormigas artificiales. Tendremos un rastro de feromonas artificial, que simula el que dejan las hormigas al recorrer un camino. Es posible complementar el uso de las feromonas con información heurística del problema, con el objetivo de crear un mecanismo probabilístico de construcción de soluciones que no solo se base en feromonas, sino que use más información disponible.

Las hormigas construirán un camino de forma incremental utilizando la información de la que disponen, comenzando desde un nodo aleatorio y eligiendo el siguiente mediante una función de probabilidad, que depende de los valores de feromonas.

Los nodos que compongan una solución serán reforzados con un aumento de feromonas, y se hará de forma que mejores soluciones reciban más cantidad de feromonas. Se utiliza también un mecanismo de evaporación de feromona simulando el que ocurre en la naturaleza, que evita un crecimiento ilimitado de la misma. Para esto, se reducirán todos los rastros de feromonas eliminando un parte de su valor actual, usualmente un porcentaje fijo. En la práctica, esto se traduce en evitar un estancamiento del algoritmo, debido a que los nodos más visitados tendrían altas probabilidades de repetirse en la siguiente solución.

Se lanzarán un número determinado de hormigas artificiales independientes, las cuales generan una solución cada una. Entre ellas, se toma la mejor, y esta será la que cambie

los niveles de feromonas. Este proceso se repite hasta que se satisfaga una condición de parada, que suele ser un número de iteraciones o un límite temporal.

Existen numerosas variantes de las colonias de hormigas, dependiendo de la función de probabilidad, la actualización de feromonas, etcétera. En este trabajo he tomado el algoritmo más clásico, sin entrar en otros que requerirían de mayor experimentación para hacer ajustes sobre los parámetros. También es posible hibridar este método con otros, usualmente un algoritmo de búsqueda basado en entornos, como una búsqueda local o un enfriamiento simulado.

#### 5.6.1 ACO en MCP

Al ser el problema del clique máximo un problema de grafos, los algoritmos de colonias de hormigas son fácilmente implementables, ya que adaptar el problema a una búsqueda de caminos en grafos es directo. Trabajaremos con conjuntos de candidatos sobre los cuales podremos realizar una acción, como un *add* o un *swap*.

En este trabajo, he considerado dos algoritmos de colonias de hormigas clásicos, lanzando una colonia de hormigas durante un número determinado de iteraciones y reforzando el nivel de feromonas en la mejor solución de cada iteración. La evaporación se realiza reduciendo un porcentaje fijo de feromonas. La diferencia entre ambos algoritmos reside en la función de probabilidad; mientras que uno de ellos utiliza solo el valor de feromona, el segundo complementa su uso con información del grafo.

#### 5.6.2 ACO básico

El primer algoritmo es una implementación propia, consistente en una colonia de hormigas de tamaño fijo, en la que las hormigas crean una solución cada una, siendo la mejor la que se tiene en cuenta para modificar los niveles de feromona. He considerado únicamente movimientos *add*, por lo que las hormigas irán recorriendo los nodos del conjunto  $C_0$  hasta que este sea vacío.

Cada hormiga comenzará desde un clique vacío y añadirá nodos del conjunto  $C_0$  según una regla probabilística, que consiste en dar a cada nodo del conjunto una probabilidad proporcional a su feromona, normalizándola para obtener un valor entre 0 y 1, y conseguir que la suma de todas las probabilidades de los nodos de  $C_0$  sea 1. Para ello, solo hemos de dividir el valor de la feromona de un nodo por la suma de los valores de los nodos de  $C_0$ . Inicialmente, la probabilidad será la misma para cada nodo, por lo que no damos ventaja a ninguno y dejamos que sea el proceso el que determine los mejores nodos.

El proceso se repite hasta que la hormiga finaliza la creación de su clique, dando paso a la siguiente, que crea uno distinto, y así hasta que todas las hormigas completen el suyo. Cada vez que una hormiga finaliza, compara su clique con el mejor obtenido en

esa iteración, sustituyéndolo de ser necesario. Así, tendremos siempre el mejor clique creado por la colonia a lo largo de cada iteración.

Una vez finalizado el trabajo de todas las hormigas, se produce la evaporación de feromona, consistente en multiplicar los valores en cada nodo por una constante  $\beta \in (0, 1)$ . Seguidamente, se aumentan los niveles de feromona en los nodos de la mejor solución, multiplicando el valor de esta por un valor  $\alpha$ . Este valor dependerá del tamaño del clique encontrado, siendo mayor cuanto más tamaño tiene el clique. Así, conseguimos dar más peso a soluciones de más calidad. El proceso se repite un número fijado de iteraciones. Una vez terminadas todas las iteraciones, el algoritmo finaliza y devuelve la mejor solución encontrada.

Se han fijado los valores  $\beta = 0,925$ ,  $\alpha = (5 \times |C| + N)/N$ , donde  $|C|$  es el tamaño del clique encontrado, y  $N$  es el número de vértices del grafo. Para fijarlos, me he valido de la experimentación en instancias pequeñas del problema, tomando aquellos que daban mejores resultados. No obstante, las diferencias eran mínimas entre todos los valores probados. El número de hormigas en cada iteración ha sido fijado a 50, efectuándose un total de 40 iteraciones.

Vemos el pseudocódigo del algoritmo:

---

**Algorithm 12** ACO básico
 

---

```

function ACO(límite, límite hormigas)
   $T_{\text{final}} = 0,01$ 
  Inicializar constantes  $\alpha, \beta$ .
  MejorClique = [].
  repeat
    Hormigas = 0.
    Establecer el mejor clique de la colonia al clique vacío.
    repeat
      Clique = []
      Añadir vértice aleatorio al clique
      Calcular  $C_0$ 
      repeat
        Calcular probabilidades de los nodos en  $C_0$ . Normalizar.
        Elegir un nodo según la probabilidad y añadirlo a clique.
        Recalcular  $C_0$ .
      until  $C_0 = \emptyset$ 
      Actualizar el mejor clique de la colonia si es necesario.
      Hormigas = Hormigas + 1.
    until Hormigas = límitehormigas
    Actualizar MejorClique si es necesario.
    Evaporar feromona.
    Aportar feromona a los nodos del mejor clique de la colonia.
     $i = i + 1$ .
  until  $i = \text{límite}$ 
  return mejor clique
  
```

---

### 5.6.3 ACO con enfriamiento simulado

Este algoritmo toma algunas de las ideas de X. Xu, J. Ma y J. Lei [19], que trata de mejorar el ACO básico considerando información relativa al clique para calcular las probabilidades de elegir un nodo. La única diferencia con el anterior está dicho cálculo, para el que utilizaremos una técnica de enfriamiento simulado, además del uso de los valores de feromonas.

La probabilidad de cada nodo se compone de dos partes: la primera de ellas es idéntica a la anterior, un valor entre 0 y 1 obtenido de las feromonas de cada nodo en  $C_0$  normalizada, para que la suma total sea 1. Además, sumaremos un valor que depende del grado de cada nodo, y estará multiplicado por una temperatura, de forma que la importancia de este término descende en el tiempo. La fórmula para el cálculo de la probabilidad queda de la siguiente forma:

$$p(v_i) = \frac{\tau(v_i)}{\sum_{v_j \in C_0} \tau(v_j)} + T \frac{D(v_i)}{N}, \quad v_i \in C_0$$

donde  $\tau(v_i)$  es el valor de feromonas en el nodo  $v_i$ ,  $T$  es la temperatura,  $N$  el número de nodos del grafo, y  $D(v_i)$  es el grado de  $v_i$ . Una vez calculadas todas las probabilidades de los nodos de  $C_0$ , volveremos a normalizarlas, para tener valores entre 0 y 1 y que la suma siga siendo 1.

La temperatura sigue un esquema de descenso geométrico similar al utilizado en [Subsección 5.4.2](#) y [Subsección 5.4.3](#), en los que la temperatura se multiplica por una constante  $\gamma$  en cada iteración, tomando  $\gamma = 0,95$ . Los valores de  $\alpha$  y  $\beta$  siguen siendo los mismos que en el algoritmo anterior, al igual que el número de hormigas y de iteraciones, que son 50 y 40 respectivamente.

Puesto que solo cambia la función del cálculo de la probabilidad, no incluyo pseudocódigo de este algoritmo, por ser similar al anterior.

## 5.7 ALGORITMOS GENÉTICOS

Los algoritmos genéticos entran dentro de la llamada **computación evolutiva**, que está compuesta por modelos de evolución basados en poblaciones. Se basan en los procesos evolutivos presentes en la naturaleza, en los que dos padres combinan sus genes para producir hijos, los cuales presentan características de ambos. Las diferencias entre generaciones pueden llevar a nuevas configuraciones que devengan en una mejor adaptación al entorno. Esto, en el ámbito que nos encontramos, se traduce en cambios que nos lleven mejores soluciones, pues estas se adaptan mejor al problema [20].

En los algoritmos genéticos, las poblaciones estarán formadas por soluciones al problema, que combinaremos dos a dos para producir descendientes que intenten mejorar a sus progenitores. Esta combinación de soluciones se produce mediante el llamado **operador de cruce**, que emula cómo los cromosomas se mezclan para producir una

nueva generación, y cuyo requisito fundamental es que se tome información de ambos progenitores para crear a sus descendientes.

En la naturaleza, es posible que al combinar dos individuos se produzca una mutación, con lo cual el hijo tendría una característica que no ha adquirido de ninguno de sus padres, y que podría conllevar una ventaja o un perjuicio para el mismo. Esta técnica se reproducirá en los algoritmos genéticos, incluyendo la posibilidad de que una solución mute al ser generada. Al igual que el operador de cruce, las mutaciones pueden seguir distintos enfoques, aunque se suele requerir que introduzcan modificaciones significativas en los elementos de la población.

