Topología Algebraica Efectiva

por

Julio Rubio

Si alguien se aproxima a la Topología Algebraica a través de los libros recientes dedicados a esta disciplina, o siguiendo los contenidos de las materias que bajo dicho nombre se imparten en las licenciaturas de Matemáticas, podría extrañarse al ver relacionada esta abstracta rama de las Matemáticas con la programación de computadores que, en principio, parece ocuparse de la manipulación de objetos mucho más concretos y cercanos. Y sin embargo, desde los inicios mismos de la Topología Algebraica la necesidad de realizar cálculos explícitos ha sido esencial en el desarrollo de la misma. La naturaleza de los espacios topológicos es demasiado abstracta para permitir, de un modo cómodo, no sólo la clasificación de los mismos, sino incluso la demostración de que dos de tales espacios no son esencialmente iguales, esto es, no son homeomorfos. De ahí nació el concepto de *invariante topológico*. Si asociamos a un espacio topológico X un objeto algebraico H de modo que cualquier otro espacio topológico homeomorfo a X tenga asociado, por el mismo procedimiento, un objeto algebraico isomorfo a H, es decir, si H es un invariante de X, tendremos la puerta abierta para detectar cuándo dos espacios topológicos no son homeomorfos: bastará comprobar que los invariantes asociados al uno y al otro no son isomorfos. (En el anterior argumento subyace la filosofía de que los objetos algebraicos serán más fáciles de identificar, clasificar y distinguir que los topológicos; o más generalmente, que el Álgebra es más fácil que la Geometría.)

No es por tanto sorprendente que los primeros esfuerzos en Topología Algebraica se concentrasen en el *cálculo* de invariantes para espacios concretos. Y del estudio sistemático de estos procesos de cálculo se llegó de modo natural a la descripción *algorítmica* de dichos procedimientos. Así en 1931, en uno de los primeros textos [7] dedicados en exclusiva a la Topología (denominada todavía con su nombre primitivo: *Analysis Situs*), O. Veblen recoge un algoritmo de cálculo de uno de los primeros invariantes que fueron introducidos: los grupos de homología asociados a ciertos espacios que admiten triangulaciones finitas.

Pese a este origen esencialmente *computacional* de la Topología Algebraica, los posteriores desarrollos de la disciplina parecieron abandonar este camino, dedicándose más bien a generalizar de modo elegante y fructífero el puente entre distintas ramas que supone la noción de invariante. Como consecuencia de este enfoque cada vez más abstracto nacieron áreas como el Algebra Homológica y la Teoría de Categorías, que, a posteriori, han

determinado los resultados, y el lenguaje en que éstos eran presentados, en Topología Algebraica.

Es de destacar que, pese a lo anterior, los esfuerzos por mejorar el cálculo explícito de invariantes nunca desaparecieron totalmente y, por ejemplo, se estudió el problema de buscar *modelos combinatoriales* de los espacios topológicos mismos, con el objeto de facilitar posteriormente la lectura en ellos de los invariantes (se introdujeron así las nociones de CW-complejo, de complejo poliedral, de complejo simplicial, de conjunto simplicial, etc.). Mas no fue hasta mediados de los años 80, con el auge de la informática personal en todas las universidades, cuando el proyecto inicial centrado en la algorítmica no se retomó con energía, apoyado en la programación de computadores. Para entender este fenómeno, hay que tener presente que, como saben todos los que se dedican a las Matemáticas efectivas, la distancia entre lo *calculable* y lo *calculado* suele ser muy grande. Hasta que la técnica informática no permitió a los investigadores contar con herramientas suficientemente eficaces, muchos de los algoritmos de la Topología estaban totalmente alejados de las aplicaciones prácticas, lo que explica que los esfuerzos se concentrasen más en las partes estructurales que en las computacionales. Para conocer los diferentes proyectos relacionados con el cálculo por computador en Topología Algebraica que aparecieron en los 80 puede consultarse [1].

