VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797

Códigos tóricos sobre hipersimplejos

Jessica Rubí Lara Rosales[1], Nuria Sydykova Méndez[2], Néstor Fabián Bravo Hernández[3], Omar Alfredo Jaloma López[4], José de Jesús Liceaga Martínez[5], Jonatan García Pernia[6].
Universidad de Guanajuato/Cimat.
jessica.rubí@cimat.mx[1], nuria.sydykova@cimat.mx[2], nestor.bravo@cimat.mx[3], omar.jaloma@cimat.mx[4], jose.liceaga@cimat.mx[5], jonatan.garcia@cimat.mx[6].

Resumen

Sea $d \geq 1$ un entero positivo y sea $\mathcal{P}$ la envolvente convexa en $\mathbb{R}^s$ de los puntos enteros $e_{i_1} + \cdots + e_{i_d}$ tal que $1 \leq i_1 \leq \cdots \leq i_d \leq s$, donde e_j es el j-esimo vector unidad en $\mathbb{R}^s$. Dado un campo finito $\mathbb{F}_q$, determinamos la mínima distancia del código torico $\mathcal{C}_{\mathcal{P}}(d)$ asociado al hipersimplejo $\mathcal{P}$.

Palabras clave: Códigos de evaluación, códigos tóricos, mínima distancia, toro afín.

Introducción

Sea $S = K[t_1, \ldots, t_r] = \bigoplus_{d=0}^{\infty} S_d$ el anillo de polinomios sobre un campo finito $K = \mathbb{F}_q$, sea $1 \leq d \leq r$ un entero y sea $\mathcal{P}$ la envolvente convexa en $\mathbb{R}^r$ de todos los puntos enteros $e_{i_1} + \cdots + e_{i_d}$ tal que $1 \leq i_1 \leq \cdots \leq i_d \leq r$, donde e_i es el i -ésimo vector unidad en $\mathbb{R}^r$. El politopo $\mathcal{P}$ es llamado el d-ésimo hipersimplejo de $\mathbb{R}^r$. El toro afín de el espacio $\mathbb{A}^s := K^s$ esta dado por $T = (K^*)^s$, donde K^* es grupo multiplicativo de K. La cardinalidad de T es igual a $(q-1)^r$. El ideal de anulamiento de T, denotado por $I(T)$, es el ideal graduado de S generado por todos los polinomios homogéneos de S que se anulan en todos los puntos de T. Si $f \in S$ es polinomio homogéneo de grado d, denotamos el conjunto de ceros de f en T por $V_T(f)$.

El código tórico de $\mathcal{P}$ de grado d, denotado por $\mathcal{C}_{\mathcal{P}}(d)$ o solo simplemente $\mathcal{C}_d$, es la imagen de la función de evaluación

$$ev_d: \mathcal{L}_d \to K^m, f \mapsto (f(P_1), \ldots, f(P_m)),$$

Donde $\mathcal{L}_d$ es el K-subespacio vectorial de S_d generado por todos los $t^a = t_1^{a_1} \cdots t_r^{a_r}$, $a = (a_1, \ldots, a_r)$, tal que $a \in \mathcal{P} \cap \mathbb{Z}^r$ y $\{P_1, \ldots, P_m\}$ son todos los puntos del toro afín T. Un monomio t^a de S esta en $\mathcal{L}_d$ si y solo si t^a es libre de cuadrados y tiene grado d, esto es, un punto entero a de $\mathbb{N}^s$ esta en $\mathcal{P}$ si y solo si t^a esta en $\mathcal{L}_d$, donde $\mathbb{N} = \{0,1,2,3, \ldots\}$. Los parametros básicos de $\mathcal{C}_{\mathcal{P}}(d)$ son la longitud m, la dimensión $dim_K(\mathcal{C}_{\mathcal{P}}(d))$ y la mínima distancia

$$\delta(\mathcal{C}_{\mathcal{P}}(d)) := min\{|T \setminus V_T(g)| | g \in \mathcal{L}_d \setminus I(T)\}$$

Los códigos tóricos fueron introducidos por Hasen y han sido estudiados muy activamente en la última década. Estos códigos son códigos de variedades afines. Si reemplazamos $\mathcal{L}_d$ por S_d en la función de evaluación, la imagen resultante es el código de tipo Reed-Muller $C_T(d)$ sobre el toro afín T. Notemos que $C_T(d)$ es el código torico de la envolvente convexa en $\mathbb{R}^r$ de todos los puntos $a = (a_1, \ldots, a_r) \in \mathbb{N}^r$ tal que $|a| := \sum_{i=1}^{r} a_i = d$.