Controlaremos el cruce entre individuos y las mutaciones de sus descendientes con parámetros que establezcan la probabilidad de que ambos sucesos ocurran. Para el cruce, seleccionaremos dos individuos que o bien generarán descendientes, o bien pasarán directamente a la siguiente generación. En caso de generar descendientes, cada uno de ellos podrá ser mutado en el momento de su creación.

Existen distintas formas de seleccionar a los padres. El muestreo puede ser aleatorio, se puede dar mayor probabilidad de combinarse a los mejores, etc. No hay restricción en cuanto al número de veces que se puede reproducir un elemento, si bien podría establecerse de considerarse necesario.

De igual forma, existen diversos métodos para crear una nueva generación. Una de ellas sería que los hijos sustituyan a los padres, el llamado modelo *generacional*. En este caso, se crea una nueva población en cada iteración del ciclo. Es posible incluir elitismo, de forma que la mejor solución de la población pasa siempre a la siguiente generación, lo que nos permite conservarla siempre sin necesidad de almacenarla explícitamente. Otro modelo es el llamado *estacionario*, en el cual se eligen dos padres, que generan dos hijos, y estos compiten para pasar a la siguiente generación, pudiendo hacerlo con los padres o con la población en general. Este modelo ya es elitista, luego no necesita introducir elitismo explícitamente.

### 5.7.1 Genéticos en MCP

Implementar un algoritmo genético para el problema del clique máximo se basa en considerar operadores de cruce y mutación apropiados, esto es, que mantengan la estructura de clique. A priori, no podemos afirmar que la combinación arbitraria de dos soluciones o la mutación produzcan cliques, por lo que estos operadores deben ser muy específicos o ser rectificados de alguna forma.

Una vez tengamos estos operadores, la ejecución del algoritmo consiste simplemente en ir obteniendo nuevas generaciones hasta un criterio de parada determinado, que, para el algoritmo implementado es un número fijo de iteraciones.

### 5.7.2 Algoritmo estacionario

El algoritmo genético implementado se basa en las ideas de S. Zhang, J. Wang, Q. Wu y J. Zhan [21], que usa un modelo estacionario, en el que los dos descendientes compiten con sus padres para entrar en la nueva generación. Al principio, se creará una población de soluciones de forma aleatoria mediante un algoritmo *greedy*, y a partir de aquí se irán generando nuevas soluciones.

El operador de cruce opera de la siguiente forma: dados dos padres, se toman aquellos nodos comunes en ambos, y estos estarán presentes en ambos hijos. Seguidamente, tomaremos el conjunto de nodos presentes en uno de los dos padres, y los distribuiremos entre los dos hijos de forma aleatoria y equiprobable.

Una vez tengamos los hijos, procedemos a mutarlos con la probabilidad dada. El algoritmo de mutación toma dos valores aleatorios entre 0 y el número de nodos del grafo, y cambia los nodos del intervalo que forman, quitándolos del hijo si pertenecen a él, e introduciéndolos si no lo hacen.

Como no tenemos garantizado que los hijos sean cliques antes o después de la mutación, los reduciremos a un clique eliminando nodos hasta que el conjunto restante sea un clique. Una vez reducido, lo ampliaremos hasta que el clique sea maximal. Ambos procesos de reducción y ampliación se hacen de forma aleatoria, si bien podríamos haber tomado cualquier otro método basado en la información del grafo, como ordenar por número de adyacencias.

La población inicial se ha tomado con un tamaño de 50 individuos, con probabilidad de cruce de 1 y probabilidad de mutación de 0,1, un valor alto para lo usual, fijado así para introducir diversidad en la población. El criterio de parada es un límite de iteraciones, que se ha fijado a 40 en las ejecuciones realizadas. Estos parámetros pueden ser modificados para evaluar la funcionalidad del algoritmo bajo otras circunstancias.

Vemos el pseudocódigo del algoritmo.



**Algorithm 13** Genético

---

```

function GENETICO(iteraciones)
     $P_{\text{mutación}} = 0,1$ .
     $P_{\text{cruce}} = 1$ .
    Crear población inicial aleatoria.
    Mejor clique = mejor clique de la población.
    repeat
        Nueva_población = [].
        repeat
            Extraer dos elementos aleatorios de la población.
            Cruzar los dos elementos con probabilidad  $P_{\text{cruce}}$ .
            Mutar primer hijo con probabilidad  $P_{\text{mutación}}$ .
            Mutar segundo hijo con probabilidad  $P_{\text{mutación}}$ .
            Reparar y completar primer hijo.
            Reparar y completar segundo hijo.
            Añadir los dos hijos a Nueva_población.
        until Población =  $\emptyset$ 
        Población = Nueva_población.
        Actualizar mejor clique.
    until i = iteraciones
    return mejor clique

```

---

## 5.8 ALGORITMOS MEMÉTICOS

Los algoritmos meméticos son el resultado de la combinación de algoritmos meméticos con búsqueda local. Tratan de aprovechar la buena capacidad de exploración de los algoritmos genéticos, y combinarla con la explotación de soluciones de las técnicas de búsqueda local.

El diseño de estos algoritmos queda muy abierto, y por lo general, suele ser necesario adaptarlos de una forma u otra dependiendo del problema. Suelen ser algoritmos específicos, que aprovechan alguna característica en concreto del problema.

Entre las diversas decisiones que hay que tomar, una de las más relevantes es el proceso de búsqueda local a considerar y cuándo aplicarlo. Podemos elegir cualquier técnica por trayectorias simples, desde una búsqueda local básica a una búsqueda tabú, y podemos aplicarlas a toda la población o a una parte, en distintas fases del algoritmo.

5.8.1 *Meméticos en MCP*

Debido a que disponemos de técnicas de búsqueda local y un algoritmo genético, la implementación de un algoritmo memético consiste en tomar las decisiones correspondientes con respecto a la hibridación de ambas técnicas.

### 5.8.2 *Algoritmo implementado*

En este trabajo, he optado por una combinación básica de dos algoritmos ya implementados, el algoritmo genético anterior y DLS como búsqueda local. De esta forma, se mantiene la estructura de algoritmo genético estacionario, introduciendo una búsqueda local al final de la creación de la nueva generación, lo que nos permite explorar su entorno antes de pasar a la siguiente.

El desarrollo del algoritmo es muy simple, pues utiliza como base el algoritmo genético introducido anteriormente. Utiliza el mismo operador de cruce y mutación, y añade una búsqueda local después de que los hijos sean reducidos y ampliados.

He utilizado DLS y no 1LS debido a su menor complejidad computacional, pues hibridar con 1LS producía tiempos de ejecución demasiado elevados.

Como parámetros, he tomado las mismas probabilidades de cruce y mutación, 1 y 0,1, y he establecido el límite de la búsqueda local en un octavo del número de nodos del grafo, para explorar únicamente la zona del espacio de soluciones más cercana a cada elemento de la población.

## IMPLEMENTACIÓN

---

En este capítulo vamos a tratar la implementación del programa encargado de aplicar los algoritmos al conjunto de instancias. Comenzaremos introduciendo el lenguaje de programación utilizado, siguiendo con una descripción de la funcionalidad completa del programa, y concluyendo con un desglose de carácter más técnico de las partes que lo componen.

### 6.1 LENGUAJE UTILIZADO

La implementación del código de este trabajo ha sido realizada en el lenguaje de programación **Ruby**, usando su versión 2.3.0. Ruby es un lenguaje orientado a objetos, interpretado, con una sintaxis muy similar a la de **Python**. Está disponible para descarga bajo una licencia de software libre.

He elegido este lenguaje por ser capaz de aportar simplicidad e intuitividad a la hora de programar los algoritmos. Como ya había trabajado con este lenguaje en ocasiones anteriores, conocía sus puntos fuertes, y he tratado de explotarlos lo máximo posible, para que el resultado sea un código legible y adaptable.

Al utilizar un lenguaje interpretado como es Ruby, cabe esperar unos tiempos de ejecución mayores que con otros lenguajes, debido a que no está hecho para ejecutar código lo más rápido posible. En el caso de querer realizar un programa capaz de ejecutar los algoritmos en el menor tiempo posible, lenguajes compilados como **C++** serían más útiles que Ruby.

### 6.2 FUNCIONALIDAD

El código implementado cumple varias funciones: la primera de ellas es llevar a memoria todas las instancias disponibles del problema. Para ello, he tenido que implementar un lector de ficheros que cree una matriz de adyacencia para cada archivo disponible. Estos archivos siguen una estructura determinada, que permite automatizar dicho proceso, no teniendo que considerar casos independientes. Cada archivo dará lugar a un problema, compuesto de una matriz de adyacencias, el número de nodos y aristas involucrados, y el nombre del grafo en cuestión.

El segundo punto, y más importante, es la implementación de las heurísticas. Cada una de ellas está contenida en una clase propia, que contendrá uno o varios métodos para la resolución de los problemas generados a partir de los archivos. Adicionalmente,

podrán contener métodos auxiliares para uso propio o de otras heurísticas, en caso de que utilicemos técnicas híbridas.

Finalmente, existe un conjunto de funciones auxiliares, que pueden ser requeridas por varias heurísticas. La idea bajo su creación es encapsular operaciones importantes, que serán repetidas frecuentemente y en diversos escenarios, evitando repetición de código y sirviendo de ayuda para facilitar el trabajo y para la comprensión del código.