Entre las distintas propuestas que surgieron en esos años, apareció la teoría de *homología efectiva*, debida a F. Sergeraert, que pretende ser un marco general en el que abordar tanto el problema de la calculabilidad en Topología Algebraica como el desarrollo concreto de sistemas de cálculo simbólico en esa área. Uno de los cuellos de botella para el cálculo con ordenador en Topología Algebraica reside en que muchos de los espacios interesantes tienen una naturaleza infinita, es decir no admiten una descripción combinatorial en términos de un número finito de elementos. Por ejemplo, si empleamos los conjuntos simpliciales como modelos combinatoriales de los espacios topológicos, podemos decir que el algoritmo de Veblen para el cálculo de la homología mencionado más arriba se aplica sobre los conjuntos simpliciales de tipo finito (los que tienen esencialmente solo un número finito de células en cada dimensión), pero no sobre muchos otros que, pese a no ser de tipo finito, sí tienen homología de tipo finito y, por tanto, susceptible de ser calculada por medio de un computador. Sin entrar en detalles técnicos, el objetivo de Sergeraert al introducir la noción de *conjunto simplicial con homología efectiva* fue enriquecer los conjuntos simpliciales con información adicional que permita aplicar sobre ellos un algoritmo de cálculo de su homología, aun en el caso de que dichos conjuntos no sean de tipo finito. Para alcanzar dicho objetivo se debieron superar, como era de esperar, serias dificultades desde el punto de vista de la calculabilidad teórica, relativas a la manipulación en el computador de

representaciones de espacios de naturaleza infinita. Para estudiar la solución encontrada por Sergeraert (que se apoya en el uso esencial de técnicas de *programación funcional*), así como una presentación general de su teoría y los resultados genéricos de la misma, puede consultarse [6].

Por otra parte, el primer análisis detallado de un problema abordable utilizando las técnicas introducidas por Sergeraert se presentó en [3]. En esa memoria se estudió el cálculo de la homología de espacios de lazos iterados, primera familia de espacios que son de naturaleza infinita y que tienen, simultáneamente, un gran interés para el desarrollo teórico de la Topología Algebraica. Dicho estudio dió lugar a un programa de computador denominado EAT (por "Effective Algebraic Topology") desarrollado conjuntamente por Sergeraert y el autor de esta nota. EAT es un programa escrito en el lenguaje de programación Common Lisp y que, por tanto, puede ser ejecutado sobre cualquier equipo que disponga de dicho lenguaje. El sistema puede ser obtenido por ftp anónimo en la dirección **fourier.ujf-grenoble.fr** en el directorio `/pub/EAT`. Para facilitar su uso, se dispone de un manual de referencia [5] que contiene diversos ejemplos. En las siguientes secciones mostramos brevemente algunos de esos ejemplos.