Pretendemos presentar una solución al siguiente problema.

Problema: Encontrar fórmulas para los parámetros básicos de un código tórico sobre un hipersimplejo.

ISSN 2395-9797

Además, mostramos un algoritmo en Python que calcula la mínima distancia de dicho código y ciertos cálculos que ejemplifican la formula encontrada para la mínima distancia.

Distancia mínima de códigos tóricos sobre hipersimplejos.

En esta parte determinamos los parámetros básicos de $\mathcal{C}_{\mathcal{P}}(d)$.

Sea $I \neq (0)$ un ideal graduado de S de dimensión de Krull k. La función de Hilbert de S/I es

$$H_I(d) := dim_K(S_d/I_d), d = 0,1,2, \ldots,$$

Donde $I_d = I \cap S_d$. Por teorema de Hilbert, existe un único polinomio $h_I(x) \in \mathbb{Q}[x]$ de grado $k-1$ tal que $H_I(d) = h_I(d)$ para $d >> 0$. El grado de el polinomio cero es -1.

El grado o multiplicidad de S/I, denotado por $deg(S/I)$, es el entero positivo dado por

$$deg(S/I) := (k-1)! \lim_{d\to\infty} H_I(d)/d^{k-1}, \text{ si } k \geq 1,$$

y $deg(S/I) = dim_K(S/I)$ si $k = 0$. Si $f \in S$, el ideal $(I:f) = \{g \in S | gf \in I\}$ se denomina el ideal cociente. Notemos que si f es un divisor de cero de S/I si y solo si $(I:f) \neq I$.

Lema 1 [3, Proposición 6.2.12, p. 662]. Sea X un subconjunto del espacio afín $\mathbb{A}^{r-1}$ sobre un campo finito K y sea $I(X) \subset S$ su ideal de anulamiento. Si $0 \neq f \in S$, entonces

$$|V_T(f)| = \begin{cases} deg(S/(I(X),f)), si (I(X):f) \neq I(X), \\ 0, si (I(X):f) = I(X). \end{cases}$$

Lema 2 [2, Lema 3.3]. Sea L el ideal de S generado por $t_1^{q-1}, \ldots, t_{r-1}^{q-1}$. Si $t^a = t_1^{a_1} \cdots t_r^{a_r}$ es un divisor de cer de S/I y $t^a \notin L$, entonces

$$deg(S/(L,t^a)) = (q-1)^{r-1} - (q-1-a_1) \cdots (q-1-a_{r-1}).$$

Proposición 3 [1, Proposición 4.3]. Si $0 \neq f \in \mathcal{L}_d, q \geq 3$ y $1 \leq d < s$,entonces

$$|V_T(f)| \leq (q-1)^{r-1} - (q-2)^d (q-1)^{r-d}.$$

Proposición 4 [1, Proposición 4.4]. Sea $\mathcal{C}_{\mathcal{P}}(d)$ el código torico de $\mathcal{P}$ de grado d. Entonces la longitud de $\mathcal{C}_{\mathcal{P}}(d)$ es $(q-1)^r$, $dim_K(\mathcal{C}_{\mathcal{P}}(d)) = \binom{r}{d}$ si $q \geq 3$ y $dim_K(\mathcal{C}_{\mathcal{P}}(d)) = 1$ si $q = 2$.

Demostración. La longitud de $\mathcal{C}_{\mathcal{P}}(d)$ es el número de puntos de T, esto es, $m = (q-1)^r$. Supongamos que $q \geq 3$. El número de monomios libres de cuadrados de S de grado d es $\binom{r}{d}$. Entonces, tenemos que $dim_K(\mathcal{L}_d) = \binom{r}{d}$. Concluimos el resultado al notar que $\mathcal{L}_d$ es isomorfo a $\mathcal{C}_{\mathcal{P}}(d)$. Si $q = 2$. Entonces, $T = \{(1,1,1, \ldots,1)\}$, m=1, $\mathcal{C}_{\mathcal{P}}(d) = \mathbb{F}_2$ y $dim_K \mathcal{C}_{\mathcal{P}}(d) = 1$.