### 6.3 INICIALIZACIÓN Y EJECUCIÓN DEL PROGRAMA

Antes de aplicar ningún algoritmo al conjunto de instancias, hay que resolver la lectura de los archivos y la creación de las estructuras de datos correspondientes para cada grafo. Estas operaciones se engloban en un método llamado *readerMethod*, que se encuentra en el archivo *reader.rb*.

Cada instancia del problema dará lugar a un objeto de la clase **Problem**, que almacena los datos de cada grafo: su nombre, número de vértices, número de aristas, su matriz de adyacencias y el orden de cada nodo, esto es, su número de adyacencias.

El método de lectura de los archivos recorre el fichero en el que están almacenadas las instancias del problema, y por cada uno de ellos construye un objeto de la clase **Problem**. Dado que los ficheros siguen todos el mismo formato, es posible extraer los datos de cada grafo sin más que hacer una distinción de casos.

La parte fundamental de la lectura, que es la creación de la matriz de adyacencias, se realiza partiendo de una matriz de ceros del tamaño adecuado, y tomando los pares que se proporcionan en el archivo, que representan los nodos que están conectados, para marcarlos como unos en la matriz de adyacencias. Dado que los nodos comienzan su numeración desde el 1 en los ficheros, y desde el 0 en el programa (debido a que las estructuras de datos comienzan en el 0), hemos de corregirlo a la hora de marcar las aristas en la matriz de adyacencias. Además, marcaremos todos los nodos como conectados con ellos mismos, pues será necesario para el correcto funcionamiento de algunas operaciones. Esto hace que todos los nodos de un clique pertenezcan a su conjunto  $C_0$  asociado, por lo que cuando lo calculemos tendremos que quitar los nodos del propio clique.

El número de adyacencias de cada nodo no lo proporciona el archivo del grafo, sino que hemos de calcularlo. Para ello, nos valemos de una función auxiliar programada con tal fin.

El archivo *reader.rb* también contendrá el método principal. En él, se ejecutarán el método de lectura y los algoritmos implementados que incluyamos, para lo que se debe añadir el código necesario. Además, se mostrarán por pantalla los resultados obtenidos, con la idea de poder redireccionarlos a un archivo de texto en el que almacenar dichos resultados.

## 6.4 ALGORITMOS

Todos los algoritmos implementados en el programa se encuentran recogidos en un mismo módulo, el módulo **Clique**, bajo el que también se encuentra el archivo *reader.rb* detallado con anterioridad. Cada tipo de algoritmo estará dentro de una clase, y se separarán los distintos algoritmos del mismo tipo en métodos diferentes.

Algunos algoritmos usan aleatoriedad en una o varias fases. Para conseguirla, se ha utilizado la clase **Random**, que proporciona un generador de números aleatorios. El método *rand* nos permite obtener un valor aleatorio dentro de un rango dado, pudiendo ser un número entero o real, lo que nos permite cubrir todas las necesidades en este aspecto.

### 6.4.1 Algoritmos greedy

Los dos algoritmos *greedy* implementados se encuentran en el archivo *greedy.rb*, dentro de la clase **Greedy**. El algoritmo *greedy* básico está en el método *solve*, mientras que el *greedy* adaptativo se encuentra en el método *solve2*. Ambos métodos reciben únicamente una instancia de la clase **Problem** como parámetro, el problema a resolver.

Además, en este archivo también se encuentran los distintos algoritmos voraces encargados de completar cliques o de reparar grafos, utilizados en otras heurísticas. Son los siguientes:

- *complete\_clique*, método que amplía un clique hasta que sea posible, tomando de entre los candidatos aquel con más adyacencias.
- *complete\_clique\_random*, que, al igual que el anterior, amplía un clique hasta que sea maximal, solo que en este caso elige de forma aleatoria.
- *solve\_random*, método que crea un clique partiendo desde cero, usando *complete\_clique\_random*. Lo usaremos para generar soluciones iniciales aleatorias.
- *repair*, método que elimina nodos de un grafo hasta que sea un clique. Elimina el nodo con menor número de adyacencias.
- *repair\_random*, con la misma función que el anterior, pero en este caso elimina un nodo aleatorio.

### 6.4.2 Búsqueda local

Los dos algoritmos de búsqueda local están en el archivo *localsearch.rb*, bajo la clase **LocalSearch**. Para la creación de la clase, solo es necesario crear un generador de números aleatorio, por lo que no hay que pasarle argumentos al constructor.

La heurística 1LS está implementada en el método *solve\_with\_solution*, y recibe como parámetro el elemento de la clase *Problem* a resolver, una solución de partida, y un límite de cambios.

Por otra parte, la heurística DLS se encuentra en el método *solve\_dynamic*. Los parámetros serán los mismos que en el caso anterior: la instancia de la clase *Problem* a resolver, la solución de partida y el límite de cambios.

Para la ejecución de cualquiera de los dos métodos partiendo desde el clique vacío, existe el método *solve*, que será el que llamemos cuando queramos ejecutar uno de los dos algoritmos. Recibe como parámetros el problema que resolver y el número de cambios, que los pasará al algoritmo que ejecute, además del clique vacío como solución de partida.

#### 6.4.3 Enfriamiento simulado

En el archivo *simulatedannealing.rb* se encuentran los dos algoritmos de enfriamiento simulado considerados, en la clase **SA**. El constructor de clase no necesita parámetros, pues crea un generador de números aleatorios y una instancia de la clase *Greedy*, ya que el algoritmo de Geng et al. necesita las funciones para reparar y completar grafos.

El algoritmo de enfriamiento simulado básico está implementado en el método *solve*, que recibe como único parámetro el problema a resolver. Para el criterio de aceptación del algoritmo Metrópolis, genera un número aleatorio entre 0 y 1, y si es menor que el valor de la probabilidad, acepta la solución. Para tomar elementos aleatorios del entorno, se utiliza la función *shuffle*, que baraja los elementos del entorno, y toma un número aleatorio como semilla.

El algoritmo de Geng et al. se encuentra en el método *solve2*, que vuelve a tener como único parámetro el problema que se quiere resolver. Efectúa las mismas operaciones para el criterio de Metrópolis y para el baraje del entorno.

Ambas funciones declaran los valores de las temperaturas y del parámetro  $\beta$  al inicio. Para cambiarlos, basta con modificar dichos valores en el código.

#### 6.4.4 GRASP

El algoritmo GRASP está implementado en el archivo *grasp.rb*, dentro de la clase **GRASP**. Además de un generador de números aleatorios, el constructor declara un objeto de la clase *LocalSearch*, que necesitaremos para aplicar el algoritmo de búsqueda local que consideremos.

El método *generate\_random\_solution* es el encargado de generar las soluciones iniciales pseudoaleatorias, tomando un problema como argumento. El tamaño de la lista restringida de candidatos está fijado por un valor en el código, que debemos cambiar en caso de querer aumentar o reducir el tamaño de dicha lista.

Como los dos algoritmos implementados solo difieren en la búsqueda local utilizada, es posible implementar ambos utilizando un único método, y cambiar la búsqueda local para alternar entre uno u otro. El método *solve* es el encargado de esto, recibiendo un problema y un número de iteraciones como parámetros. El número de cambios para el algoritmo de búsqueda local está declarado como una variable, que deberemos modificar si queremos variar su valor.

#### 6.4.5 ILS

El algoritmo ILS se encuentra en el archivo *ils.rb*, en la clase **ILS**. Su construcción no necesita parámetros, y se encarga de crear el generador de números aleatorios y un objeto de la clase *LocalSearch* para las búsquedas locales.

La clase tiene un único método, *solve*, que es el encargado de aplicar la búsqueda local y las permutaciones a las soluciones obtenidas. Recibe el problema a resolver y el número de iteraciones. El límite para la búsqueda local es una variable, que modificaremos de ser necesario. Al igual que en el caso de GRASP, para cambiar la búsqueda local que aplicamos basta con cambiar la llamada al método de búsqueda local.

#### 6.4.6 Algoritmos de colonia de hormigas

Los dos algoritmos de colonia de hormigas implementados están en el archivo *aco.rb*, dentro de la clase **ACO**. El constructor no recibe ningún parámetro, pues solo crea un generador de números aleatorios.

Mi versión del algoritmo está en el método *solve*, que recibe el problema que se quiere resolver y el número de iteraciones. El número de hormigas y el parámetro  $\beta$ , necesario para la evaporación de feromonas, se encuentran en el código, pudiendo cambiarse su valor de ser necesario.

Para elegir cada nodo al que va una hormiga, una vez calculadas las probabilidades, se genera un valor aleatorio entre 0 y 1, y vamos tomando nodos, acumulando su valor de probabilidad, hasta que dicho valor supere el valor aleatorio generado.

El algoritmo de Xu et al. se encuentra en el método *solve2*, y, debido a que su estructura es igual que el algoritmo anterior, salvo el cálculo de las probabilidades, se le puede aplicar todo lo mencionado anteriormente.

#### 6.4.7 Algoritmos genéticos

Tanto el algoritmo genético como el memético implementados se encuentran dentro de la clase **Genetics**, en el archivo *genetics.rb*. El constructor de la clase no recibe ningún parámetro, y crea un generador de números aleatorios, un elemento de la clase

LocalSearch y uno de la clase Greedy, que se usarán en el algoritmo memético y para completar o reparar las soluciones, respectivamente.

El algoritmo genético de Zhang et al. está implementado en el método *solve*, que recibe el problema a resolver y el número de iteraciones. Tanto las probabilidades de cruce y mutación como el tamaño de la población se encuentran en el código, donde deben ser modificadas de ser necesario. Como en ocasiones anteriores, para decidir eventos que dependen de una probabilidad, se genera un número entre 0 y 1, y si este es menor que la probabilidad, el suceso se lleva a cabo. Esto se utiliza en los mecanismos de cruce y de mutación, y además se usa también en el propio operador de cruce, donde los nodos que pertenecen solo a un padre van a cada uno de los hijos con probabilidad 0,5.