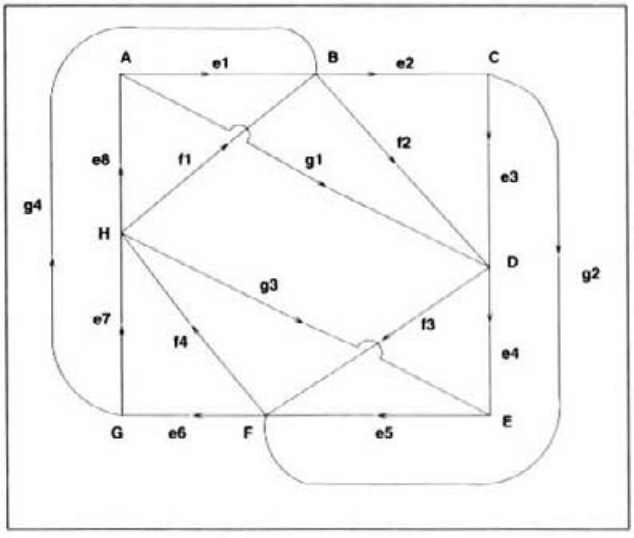
UN EJEMPLO ELEMENTAL

Como primer ejemplo, vamos a describir como puede ser usado el programa para trabajar con grafos. El grafo orientado que aparece en la siguiente figura es un caso particular de conjunto simplicial, con sólo dos conjuntos de símplices (no degenerados): el conjunto de los vértices y el conjunto de las aristas, cuyas caras son los vértices que las limitan. Este conjunto simplicial es construido por la función `build-finite-ss`. Los argumentos que se le dan a esta función se corresponden directamente con los elementos del grafo. En dimensión 0, los vértices son ternas como `0 'b ()`. Una tal terna debe ser interpretada del siguiente modo: el vértice llamado `b` es de dimensión 0 y no tiene ninguna cara. En dimensión 1 las aristas (orientadas) son representadas por medio de ternas como `1 é1 '(b a)`, con la siguiente interpretación: `e1` es de dimensión 1 y la lista de sus caras está constituida por los vértices `B` y `A` (en ese orden). En lo que sigue, las entradas para el programa serán escritas en caracteres inclinados y las salidas en letras de máquina de escribir.

```
(setf graph (build-finite-ss
        'a ; A es el punto base
        0 'b () 0 'c () 0 'd ()
        0 'e () 0 'f () 0 'g ()
        0 'h ()
        1 'e1 '(b a) 1 'e2 '(c b) 1 'e3 '(d c)
```

1 'e4 '(e d) 1 'e5 '(f e) 1 'e6 '(g f)
1 'e7 '(h g) 1 'e8 '(a h)
1 'f1 '(b h) 1 'f2 '(d b) 1 'f3 '(d b)
1 'f4 '(h f)
1 'g1 '(d a) 1 'g2 '(f c) 1 'g3 '(h e)
1 'g4 '(b g)))

A partir del objeto **graph**, que representa un conjunto simplicial, podemos construir un *complejo de cadenas* (un complejo de cadenas es un grupo graduado $\{C_n\}$, dotado de morfismos $d_n : C_n \to C_{n-1}$ que verifican la condición de diferencial: $d_n d_{n+1} = 0$) con sólo dos componentes no nulas en las dimensiones 0 y 1. Más concretamente, el complejo de cadenas tendrá solamente base no vacía en esas dos dimensiones y, entre ambas, un operador diferencial. La función **ss-cc** construye ese complejo de cadenas y, en particular, define la diferencial a partir de la descripción de las caras, almacenándola en la estructura que representa al complejo de cadenas.



(setf cc-graph (ss-cc graph))

Las siguientes instrucciones proporcionan las bases del complejo de cadenas **cc-graph** en dimensiones 0 y 1, respectivamente.

(cbs cc-graph 0)

```
(H G F E D C B A)
```

(cbs cc-graph 1)

```
(G4 G3 G2 G1 F4 F3 F2 F1 E8 E7 E6 E5 E4 E3 E2 E1)
```

La siguiente instrucción crea el elemento $e1+3*e2+5*f1+7*g1+11*g4$, que pertenece al grupo que es la componente del complejo de cadenas en dimensión 1 (dicho con otras palabras, dicho elemento es una *combinación* de grado 1). La diferencial de ese elemento es

$$3*c-5*h+7*d-8*a+14*b-11*g$$

(una combinación de grado 0). Estos cálculos son realizados respectivamente por las funciones `cmb` y `d-???`.

(setf comb (cmb 1 1 'e1 3 'e2 5 'f1 7 'g1 11 'g4))

```
---------------------------------{CMB 1}
<MNM 11 * G4>
<MNM 7 * G1>
<MNM 5 * F1>
<MNM 3 * E2>
<MNM 1 * E1>
---------------------------------------
```

(d-??? cc-graph comb)

```
---------------------------------{CMB 0}
<MNM 3 * C>
<MNM -5 * H>
<MNM 7 * D>
<MNM -8 * A>
<MNM 14 * B>
<MNM -11 * G>
---------------------------------------
```