Aquí presentamos el resultado principal de este articulo

Teorema 5 [1, Teorema 4.5]. Sea $\mathcal{C}_{\mathcal{P}}(d)$ el código tórico de $\mathcal{P}$ de grado d y sea $\delta(\mathcal{C}_{\mathcal{P}}(d))$ su mínima distancia. Entonces

ISSN 2395-9797

$$\delta(\mathcal{C}_{\mathcal{P}}(d)) = \begin{cases} (q-2)^d(q-1)^{r-d} & si \quad d \leq r, q \geq 3, \\ (q-2)^{r-d}(q-1)^d & si \quad r/2 < d < r, q \geq 3, \\ (q-1)^r & si \quad d = r, \\ 1, & si \quad q = 2. \end{cases}$$

A continuación, describimos algunos ejemplos y casos particulares de la formula anterior.

Como ejemplo, consideremos $q = 3$, $r = 3$ y $d = 1$. Así, trabajamos en el campo $K = \mathbb{F}_3$ y el anillo de polinomios $S = K[t_1, t_2, t_3]$. El toro afín es

$$T = \{111,112,121,122,211,212,221,222\}$$

y $\mathcal{L}_d$ está generado por t_1, t_2 y t_3. Luego, la matriz

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \end{bmatrix},$$

cuyos renglones son $ev_d(t_1)$, $ev_d(t_2)$, $ev_d(t_3)$, genera al código $\mathcal{C}_d$. Haciendo eliminación Gaussiana en esta matriz obtenemos la matriz generadora

$$G_{3,3,1} = \begin{bmatrix} 1 & 0 & 0 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 & 2 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 \end{bmatrix}$$

del mismo código, pero en su forma estándar. Por ser matriz generadora, todo elemento de $\mathcal{C}_d$ se puede expresar en la forma $(a, b, c)G_{3,3,1}$, con $a, b, c \in K$. A (a, b, c) lo llamaremos *vector de coeficientes*. Para calcular la mínima distancia debemos hallar a, b, c qué minimicen el peso de dicho vector. Un candidato a $\delta(\mathcal{C}_d)$ es 4, tomando $(1,0,0)$ como vector de coeficientes. Para verificar qué 4 es la distancia mínima, basta comprobar que ningún vector de coeficientes (sin todos ceros) da una "*codeword*" de menor peso.

Si hay al menos dos coeficientes distintos de cero, entonces al menos 4 de las coordenadas 1, 2, 3, 6, 7, 8 serán también distintas de cero. Esto resulta en una palabra con peso al menos 4. Por el contrario, sí hay exactamente un coeficiente distinto de cero, simplemente obtenemos un múltiplo de un renglón, y todos ellos tienen peso 4.

Caso general para $d = s$

Si $d = r$, entonces $\mathcal{L}_d = \langle t_1 t_2 \cdots t_r \rangle$. Luego, la matriz generadora de $\mathcal{C}_d$ es

$$G_{q,r,r} = [a_1 \quad \cdots \quad a_m],$$

donde a_i es el producto de todas las entradas de p_i, el i -ésimo elemento del toro. Para cada i se cumple que $a_i \neq 0$ por definición del toro. Por lo tanto, una entrada de una "*codeword*" es 0 si, y sólo si, todas las entradas son cero. Así, la distancia mínima del código es también la cardinalidad del toro $m = (q-1)^r$.

Idea para el caso $d = 1$

En este caso, $\mathcal{L}_d = \langle t_1, \ldots, t_r \rangle$. Al igual que en el ejemplo, compararemos los pesos de las palabras obtenidas con cada vector de coeficientes $(a_1, a_2, \ldots, a_r)$, las cuales son

$$\alpha(\boldsymbol{t}) = a_1 t_1 + a_2 t_2 + \cdots + a_r t_r.$$

Deseamos que α valga cero para la mayor cantidad de elementos del toro T. Sea j tal que $a_j \neq 0$. Para cada colección fija $u_1, \ldots, u_{j-1}, u_{j+1}, \ldots, u_r$, hay $(q-1)$ elementos en el toro que tienen esas entradas correspondientes. Y de esos $(q-1)$ elementos, exactamente uno de ellos es raíz de α, a saber, aquel con

$$u_j = -\frac{a_1 u_1 + \cdots + a_{j-1} u_{j-1} + a_{j+1} u_{j+1} + \cdots + a_r u_r}{a_j}.$$

Por lo tanto, la mayor cantidad de ceros que tiene una palabra es una parte $(q-1)$-ésima del total de $m = (q-1)^r$. Por tanto, la distancia mínima es

$$(q-1)^r \cdot \frac{(q-2)}{(q-1)} = (q-2)(q-1)^{r-1}.$$

A continuación, mostramos la implementación en Python de una función que permite calcular la distancia mínima de uno de nuestros códigos de evaluación. Esta toma como parámetros un primo q, que corresponde a la cardinalidad del campo, y dos enteros r y d, que corresponden al número de variables que tiene el anillo de polinomios sobre el que trabajamos y al grado de los monomios libres de cuadrados que consideraremos para generar el espacio vectorial, respectivamente.