El algoritmo memético, implementado en *solve\_memetic*, funciona de la misma forma, aplicando búsqueda local al generar la nueva población, con un límite que también está en el código. Por lo demás, opera igual que el genético, por lo que son aplicables las consideraciones anteriores.

Además, existe un método privado para las mutaciones, llamado *mutation*. Recibe como parámetro la solución a mutar y la matriz de adyacencias, y devuelve la solución mutada.

## 6.5 MÉTODOS AUXILIARES

Además de los algoritmos, también se han implementado funciones que realizan cálculos necesarios en varios de los algoritmos, que han sido incluidas en un archivo adicional creado para almacenarlas. Estas funciones no se encuentran dentro de una clase, pero sí que están en un módulo distinto, el módulo **Algorithm**, para diferenciarlas de las implementaciones de las heurísticas. Todas estas funciones se encuentran en el archivo *algorithm.rb*, y las detallamos a continuación:

- *add*: Operador *add*. Recibe un clique y un vértice, y añade el vértice al clique. Supone que el vértice no está en el clique y que se mantiene la estructura.
- *swap*: Operador *swap*. Recibe un clique, un vértice  $v$  del conjunto  $C_1$ , y la matriz de adyacencias. Busca el nodo dentro del clique que no está conectado a  $v$  y lo intercambia con  $v$ . De nuevo, suponemos que  $v \in C_1$ , por lo que se puede hacer el intercambio.
- *swap\_two*: Operador *swap*. En este caso, se pasan como parámetros el clique y los dos nodos a intercambiar,  $p$  en  $C_1$  y  $q$  dentro del clique. El método elimina  $q$  del clique y mete  $p$ , suponiendo que se verifica lo anterior y que  $q$  es el único nodo del clique no conectado a  $p$ .
- *drop*: Operador *drop*. Recibe un clique y un nodo, y elimina dicho nodo del clique.
- *adjacencias*: Método que recibe un nodo y una matriz de adyacencias, y devuelve el número de adyacencias de dicho nodo. Se utiliza para calcular las adyacencias de todos los nodos al crear un problema.



- *connections*: Dados un nodo  $n$ , un conjunto de nodos  $S$  y una matriz de adyacencias, este método devuelve el número de nodos de  $S$  conectados con  $c$ .
- *is\_connected*: Recibe un nodo  $n$ , un conjunto de nodos  $S$  y una matriz de adyacencias, y devuelve si  $n$  está conectado con todos los nodos de  $S$ . Este método nos servirá para calcular el conjunto  $C_0$ .
- *is\_connected\_but\_one*: Funciona igual que el anterior, pero comprueba que el nodo  $n$  esté conectado con todos los nodos de  $S$  salvo uno. Nos servirá para el cálculo del conjunto  $C_1$ .
- *connected*: Dados un nodo y una matriz de adyacencias, devuelve todos los nodos del grafo conectados a él.
- *connected\_with\_all*: Dado un conjunto de nodos  $S$  y una matriz de adyacencias, usando el método *is\_connected*, devuelve el conjunto  $M$ , formado por los nodos conectados con todos los de  $S$ , y elimina el conjunto  $S$  de  $M$ , pues si  $S$  es un clique,  $S \subseteq M$ . Obtenemos por tanto el conjunto  $C_0$ , que será para lo que utilizemos este método.
- *missing\_one\_connection*: Dado un conjunto de nodos  $S$  y una matriz de adyacencias, devuelve el conjunto de nodos conectados con todos los de  $S$  salvo uno. Lo usaremos para calcular el conjunto  $C_1$  asociado a un clique.
- *one\_connected\_with\_all*: Recibe una lista y una matriz de adyacencias, y devuelve *true* si existe un nodo fuera de la lista conectado a todos los nodos de ella.
- *value* y *value2*: Funciones objetivo utilizadas en los algoritmos de enfriamiento simulado. Ambas reciben un grafo y una matriz de adyacencias, y devuelven el valor calculado.
- *is\_clique*: Función que recibe un grafo y una matriz de adyacencias, y devuelve si dicho grafo es un clique o no.
- *print\_solution*: Método para mostrar por pantalla una solución. Primero, comprueba si es un clique con el función anterior y lo muestra. Después, imprime su longitud. Además, imprime la solución y su longitud por la salida de error, lo que permite redirigirlo a un archivo distinto a la hora de ejecutar el programa.

## RESULTADOS Y ANÁLISIS

---

En este capítulo expondremos los resultados obtenidos por cada algoritmo sobre el conjunto de cliques que se ha considerado, y evaluaremos el funcionamiento individual de cada uno de ellos. Además, compararemos los algoritmos del mismo tipo para ver cómo afectan sus diferencias en los resultados.

### 7.1 CONSIDERACIONES

Para la toma de datos, he utilizado un portátil con el sistema operativo Antergos Linux 64-bit, con un procesador Intel Core i7-3610QM, que cuenta con 4 núcleos y 2 hebras por núcleo, con una frecuencia de reloj de 2,30 gigahercios. Los algoritmos han sido ejecutados individualmente, sin uso de paralelismo, por lo que cada ejecución ha ocupado una sola hebra.

Algunos de los algoritmos implementados en este trabajo cuentan con una o varias componentes aleatorias: elección de un elemento aleatorio, cálculo de un valor para aceptar o no una acción que depende de una probabilidad, etcétera. Debido a esto, para la obtención de resultados, se han realizado varias ejecuciones de estos algoritmos, de las cuales se ha tomado el mejor valor obtenido. He tomado el mejor, y no la media, porque representa el tamaño del clique que dicho algoritmo puede alcanzar en un número razonable de ejecuciones.

En el caso de los algoritmos que no tienen componentes aleatorias, como su ejecución lleva siempre a la misma solución, el programa se ha ejecutado una única vez en cada caso. Estos algoritmos son los dos *greedy* y 1LS. Además, los algoritmos ILS y GRASP que usan 1LS también se han ejecutado una única vez, aunque el método de obtención de soluciones iniciales tenga componentes aleatorias, debido a que su tiempo de ejecución es muy alto.

El resto de algoritmos incluyen componentes aleatorias, y se ha considerado un número variable de ejecuciones para cada uno de ellos, dependiendo del tiempo que tardasen. Se detallan a continuación:

- DLS: 20 iteraciones.
- Enfriamiento simulado: 10 iteraciones para cada algoritmo.
- Colonia de hormigas: 10 iteraciones para cada algoritmo.
- ILS y GRASP con DLS: 10 iteraciones.
- Genético: 10 iteraciones.

- Memético: 5 iteraciones.

## 7.2 ANÁLISIS

A continuación, vamos a ver los resultados obtenidos por cada algoritmo en forma de tabla, para hacer un análisis de cada uno de ellos. Debido a la cantidad de algoritmos, estos han sido divididos según su tipo, en grupos de dos: dos *greedy*, dos búsquedas locales, dos enfriamientos simulados, dos ILS, dos GRASP, dos algoritmos de colonia de hormigas y los algoritmos genético y memético. Para cada grupo, mostraremos los resultados y haremos un análisis individual y comparativo de los dos algoritmos que contiene.

Las tablas tienen la siguiente disposición: la primera columna contiene cada uno de los grafos considerados. La segunda columna indica el tamaño del mejor clique conocido para cada grafo, notando con un asterisco los valores que están confirmados como óptimos globales. El resto de columnas contienen el tamaño del mejor clique encontrado para cada algoritmo, notado como  $|C|$ , y el tiempo de ejecución, que se ha tomado el valor medio en caso de haberse efectuado más de una ejecución. Como medidas para evaluar a los grafos, he considerado el porcentaje medio de tamaño de los cliques encontrados, calculado como la media aritmética de los porcentajes en cada grafo, y el número de óptimos encontrados.

### 7.2.1 Algoritmos *greedy*

Grafo	Clique	Greedy básico		Greedy adaptativo	
		$ C $	Tiempo (s)	$ C $	Tiempo (s)
brock200_2	12	7	<0.01	9	0.18
brock200_4	17	13	0.02	15	0.27
brock400_2	29	20	0.08	22	1.61
brock400_4	33	18	0.07	21	1.58
brock800_2	24	14	0.18	19	4.98
brock800_4	26	14	0.18	19	4.41
C125.9	34*	29	0.02	32	0.44
C250.9	44*	35	0.08	40	1.49
C500.9	57	43	0.31	51	24.10
C1000.9	68	51	1.16	56	126.73
C2000.5	16*	10	0.72	13	24.73
C2000.9	80	56	4.39	66	1058.85
C4000.5	18*	12	2.86	15	3.64
DSJC500_5	13	8	0.05	11	0.07

Grafo	Clique	Greedy básico		Greedy adaptativo	
		C	Tiempo (s)	C	Tiempo (s)
DSJC1000_5	15	10	0.19	12	6.18
gen200_p0.9_44	44	32	0.06	36	1.81
gen200_p0.9_55	55	36	0.06	39	1.56
gen400_p0.9_55	55	44	0.22	43	12.67
gen400_p0.9_65	65	40	0.21	44	12.82
gen400_p0.9_75	75	45	0.23	46	13.36
hamming10-4	40	32	0.70	36	120.37
hamming8-4	16	16	0.02	16	0.57
keller4	11	8	<0.01	11	0.25
keller5	27	17	0.29	24	45.48
keller6	59	37	9.39	48	5575.49
MANN_a27	126	125	1.03	125	17.20
p_hat300-1	8	7	0.01	8	0.02
p_hat300-2	25	23	0.05	23	0.83
p_hat300-3	36	30	0.08	34	2.68
p_hat700-1	11	7	0.007	8	0.10
p_hat700-2	44	38	0.32	43	7.51
p_hat700-3	62*	55	0.59	58	10.20
p_hat1500-1	12	8	0.32	11	0.44
p_hat1500-2	65*	54	1.65	62	42.14
p_hat1500-3	94*	75	2.88	87	94.61

Cuadro 1: Resultados en algoritmos greedy.