Más interesante que estas pequeñas manipulaciones resulta el cálculo de la homología en dimensión 1, que proporciona un conjunto de lazos independientes en el grafo, en concreto 9 lazos en el caso que nos ocupa, descritos en términos de las aristas (que, como hemos visto, forman la base del complejo de cadenas en dimensión 1). Este cálculo es llevado a cabo por la función `cc-homology-gen`. El algoritmo para calcular $\mathcal{H}_n$, el n-ésimo grupo de homología de un complejo de cadenas, es bien conocido desde el comienzo mismo de la Topología Algebraica y fue ya recogido por Veblen en [7], como hemos evocado en la introducción. Llamemos `MZ` a la matriz del homomorfismo diferencial $d_n : C_n \to C_{n-1}$, donde C_n es el n–ésimo grupo del complejo de cadenas y sea `NB` la matriz correspondiente a $d_{n+1} : C_{n+1} \to C_n$. Si llamamos $\mathcal{B}_n$ a la imagen de d_{n+1} y $\mathcal{Z}_n$ al núcleo de d_n, entonces la definición del n–ésimo grupo de homología es: $\mathcal{H}_n = \mathcal{Z}_n/\mathcal{B}_n$. Este grupo es determinado a partir de las matrices `MZ` y `NB` por medio de

un conocido algoritmo de diagonalización (véase [7] o, para una referencia más reciente, [2]): la matriz `MZ` es utilizada para encontrar una base para el núcleo $\mathcal{Z}_n$ y la matriz `NB` permite determinar en $\mathcal{Z}_n$ una presentación por generadores y relaciones del grupo $\mathcal{H}_n = \mathcal{Z}_n/\mathcal{B}_n$.

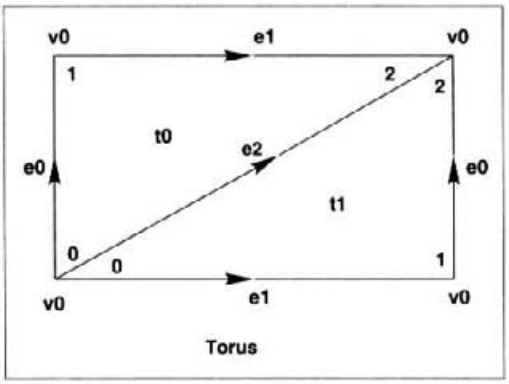
(cc-homology-gen cc-graph 1)

```
Homology in dimension 1 :

Component Z

Generator :
1     *  E8
1     *  E7
1     *  E6
1     *  E5
1     *  E4
1     *  E3
1     *  E2
1     *  E1

Component Z

Generator :
1     *  F1
1     *  E7
1     *  E6
1     *  E5
1     *  E4
1     *  E3
1     *  E2

Component Z

Generator :
1     *  G4
1     *  E6
1     *  E5
1     *  E4
1     *  E3
1     *  E2

Component Z

Generator :
1     *  G3
-1    *  E7
-1    *  E6
-1    *  E5

Component Z

Generator :
1     *  G2
-1    *  E5
-1    *  E4
-1    *  E3

Component Z

Generator :
1     *  G1
-1    *  E3
-1    *  E2
-1    *  E1

Component Z

Generator :
1     *  F4
-1    *  E7
-1    *  E6

Component Z

Generator :
1     *  F3
-1    *  E3
-1    *  E2

Component Z

Generator :
1     *  F2
-1    *  E3
-1    *  E2

---done---
```

LAS SUPERFICIES

El programa EAT puede ser utilizado, por supuesto, sobre conjuntos simpliciales más complicados, como las superficies. Por ejemplo, es bien

conocido que el toro puede ser representado por el diagrama siguiente, en el que los elementos etiquetados con el mismo símbolo deben ser identificados.




```
(setf toruss (build-finite-ss
        'v0 ; V0 es el punto base
        1 'e0 '(v0 v0) 1 'e1 '(v0 v0) 1 'e2 '(v0 v0)
        2 't0 '(e1 e2 e0) 2 't1 '(e0 e2 e1) ))
(setf toruss-cc (ss-cc toruss))
```