Los resultados obtenidos por los dos algoritmos *greedy* han sido de un óptimo y una media de 73,2 % de tamaño para el *greedy* básico, y tres óptimos y un 84,9 % de media para el *greedy* adaptativo.

Los resultados del algoritmo *greedy* básico muestran que este falla a la hora de alcanzar el óptimo en el conjunto de instancias, pues lo logra en solo una de las 35. El tamaño medio manifiesta que no se obtienen soluciones de calidad, existiendo un amplio margen de mejora en la mayoría de instancias del problema. En cuanto a los tiempos, es un algoritmo rápido, que por lo general requiere menos de 1 segundo para ejecutarse, y que no supera los 10 segundos en ningún caso.

El *greedy* adaptativo logra mejorar o igualar los resultados obtenidos por el algoritmo anterior en todos los grafos, lo que confirma que ampliar el entorno con movimientos *swap* mejora sensiblemente los resultados, aunque aumenta de forma considerable los tiempos de ejecución, llegando a superar la hora en *keller6*. Aun así, en la mayoría de los grafos, la ejecución es rápida, no llegando a superar los 60 segundos. Si nos fijamos en sus resultados por separado, el porcentaje medio de tamaño de los cliques hallados y la

obtención de únicamente 3 óptimos, hacen que este algoritmo tampoco consiga buenos resultados si se aplica sin combinar con ningún otro.

Estos resultados muestran que los algoritmos voraces implementados no son capaces de alcanzar buenas soluciones por sí solos. Aun así, los cliques obtenidos pueden servir como soluciones de partida para otros algoritmos que sean capaces de mejorar las carencias que tienen los *greedy*. En este caso, si trabajamos con cliques de gran tamaño nos interesa un algoritmo más veloz como es el *greedy* básico. Si por el contrario los grafos son pequeños, a cambio de algo más de tiempo de ejecución, el algoritmo *greedy* adaptativo nos da soluciones de partida de más calidad.

### 7.2.2 Búsqueda local

Grafo	Clique	1LS		DLS	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	9	0.36	11	<0.01
brock200_4	17	15	0.37	15	0.01
brock400_2	29	22	2.05	23	0.06
brock400_4	33	23	1.85	23	0.07
brock800_2	24	19	6.82	19	0.10
brock800_4	26	20	6.13	18	0.09
C125.9	34*	33	0.36	34	0.04
C250.9	44*	42	1.91	41	0.14
C500.9	57	54	7.69	54	0.39
C1000.9	68	63	32.02	62	1.62
C2000.5	16*	13	46.60	14	0.19
C2000.9	80	72	179.76	68	3.63
C4000.5	18*	15	141.92	15	0.48
DSJC500_5	13	11	1.89	12	0.03
DSJC1000_5	15	13	8.28	13	0.05
gen200_p0.9_44	44	39	1.01	43	0.08
gen200_p0.9_55	55	39	1.01	55	0.12
gen400_p0.9_55	55	50	5.03	49	0.35
gen400_p0.9_65	65	45	9.12	49	0.37
gen400_p0.9_75	75	49	5.83	75	0.37
hamming10-4	40	36	19.48	40	0.68
hamming8-4	16	16	0.50	16	<0.01
keller4	11	11	0.22	11	0.01
keller5	27	23	17.27	27	0.22
keller6	59	51	556.38	50	4.97

Grafo	Clique	1LS		DLS	
		C	Tiempo (s)	C	Tiempo (s)
MANN_a27	126	125	62.29	123	3.84
p_hat300-1	8	7	0.55	7	<0.01
p_hat300-2	25	25	1.35	25	0.05
p_hat300-3	36	34	1.67	36	0.13
p_hat700-1	11	9	2.76	9	0.02
p_hat700-2	44	44	11.17	43	0.39
p_hat700-3	62*	60	28.40	61	0.76
p_hat1500-1	12	11	17.38	11	0.08
p_hat1500-2	65*	64	132.85	63	1.85
p_hat1500-3	94*	91	128.08	91	4.61

Cuadro 2: Resultados en búsqueda local.

Los dos algoritmos de búsqueda local han dado unos resultados de 4 óptimos y un 87,5 % de media para 1LS, y 9 óptimos y un 91,0 % de media para DLS.

Vemos como 1LS consigue mejorar los resultados obtenidos por ambos algoritmos *greedy* tanto en número de óptimos como de tamaño medio de los grafos, lo que reafirma que la exploración del entorno es importante a la hora de buscar soluciones. No obstante, en vista de los resultados, estos siguen siendo mejorables en cuanto a la calidad de las soluciones encontradas. Los tiempos de ejecución no son excesivos, si bien en los grafos más grandes pueden extenderse un poco, rozando los 10 minutos en *keller6*. Aun así, en la mayoría de los grafos *greedy* no se superan los 30 segundos.

DLS logra mejorar los resultados de 1LS, pues es capaz de alcanzar más óptimos y de obtener un mayor tamaño medio, que en este caso supera el 90 %, por lo que tenemos un algoritmo que empieza a obtener soluciones satisfactorias. Los tiempos de ejecución son significativamente menores, lo cual era de esperar, ya que DLS, al contrario que ILS, no hace cálculos para la elección de candidatos, sino que dicha elección es aleatoria, lo que reduce la complejidad del algoritmo. Todas las ejecuciones se completan en menos de 5 segundos, la mayoría de ellas requiriendo menos de 1 segundo.

Debido a la mejora en tiempos de ejecución y en resultados, podemos afirmar que DLS funciona mejor que 1LS al aplicarlo directamente sobre los grafos, y cabe esperar que también ofrezca mejores resultados cuando lo utilicemos en ILS y GRASP. Aun así, existen grafos, como *C2000.9* o *MANN\_a27* en los que 1LS proporciona resultados ligeramente mejores que DLS. Esto puede indicar que en grafos con una determinada estructura, 1LS pueda ser de más utilidad que DLS.

Recordando el funcionamiento de los algoritmos, 1LS ordenaba los tres movimientos para obtener el entorno y trataba de aprovechar información del grafo para elegir entre los nodos candidatos, mientras que DLS no consideraba movimientos *drop*, y las elecciones eran aleatorias. Los resultados parecen apuntar a que el entorno considerado en

DLS da mejores resultados que el tomado en 1LS. Además, el uso de información inherente al grafo no parece aportar beneficio, si bien sería necesaria más experimentación para poder afirmarlo.

### 7.2.3 Enfriamiento simulado

Grafo	Clique	SA básico		SA adaptado	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	10	1.25	9	10.00
brock200_4	17	16	2.15	13	9.67
brock400_2	29	24	7.73	18	24.15
brock400_4	33	24	5.55	19	23.30
brock800_2	24	19	11.19	16	54.90
brock800_4	26	20	11.13	16	43.35
C125.9	34*	34	2.06	32	6.31
C250.9	44*	44	8.78	36	14.42
C500.9	57	53	22.86	45	27.79
C1000.9	68	63	44.07	52	57.76
C2000.5	16*	15	29.26	12	136.12
C2000.9	80	71	83.85	57	133.82
C4000.5	18*	15	44.58	13	220.79
DSJC500_5	13	12	4.10	10	28.73
DSJC1000_5	15	14	9.28	11	60.67
gen200_p0.9_44	44	44	5.29	33	11.50
gen200_p0.9_55	55	55	6.18	37	10.82
gen400_p0.9_55	55	51	9.95	43	19.63
gen400_p0.9_65	65	65	21.35	44	24.56
gen400_p0.9_75	75	75	13.42	44	23.98
hamming10-4	40	40	52.06	30	61.17
hamming8-4	16	16	5.23	12	13.84
keller4	11	11	0.78	9	8.85
keller5	27	25	6.38	21	48.53
keller6	59	49	51.53	41	244.11
MANN_a27	126	121	12.68	122	17.65
p_hat300-1	8	8	1.22	7	18.32
p_hat300-2	25	25	4.16	21	15.20
p_hat300-3	36	36	7.03	32	16.92
p_hat700-1	11	10	3.65	8	39.18
p_hat700-2	44	44	22.02	37	41.78

Grafo	Clique	SA básico		SA adaptado	
		C	Tiempo (s)	C	Tiempo (s)
p_hat700-3	62*	62	28.50	52	41.75
p_hat1500-1	12	11	9.77	8	89.71
p_hat1500-2	65*	65	62.62	51	94.75
p_hat1500-3	94*	94	77.15	75	91.43

Cuadro 3: Resultados en enfriamiento simulado.

El algoritmo de enfriamiento simulado básico ha obtenido unos resultados de 16 óptimos y un 93,5 % de media en el tamaño de los cliques, con respecto al máximo conocido, mientras que la adaptación del algoritmo de Geng et al. ha obtenido 0 óptimos y un 75,7 % de tamaño medio.

Los resultados en el algoritmo básico mejoran los obtenidos por ambos algoritmos de búsqueda local, convirtiéndolo en una buena herramienta para resolver el problema, pues supera ampliamente el 90 % de tamaño medio, y alcanza el óptimo casi en la mitad de instancias. Recordemos que toma un entorno distinto al considerado en la búsqueda local, sin priorizar ningún movimiento sobre otro. Esto, y el uso de técnicas de enfriamiento simulado podrían ser los causantes de que el algoritmo mejore las búsquedas locales. Además, presenta unos tiempos de ejecución bajos, sin llegar a los 90 segundos en ningún caso. Esto se debe a que el número de iteraciones no depende del grafo, sino que viene dado por la temperatura y es fijo en todos ellos.

Por otra parte, la adaptación del algoritmo de Geng et al. presenta unos resultados muy pobres, sin ser capaz de encontrar en óptimo en ningún caso. Podemos afirmar que falla a la hora de resolver el problema de forma satisfactoria. Dado que la exploración del entorno es aleatoria, como en el caso anterior, cabe plantearse si considerar grafos cualesquiera en lugar de cliques como espacio de soluciones es la causa de este mal rendimiento. La función objetivo, con componentes aparte del tamaño de los grafos, también podría ser causante de que el algoritmo no se dirija a las zonas del espacio de soluciones de mayor calidad. Además, la función de reducción de grafos a cliques es otro factor a tener en cuenta, pues no tenemos garantizado que la reducción sea óptima, y añade otra etapa de posible pérdida de calidad en las soluciones.

Con estos resultados qued comprobado que el algoritmo básico funciona mucho mejor que la adaptación del algoritmo de Geng et al.

#### 7.2.4 ILS



Grafo	Clique	ILS - 1LS		ILS - DLS	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	10	4.36	10	0.12
brock200_4	17	15	6.69	16	0.21
brock400_2	29	24	34.90	24	0.89
brock400_4	33	24	35.40	24	0.90
brock800_2	24	18	116.12	19	1.72
brock800_4	26	18	116.78	20	1.53
C125.9	34*	34	6.91	34	0.55
C250.9	44*	44	25.94	44	1.36
C500.9	57	57	124.56	55	3.72
C1000.9	68	61	514.84	63	10.28
C2000.5	16*	12	547.22	14	2.59
C2000.9	80	72	2512.71	72	29.10
C4000.5	18*	16	2489.99	15	6.50
DSJC500_5	13	12	31.75	12	0.43
DSJC1000_5	15	14	145.22	14	1.14
gen200_p0.9_44	44	39	14.99	40	0.98
gen200_p0.9_55	55	55	17.33	55	0.99
gen400_p0.9_55	55	51	82.47	51	2.43
gen400_p0.9_65	65	49	71.32	65	2.48
gen400_p0.9_75	75	75	134.14	75	2.98
hamming10-4	40	38	315.82	40	3.94
hamming8-4	16	16	8.27	16	0.13
keller4	11	10	4.40	11	0.17
keller5	27	23	180.97	25	2.91
keller6	59	48	10034.52	54	58.89
MANN_a27	126	126	1106.07	125	55.67
p_hat300-1	8	7	8.36	8	0.12
p_hat300-2	25	25	21.33	25	0.83
p_hat300-3	36	36	26.64	36	1.24
p_hat700-1	11	9	53.69	10	0.47
p_hat700-2	44	44	179.95	44	4.36
p_hat700-3	62*	61	282.32	62	6.67
p_hat1500-1	12	10	255.15	10	1.24
p_hat1500-2	65*	65	1417.50	65	22.65
p_hat1500-3	94*	93	2167.37	94	35.91

Cuadro 4: Resultados en ILS.

El algoritmo ILS que utiliza 1LS, ha obtenido unos resultados de 11 óptimos y un 90,3 % de tamaño medio, mientras que el que utiliza DLS ha obtenido 15 óptimos y un 93,3 % de tamaño medio.

Los resultados en ambos algoritmos mejoran sensiblemente los alcanzados en las respectivas búsquedas locales, lo que nos dice que el uso de técnicas de reinicialización es efectivo a la hora de salir de óptimos globales, obteniendo soluciones de partida que nos permiten explorar de forma más efectiva el espacio de soluciones y alcanzar mejores cliques.

ILS + 1LS es capaz de obtener buenas soluciones, estando ligeramente por encima del 90 % de tamaño medio en cliques. Sus tiempos de ejecución empiezan a ser notablemente altos, con valores que llegan hasta casi 3 horas en *keller6*, debido a la repetición de 1LS en 20 ocasiones. Para tratar de que este algoritmo pueda competir con el resto, habría que experimentar si se pueden mantener los resultados reduciendo el número de reinicializaciones, para así disminuir los tiempos de ejecución.

ILS + DLS es mejor que la versión con 1LS tanto en resultados como en tiempos de ejecución, resultado que era de esperar, vistos los obtenidos en la búsqueda local. En este caso la ejecución no toma más de 60 segundos en ninguno de los grafos, mejorando sensiblemente los tiempos del algoritmo anterior.

Concluimos que ambos algoritmos son capaces de obtener soluciones de calidad, siendo mejor la versión con DLS por sus mejores resultados y menor tiempo de ejecución, resultado que es consistente con los obtenidos para DLS e ILS.

#### 7.2.5 GRASP

Grafo	Clique	GRASP - 1LS		GRASP - DLS	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	11	4.38	11	0.28
brock200_4	17	16	6.79	16	0.47
brock400_2	29	24	33.76	23	2.21
brock400_4	33	25	32.25	33	2.23
brock800_2	24	19	116.77	19	4.78
brock800_4	26	19	101.59	20	4.60
C125.9	34*	34	5.90	34	0.99
C250.9	44*	44	25.92	43	3.38
C500.9	57	53	114.25	53	12.17
C1000.9	68	63	575.84	63	36.31
C2000.5	16*	15	598.26	15	16.16
C2000.9	80	71	2811.73	69	134.39
C4000.5	18*	16	2561.37	16	58.44

Grafo	Clique	GRASP - 1LS		GRASP - DLS	
		C	Tiempo (s)	C	Tiempo (s)
DSJC500_5	13	13	31.00	13	1.26
DSJC1000_5	15	14	150.85	14	4.44
gen200_p0.9_44	44	44	17.04	44	2.24
gen200_p0.9_55	55	55	16.21	55	2.57
gen400_p0.9_55	55	51	78.42	49	8.42
gen400_p0.9_65	65	52	76.15	65	8.72
gen400_p0.9_75	75	75	94.38	75	9.98
hamming10-4	40	40	340.81	40	19.86
hamming8-4	16	16	8.46	16	0.47
keller4	11	11	4.47	11	0.33
keller5	27	26	200.16	27	7.50
keller6	59	54	8958.52	53	181.52
MANN_a27	126	125	1453.48	125	79.18
p_hat300-1	8	8	9.00	8	0.32
p_hat300-2	25	25	21.48	25	1.46
p_hat300-3	36	36	31.68	36	2.70
p_hat700-1	11	9	54.35	11	1.51
p_hat700-2	44	44	215.50	44	9.21
p_hat700-3	62*	62	303.31	62	18.55
p_hat1500-1	12	10	266.69	11	5.94
p_hat1500-2	65*	65	1600.58	65	48.06
p_hat1500-3	94*	94	2297.67	93	103.41

Cuadro 5: Resultados en GRASP.

Los resultados obtenidos por ambos algoritmos GRASP han sido de 16 óptimos y un 93,5 % de tamaño medio para la versión con 1LS, y 18 óptimos y un 95,3 % para la versión con DLS.

Estos algoritmos son capaces de alcanzar el óptimo en aproximadamente un 50 % de los casos de estudio, reflejando un buen comportamiento general. Además, la versión que usa DLS como búsqueda local supera el 95 % de tamaño medio en los grafos, por lo que estamos ante un algoritmo capaz de dar soluciones cercanas al óptimo. El algoritmo con 1LS también ofrece resultados de calidad, incluso mejorando los resultados obtenidos por el otro GRASP en grafos como *C2000.9* o *p\_hat1500-3*.

Nuevamente, el uso de DLS como búsqueda local proporciona mejores resultados que si utilizamos la primera búsqueda local, lo que nos reafirma en su mejor comportamiento, tanto por separado como si se utiliza en técnicas multiarranque. Los tiempos de ejecución también favorecen a la versión con DLS, sobre todo en grafos con un mayor número de nodos, donde la mayor complejidad computacional de 1LS afecta notable-

mente a dichos tiempos. Ejemplos de esto son grafos como *keller6*, *C4000.5* y *C2000.9*, en los que la versión con DLS tiene tiempos del orden de los minutos, y la versión con 1LS tiene tiempos del orden de horas.

Los dos algoritmos proporcionan mejores resultados que sus correspondientes versiones de búsqueda local iterada, lo que en un principio nos dice que el uso de soluciones pseudoaleatorias como solución de partida proporciona mejores resultados que el proceso de mutación utilizado en ILS. No obstante, dado que estos procesos pueden ser modificados, queda aun margen de mejora en ambos algoritmos, por lo que sería pronto para decir que GRASP funciona mejor que ILS en el problema del clique máximo. Sería necesaria más experimentación para poder sacar conclusiones a la hora de comparar ambas técnicas.