En este momento, un complejo de cadenas asociado al conjunto simplicial `toruss` ha sido definido y podemos consultar su homología en dimensiones 0, 1 y 2. Los resultados impresos significan respectivamente que en dimensión 0 el grupo de homología es isomorfo a $\mathbb{Z}$ (el espacio es conexo), en dimensión 1 el grupo de homología es isomorfo a $\mathbb{Z} \oplus \mathbb{Z}$, y en dimensión 2 el grupo de homología es isomorfo a $\mathbb{Z}$ (el toro es una superficie orientable y sin borde).

```
(dotimes (i 3) (cc-homology toruss-cc i))

Homology in dimension 0 :

Component Z

Homology in dimension 1 :

Component Z
Component Z

Homology in dimension 2 :

Component Z
```

Un Ejemplo más Sofisticado

Los ejemplos precedentes son bien conocidos y pueden ser encontrados en cualquier libro de texto. Se trata de espacios elementales puesto que sus complejos de cadenas asociados son de tipo finito, es decir, sus bases son finitas en cada dimensión. Pero, de hecho, estas aplicaciones de EAT no son más que sub-productos, puesto que el programa fue diseñado para el estudio de espacios cuyos complejos de cadenas no son de tipo finito, aunque su homología sí es de tipo finito. No podemos, en el marco de esta breve nota, entrar en los detalles involucrados en este género de cálculos, pero vamos a ilustrar las verdaderas posibilidades de EAT por medio de un ejemplo. Comenzamos por la construcción paso a paso de $\Omega^1(S^3)$, el primer espacio de lazos de la esfera de dimensión 3. (El espacio de lazos de un espacio se define como el conjunto de aquellas funciones continuas de la circunferencia en el espacio que preservan el punto base, dotado de la topología compacto-abierta; en realidad lo que el programa construye y manipula es un conjunto simplicial que es una versión combinatoria de dicho espacio; como se puede esperar dicho conjunto simplicial es de naturaleza infinita). En primer lugar, construimos la esfera S^3:

(setf s3 (sphere 3))

a continuación un objeto con homología efectiva asociado a ese conjunto simplicial:

(setf s3eh (ess-sseh s3))

y finalmente el objeto con homología efectiva que es un modelo de $\Omega^1(S^3)$:

(setf os3eh (loop-space-eh s3eh))

El cálculo de la homología en dimensión 2 es obtenido por evaluación de:

(homology os3eh 2)

```
Homology in dimension 2 :

Component Z

Generator :

-------------------------------------------------------------------{CMB 2}
<MNM -1 * <<LOOP (<PWR * <S3> ** 1>)>>>
-------------------------------------------------------------------
```

El resultado así impreso es la representación del opuesto del símplice "fundamental" del espacio de lazos. En la siguiente instrucción redefinimos

un generador de $H_2(\Omega^1 S^3)$ que está ligado al símbolo **fund-simp**:

(setf fund-simp (asm nil (loop3 nil '<S3> 1)))

```
<ASM * <<LOOP (<PWR * <S3> ** 1>)>>>
```

La siguiente instrucción nos permite construir el "lazo nulo" (el que corresponde a la aplicación constante sobre el punto base) en dimensión 2:

(setf null-simp (null-asm-loop 2))

```
<ASM 1-0 <<LOOP *>>>
```

Ahora estamos en condiciones de construir un nuevo objeto con homología efectiva, nuestro modelo del espacio $X = \Omega^1(S^3) \bigcup_2 D^3$, obtenido pegando el disco D^3 a través de una aplicación de grado 2. Esto se consigue con la siguiente llamada a la función **disk-paste-eh**:

(setf dos3eh (disk-paste-eh os3eh 3
(list fund-simp null-simp fund-simp null-simp) :new '<D3>))

La homología en dimensión 2 de este nuevo espacio X, ligado al símbolo **dos3eh**, no es nada sorprendente:

(homology dos3eh 2)

```
Homology in dimension 2:

Component Z/2Z

Generator :

------------------------------------------------------------------------{CMB 2}
<MNM -1 * <<LOOP (<PWR * <S3> ** 1>)>>>
------------------------------------------------------------------------
```

Sin embargo, si sobre X aplicamos de nuevo la construcción "espacio de lazos con homología efectiva", podemos obtener información acerca de la homología de ΩX, información que parece ser de difícil acceso utilizando métodos teóricos. Por ejemplo, EAT permite calcular $H_5(\Omega X)$.