### 7.2.6 Algoritmos de colonia de hormigas

Grafo	Clique	ACO básico		ACO + SA	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	12	2.24	12	2.56
brock200_4	17	16	4.60	16	4.96
brock400_2	29	22	17.35	22	18.92
brock400_4	33	23	17.45	22	18.99
brock800_2	24	19	25.39	19	27.56
brock800_4	26	18	25.19	19	27.41
C125.9	34*	34	17.12	34	14.18
C250.9	44*	44	45.33	42	38.78
C500.9	57	53	111.15	50	104.84
C1000.9	68	55	264.30	54	268.65
C2000.5	16*	14	42.72	14	46.29
C2000.9	80	60	724.77	60	748.91
C4000.5	18*	15	98.38	16	116.37
DSJC500_5	13	13	7.32	12	8.20
DSJC1000_5	15	13	17.80	13	19.62
gen200_p0.9_44	44	41	32.81	39	27.14
gen200_p0.9_55	55	55	53.89	55	34.24
gen400_p0.9_55	55	49	89.14	46	77.33
gen400_p0.9_65	65	57	93.20	48	79.71
gen400_p0.9_75	75	75	112.08	65	85.24
hamming10-4	40	33	68.76	32	72.97
hamming8-4	16	16	3.52	16	4.03
keller4	11	11	2.27	11	2.54

Grafo	Clique	ACO básico		ACO + SA	
		C	Tiempo (s)	C	Tiempo (s)
keller5	27	24	34.34	24	37.73
keller6	59	48	618.39	45	630.96
MANN_a27	126	125	665.68	124	635.35
p_hat300-1	8	8	1.88	8	2.19
p_hat300-2	25	25	14.57	25	13.66
p_hat300-3	36	36	32.46	35	28.17
p_hat700-1	11	9	5.52	10	6.35
p_hat700-2	44	44	70.94	43	64.89
p_hat700-3	62*	61	182.29	61	150.63
p_hat1500-1	12	10	15.65	10	28.12
p_hat1500-2	65*	44	195.39	54	203.79
p_hat1500-3	94*	87	567.45	82	524.60

Cuadro 6: Resultados en algoritmos de colonia de hormigas.

Los algoritmos de colonia de hormigas han obtenido unos resultados de 12 óptimos y un 89,7 % de tamaño medio la versión básica, y 7 óptimos, con un 88,5 % de tamaño medio la versión con enfriamiento simulado.

Ambos algoritmos ofrecen resultados similares en cuanto al tamaño medio de los cliques encontrados, si bien la versión que añade información del grafo con enfriamiento simulado es ligeramente peor, encontrando menos óptimos y con un porcentaje menor de tamaño medio. Estos resultados indican que son necesarios cambios y más experimentación para ver si añadir dicho término a la probabilidad es capaz de aportar mejores nodos a la construcción de soluciones.

Los resultados del algoritmo básico son buenos, rozando el 90 % de tamaño medio, aunque queda por detrás de otras técnicas, como las búsquedas multiarranque o el enfriamiento simulado básico. Para tratar de mejorar su rendimiento, podríamos considerar añadir técnicas para construir un algoritmo híbrido, como una búsqueda basada en entornos, aprovechando mejor el comportamiento de las hormigas e intentando que estas construyan mejores soluciones.

En el algoritmo con enfriamiento simulado, si comparamos con el anterior, se alcanzan soluciones de menor o igual tamaño en 31 de los 35 casos, por lo que el término añadido al cálculo de la probabilidad no parece haber tenido repercusión positiva en la obtención de soluciones. Habría que plantearse otra forma de cambiar la función de probabilidad para que se use la información del grafo de forma positiva.

Ambos algoritmos son similares en cuanto a tiempos, ya que la única diferencia es el cálculo añadido en la probabilidad en la versión con enfriamiento simulado, que no parece repercutir de forma significativa. En general, los tiempos no son muy extensos en todo el conjunto de grafos, aunque alcanzan valores un tanto altos en las instancias

con más nodos, siendo C2000.9 la que más tiempo requiere, unos 12 minutos para cada uno de los dos algoritmos.

### 7.2.7 Algoritmos genéticos

Grafo	Clique	Genético		Memético	
		C	Tiempo (s)	C	Tiempo (s)
brock200_2	12	12	2.10	12	6.74
brock200_4	17	17	4.02	17	13.11
brock400_2	29	23	16.45	25	50.66
brock400_4	33	25	16.44	33	51.26
brock800_2	24	20	24.77	20	72.04
brock800_4	26	19	24.16	20	77.78
C125.9	34*	34	12.48	34	42.86
C250.9	44*	42	33.91	44	143.46
C500.9	57	48	94.39	56	284.240
C1000.9	68	54	261.16	65	765.52
C2000.5	16*	15	44.22	16	123.12
C2000.9	80	60	737.43	73	2128.81
C4000.5	18*	12	115.13	17	297.26
DSJC500_5	13	13	7.14	13	20.25
DSJC1000_5	15	13	17.62	15	51.49
gen200_p0.9_44	44	40	24.00	44	78.15
gen200_p0.9_55	55	55	24.99	55	54.66
gen400_p0.9_55	55	46	68.00	55	233.02
gen400_p0.9_65	65	47	69.90	65	216.87
gen400_p0.9_75	75	69	77.20	75	199.47
hamming10-4	40	34	75.71	40	205.56
hamming8-4	16	16	3.58	16	6.14
keller4	11	11	1.95	11	10.73
keller5	27	27	33.26	27	192.61
keller6	59	47	634.94	53	4265.78
MANN_a27	126	125	646.76	125	1682.67
p_hat300-1	8	8	1.92	8	6.20
p_hat300-2	25	25	11.45	25	50.88
p_hat300-3	36	36	25.37	36	85.84
p_hat700-1	11	11	6.05	11	18.93
p_hat700-2	44	44	81.02	44	270.53
p_hat700-3	62*	60	149.41	62	565.92

Grafo	Clique	Genético		Memético	
		C	Tiempo (s)	C	Tiempo (s)
p_hat1500-1	12	11	18.45	11	56.23
p_hat1500-2	65*	61	319.06	65	1472.25
p_hat1500-3	94*	86	615.29	94	2964.62

Cuadro 7: Resultados en algoritmos genéticos.

El algoritmo genético ha dado unos resultados de 13 óptimos encontrados y un 90,5 % de tamaño medio de los cliques, mientras que su versión híbrida con la búsqueda local DLS ha encontrado 25 óptimos, con un tamaño medio de un 97,3 %, lo que lo convierte en el mejor de los algoritmos implementados en cuanto a resultados. Queda claro que usar una búsqueda local para mejorar la población del algoritmo genético ayuda a obtener mejores soluciones.

Por sí solo, el algoritmo genético presenta un buen comportamiento, siendo capaz de encontrar soluciones aceptables en el conjunto de instancias considerado. Habría que experimentar con distintos parámetros tanto en probabilidades de cruce y mutación como en número de iteraciones y tamaño de población para poder realizar un ajuste más minucioso, que conllevara una mejora de los resultados. También debería estudiarse la necesidad de reinicializar algunos individuos de la población, en caso de que estos fueran similares unos a otros, para introducir diversidad.

El algoritmo memético fruto de la combinación del genético y DLS logra mejorar considerablemente el funcionamiento del genético, proporcionando muy buenos resultados en el conjunto de instancias. El 97,3 % de tamaño medio nos indica que estamos ante un algoritmo capaz de encontrar soluciones cercanas al óptimo en todas las instancias, pues si nos fijamos en los grafos por separado, vemos que no hay ninguno en el que el rendimiento sea especialmente malo.

Dado que la única diferencia que existe entre las dos versiones es el uso de DLS, se pone de manifiesto que es capaz de mejorar buenos algoritmos, lo que nos lleva a considerar su uso para otros algoritmos, como podría ser un ACO, el cual daba unos resultados similares al genético.

### 7.2.8 Comportamiento general

El conjunto de algoritmos implementados muestra, en general, un comportamiento dentro de lo esperado. Hemos visto como los algoritmos *greedy* no son capaces de encontrar por sí solos buenas soluciones al problema, aunque aportan cliques que pueden ser utilizados por otros métodos como soluciones iniciales. La exploración del entorno hecha con búsquedas locales o enfriamiento simulado mejora estos resultados, si bien no todos los algoritmos de este tipo proporcionan buenas soluciones, como demuestra la adaptación de la idea de Geng et al. Estas soluciones pueden ser aun mejores si se utili-

zan técnicas multiarranque, hecho que respaldan los resultados de los algoritmos ILS y GRASP.

Dado su amplio uso en problemas de grafos, y sus buenos resultados en general, cabía esperar que los algoritmos de colonia de hormigas funcionasen bien en este problema, y así ha sido, aunque sus resultados no han destacado especialmente. Por otra parte, algoritmos menos usuales en problemas de grafos como los genéticos, han mostrado un comportamiento bueno, en especial el algoritmo memético, claramente el mejor en cuanto a resultados de todos los considerados. Ajustar en ambos tipos de algoritmos los parámetros involucrados podría mejorar su comportamiento, sobre todo en el caso de los ACO, donde los resultados han sido algo peores.

Para realizar una comparativa entre todos los algoritmos, vamos a clasificarlos en todos los grafos que se han considerado. Tomaremos los tres algoritmos que den mejores resultados en cuanto al tamaño del clique encontrado, y usaremos el tiempo para decidir en caso de empate. De esta forma, otorgaremos primer, segundo y tercer puesto para cada grafo, y, haciendo un recuento para todos los algoritmos, veremos como funcionan con respecto a los demas.