(setf odos3eh (loop-space-eh dos3eh))

(homology odos3eh 5)

```
Homology in dimension 5 :

Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z

Component Z
```

EFICIENCIA Y TIEMPOS DE EJECUCIÓN

Presentamos en la siguiente tabla algunos resultados obtenidos sobre una Sun-Sparc 10. En general, los cálculos para espacios finitos con un número razonable de símplices (como grafos, superficies, etc.) son instantáneos. En los casos interesantes que involucran espacios que no son de tipo finito, los grupos en dimensiones bajas son calculados en breve tiempo, pero éste crece de un modo altamente exponencial con la dimensión. Para los dos espacios reseñados en la tabla, el cálculo de $\mathcal{H}_8$ hubiese requerido varios días de tiempo de CPU. Sin embargo, Sergeraert anuncia para 1998 una nueva versión del programa que mejorará considerablemente, entre otras cosas, el tiempo de ejecución. Algunos grupos de dimensión alta, que eran imposibles de calcular con la versión que está siendo comentada (por ejemplo, $\mathcal{H}_8$ de $\Omega^2 S^3$), son calculados en la nueva versión en un tiempo razonable.

Espacios	$\Omega^2 S^3$		$\Omega(\Omega S^3 \bigcup_2 D^3)$	
$\mathcal{H}_1$	$\mathbb{Z}$	30 mseg	$\mathbb{Z}/2\mathbb{Z}$	70 mseg
$\mathcal{H}_2$	$\mathbb{Z}/2\mathbb{Z}$	140 mseg	$\mathbb{Z}/2\mathbb{Z}$	170 mseg
$\mathcal{H}_3$	$\mathbb{Z}/2\mathbb{Z}$	170 mseg	$\mathbb{Z} \oplus (\mathbb{Z}/2\mathbb{Z})^2$	280 mseg
$\mathcal{H}_4$	$\mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/3\mathbb{Z}$	2.6 seg	$(\mathbb{Z}/2\mathbb{Z})^4$	3.0 seg
$\mathcal{H}_5$	$\mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/3\mathbb{Z}$	2.6 seg	$\mathbb{Z} \oplus (\mathbb{Z}/2\mathbb{Z})^6$	4.0 seg
$\mathcal{H}_6$	$(\mathbb{Z}/2\mathbb{Z})^2$	900 seg	$\mathbb{Z}/6\mathbb{Z} \oplus (\mathbb{Z}/2\mathbb{Z})^{12}$	850 seg

AGRADECIMIENTOS

El presente texto es, en gran medida, una traducción de parte del artículo [4], escrito junto a F. Sergeraert e Y. Siret, a los que agradezco su amabilidad por permitirme utilizar dicho material.

Referencias

[1] DOMÍNGUEZ, E., RUBIO, J.: *Computers in Algebraic Topology*, in The math. heritage of Gauss, World Scientific Publ., 1991, pp. 179-194.

[2] MACLANE, BIRKHOFF: *Algebra*, The MacMillan Company, 1967.

[3] RUBIO, J.: *Homologie effective des espaces de lacets itérés : un logiciel*, Thèse, Institut Fourier, Grenoble, 1991.

[4] RUBIO, J.; SERGERAERT, F.; SIRET, Y.: *An overview of EAT.* Aparecerá en SAC Newsletter, n. 3, 1998.

[5] RUBIO, J.; SERGERAERT, F., SIRET, Y.: *EAT: Symbolic Software for Effective Homology Computation*, **ftp://fourier.ujf-grenoble.fr/pub/EAT**, Institut Fourier, Grenoble, 1997.

[6] SERGERAERT, F.: *The computability problem in algebraic topology*, Advances in Mathematics, 1994, vol. 104, pp. 1-29.

[7] VEBLEN, O.: *Analysis Situs*, AMS Coll. Publ. 5, 1931.

Departamento de Informática e Ingeniería de Sistemas. Facultad de Ciencias. Universidad de Zaragoza. 50009 Zaragoza. e-mail: rubio@posta.unizar.es