Grafo	1 <sup>er</sup> puesto	2 <sup>o</sup> puesto	3 <sup>er</sup> puesto
brock200_2	Genético	ACO básico	ACO + SA
brock200_4	Genético	Memético	ILS + DLS
brock400_2	Memético	ILS + DLS	SA básico
brock400_4	GRASP + DLS	Memético	Genético
brock800_2	Genético	Memético	DLS
brock800_4	ILS + DLS	GRASP + DLS	1LS
C125.9	DLS	ILS + DLS	GRASP + DLS
C250.9	ILS + DLS	SA básico	GRASP + 1LS
C500.9	ILS + 1LS	Memético	ILS + DLS
C1000.9	Memético	ILS + DLS	1LS
C2000.5	Memético	GRASP + DLS	SA básico
C2000.9	Memético	ILS + DLS	1LS
C4000.5	Memético	GRASP + DLS	ACO + SA
DSJC500_5	GRASP + DLS	Genético	ACO básico
DSJC1000_5	Memético	ILS + DLS	GRASP + DLS
gen200_p0.9_44	GRASP + DLS	SA básico	GRASP + 1LS
gen200_p0.9_55	DLS	ILS + DLS	GRASP + DLS
gen400_p0.9_55	Memético	ILS + DLS	SA básico
gen400_p0.9_65	ILS + DLS	GRASP + DLS	SA básico
gen400_p0.9_75	DLS	ILS + DLS	GRASP + DLS
hamming10-4	DLS	ILS + DLS	GRASP + DLS
hamming8-4	DLS	Greedy básico	ILS + DLS
keller4	DLS	ILS + DLS	1LS



keller5	DLS	GRASP + DLS	Genético
keller6	ILS + DLS	GRASP + 1LS	GRASP + DLS
MANN_a27	ILS + 1LS	Greedy básico	DLS
p_hat300-1	DLS	Greedy adaptativo	ILS + DLS
p_hat300-2	DLS	ILS + DLS	1LS
p_hat300-3	DLS	ILS + DLS	GRASP + DLS
p_hat700-1	GRASP + DLS	Genético	Memético
p_hat700-2	ILS + DLS	GRASP + DLS	1LS
p_hat700-3	ILS + DLS	GRASP + DLS	SA básico
p_hat1500-1	DLS	Greedy adaptativo	GRASP + DLS
p_hat1500-2	ILS + DLS	GRASP + DLS	SA básico
p_hat1500-3	ILS + DLS	SA básico	GRASP + 1LS

Cuadro 8: Clasificación por grafos.

Si hacemos un recuento de las posiciones obtenidas por los grafos, obtenemos los siguientes resultados:

Algoritmo	1 <sup>er</sup> puesto	2 <sup>o</sup> puesto	3 <sup>er</sup> puesto
Greedy básico	0	2	0
Greedy adaptativo	0	2	0
1LS	0	0	6
DLS	11	0	2
SA básico	0	3	6
SA adaptado	0	0	0
ILS + 1LS	2	0	0
ILS + DLS	8	12	4
GRASP + 1LS	0	1	3
GRASP + DLS	4	8	8
ACO básico	0	1	1
ACO + SA	0	0	2
Genético	3	2	2
Memético	7	4	1

Cuadro 9: Recuento.

Vemos como los primeros puestos los ocupan los algoritmos DLS, las versiones de ILS y GRASP que usan DLS, la versión ILS de 1LS y los algoritmos genético y memético, mientras que el resto de posiciones están más distribuidas entre todos los algoritmos. El único algoritmo que no logra estar entre los tres mejores en ningún grafo es la

adaptación del enfriamiento simulado de Geng et al., lo que nos reafirma en su mal comportamiento.

En grafos más simples, donde el óptimo es alcanzado por varios algoritmos, los algoritmos más veloces son los que ocupan los primeros puestos. Así, DLS es capaz de alcanzar 11 primeros puestos, gracias a ser el algoritmo más rápido de entre los considerados, haciendo que siempre que alcance el óptimo se coloque en cabeza. Las versiones multiarranque de DLS también logran una gran presencia entre los tres mejores, ya que son capaces de mejorar el comportamiento de DLS en los grafos donde este no alcanza el óptimo, sin aumentar su tiempo de ejecución tanto como otros algoritmos.

Es en los grafos complejos donde algoritmos como el genético o el memético logran diferenciarse del resto, pues, si bien son más lentos que los mencionados anteriormente, son capaces de alcanzar mejores soluciones. De esta forma, los algoritmos genético y memético pueden estar entre los tres primeros, y usualmente en primer lugar, sobre todo el algoritmo memético, cuyos tiempos elevados le han hecho perder en la comparativa con búsquedas locales en caso de empate.

Por tanto, con esta comparación, podemos decir que las búsquedas locales basadas en DLS y los algoritmos genéticos son los mejores algoritmos de todos los considerados. Los primeros destacan por su velocidad y buena capacidad de resolución de un gran número de instancias. Los segundos tratan de profundizar más en el espacio de soluciones a costa de mayores tiempos de ejecución, lo que les permite ganar calidad en los grafos donde algoritmos más simples no trabajan tan bien. Otros algoritmos, como el enfriamiento simulado simple, o las búsquedas basadas en 1LS, también dan buenos resultados en ciertas instancias, dando a entender que son algoritmos competitivos en ciertos escenarios.

### 7.3 CONCLUSIONES

Hemos estudiado de forma satisfactoria el funcionamiento de todos los algoritmos implementados, que pueden verse como un primer acercamiento al problema que nos permite evaluar las prestaciones de distintas formas de abordarlo. El trabajo ha sido provechoso en cuanto a resultados obtenidos, y puede utilizarse como base para futuros trabajos en esta materia.

La línea de trabajo a seguir a partir de aquí podría ser una de las siguientes:

- Enfatización en un tipo de algoritmos, experimentando con diversos cambios que mejoren su funcionamiento. Dependiendo del tipo de algoritmo, podrían considerarse estrategias de reinicialización, énfasis en ciertas fases de búsqueda de soluciones, o búsqueda de nuevas medidas que clasifiquen mejor a las soluciones obtenidas, entre muchas otras. De esta forma, se intentaría obtener un algoritmo más específico para resolver el problema, con un mejor funcionamiento.
- Ampliación del trabajo realizado. Existen numerosas técnicas que no se han implementado en este trabajo, como la búsqueda tabú, la búsqueda en entornos variables, la búsqueda dispersa, etc. Analizar su comportamiento nos daría más

información sobre el problema, aportando nuevas ideas para tratar de mejorar estos u otros algoritmos.

## BIBLIOGRAFÍA

---

- [1] M. Dorigo, V. Maniezzo y A. Coloni. «Ant System: Optimization by a colony of cooperating agents». En: *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 26 (1996), págs. 29-41.
- [2] T.A. Feo y M.G.C. Resende. «A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem». En: *Operations Research Letters* 8 (1989), págs. 67-71.
- [3] T.A. Feo y M.G.C. Resende. «Greedy Randomized Adaptive Search Procedures». En: *Journal of Global Optimization* 6 (1995), págs. 109-134.
- [4] M. Garey y D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979.
- [5] X. Geng, J. Xu, J. Xiao y L. Pan. «A Simple Simulated Annealing Algorithm for the Maximum Clique Problem». En: *Information Sciences* 177 (2007), págs. 5064-5071.
- [6] F. Glover. «Future Paths for Integer Programming and links to Artificial Intelligence». En: *Computers and Operations Research* 5 (1986), págs. 533-549.
- [7] A. Grosso, M. Locatelli y F. Della Croce. «Combining Swaps and Node Weights in an Adaptive Greedy Approach for the Maximum Clique Problem». En: *Journal of Heuristics* 10 (2004), págs. 135-152.
- [8] A. Grosso, M. Locatelli y W. Pullan. «Simple ingredients leading to very efficient heuristics for the maximum clique problem». En: *Journal of Heuristics* 14 (2008), págs. 587-612.
- [9] J.P. Hart y A.W. Shogan. «Semi-greedy Heuristics: An Empirical Study». En: *Operations Research Letters* 6 (1987), págs. 107-114.
- [10] J. Hastad. «Clique is hard to approximate within  $n^{1-\epsilon}$ ». En: *Acta Mathematica* 182 (1999), págs. 105-142.
- [11] Sze-Tsen Hu. *Homology Theory: A First Course in Algebraic Topology*. Holden-Day, 1966.
- [12] K. Katayama, A. Hamamoto y H. Narihisa. «An Effective Local Search for the Maximum Clique Problem». En: *Information Processing Letters* 95 (2005), págs. 503-511.
- [13] S. Kirkpatrick, D. Gelatt y M. P. Vecchi. «Optimization by simulated annealing». En: *Science* 220 (1983), págs. 671-680.
- [14] W. S. Massey. *Introducción a la topología algebraica*. Reverté, 1972.
- [15] B. Melián, J. A. Moreno y J. M. Moreno. «Metaheurísticas: Una Visión Global». En: *Revista Iberoamericana de Inteligencia Artificial* 19 (2003), págs. 7-28.
- [16] W. Pullan y Holger H. Hoos. «Dynamic Local Search for Maximum Clique Problem». En: *Journal of Artificial Intelligence Research* 25 (2006), págs. 159-185.

- [17] T. Stützle. «Local Search Algorithms for Combinatorial Problems». Tesis doct. 1998.
- [18] James W. Vick. *Homology Theory: An Introduction to Algebraic Topology*. Academic Press, 1973.
- [19] X. Xinshun, M. Jun y L. Jingsheng. «An Improved Ant Colony Optimization for the Maximum Clique Problem». En: *Third International Conference on Natural Computation* (2007), págs. 135-152.
- [20] N. Xiong, D. Molina, M.L. Ortiz y F. Herrera. «A Walk into Metaheuristics for Engineering Optimization: Principles, Methods and Recent Trends». En: *International Journal of Computational Intelligence Systems* 8.4 (2014), págs. 606-636.
- [21] S. Zhang, J. Wang, Q. Wu y J. Zhan. «A Fast Genetic Algorithm for Solving the Maximum Clique Problem». En: *10th International Conference on Natural Computation*. IEEE. 2014, págs. 764-768.