ISSN 2395-9797

```
def minD(q, r, d):
    dist = (q - 1)**r
    coord = 1
    coordt = 0
    ceros = 0
    flag = False

    polinomios_prev = []
    for i in range(int(scipy.special.binom(r, d))):
      polinomios_prev.append(range(q))
    polinomios = list(itertools.product(*polinomios_prev))

    toro_prev = []
    for i in range(r):
      toro_prev.append(range(1, q))
    toro = list(itertools.product(*toro_prev))

    monomios = list(itertools.combinations(range(r), d))

    for polinomio in polinomios:
      for x in toro:
        for i in range(int(scipy.special.binom(r, d))):
          if polinomio[i] != 0:
            flag = True
```

ISSN 2395-9797

```
            for j in monomios[i]:
              coord = (coord * x[j]) % q
            coord = (coord * polinomio[i]) % q
            coordt = (coordt + coord) % q
            coord = 1
        if (coordt == 0) and (flag == True):
          ceros += 1
        coordt = 0
        flag = False
      if (q - 1)**r - ceros < dist:
        dist = (q - 1)**r - ceros
      ceros = 0

    return dist
```

ISSN 2395-9797

En primer lugar, inicializamos 5 variables: dist, coord, coordt, ceros, flag.

```
2.      dist = (q - 1)**r
3.      coord = 1
4.      coordt = 0
5.      ceros = 0
6.      flag = False
```

Estas sirven para lo siguiente:

- dist: almacena la distancia mínima. Su valor inicial es $(q-1)^r$, pues es el mayor valor que puede tomar.
- coord: es una variable auxiliar que usaremos para calcular coordt.
- coordt: almacena el valor de cada coordenada de nuestros puntos en el código de evaluación.
- ceros: indica la cantidad de coordenadas iguales a 0 que tiene un punto de nuestro código de evaluación.
- flag: indica si entramos o no a cierto ciclo de nuestra función.

Después de esto, creamos un conjunto de listas, cada una de las cuales almacena una posibilidad para los coeficientes que pueden tener los polinomios de nuestro subespacio vectorial.

Como el subespacio vectorial es generado por los monomios libres de cuadrado de grado d, entonces todo polinomio en éste es una combinación lineal de los r en d monomios, por lo que, si ordenamos a los monomios, una lista de r en d elementos, cada uno en $\{0,1,\ldots,q-1\}$, corresponde a exactamente un polinomio en el subespacio vectorial, y a cada polinomio se le puede asignar exactamente una lista de este estilo. Luego, podemos pensar los polinomios como si fueran estas listas.

```
8.      polinomios_prev = []
9.      for i in range(int(scipy.special.binom(r, d))):
10.       polinomios_prev.append(range(q))
11.     polinomios = list(itertools.product(*polinomios_prev))
```

ISSN 2395-9797

Ahora, definimos el toro, que simplemente es el conjunto de todos los puntos con r entradas que toman valores en $\{1, \ldots, q-1\}$.

```
13.    toro_prev = []
14.    for i in range(r):
15.      toro_prev.append(range(1, q))
16.    toro = list(itertools.product(*toro_prev))
```

Finalmente, creamos un conjunto en el que guardamos todos los monomios libres de cuadrados de grado d que hay en el anillo de polinomios. Para esto, lo que hacemos es considerar todos los subconjuntos con d elementos de $\{1, \ldots, r\}$ y pensamos a cada uno de ellos como un monomio. Por ejemplo, si $r = 4$ y $d = 2$, el subconjunto $\{1,4\}$ de $\{1,2,3,4\}$ corresponde al monomio $t_1 t_4$.

```
18.  monomios = list(itertools.combinations(range(r), d))
```

A continuación, viene un bucle en el cual evaluamos cada polinomio en todos los puntos del toro. Para cada monomio, vemos si su coeficiente asociado en el polinomio es distinto de 0 y en caso afirmativo evaluamos el punto en el monomio y multiplicamos por su coeficiente. Haciendo esto para cada monomio y sumando los resultados, obtenemos el resultado de evaluar nuestro polinomio en un determinado punto del toro.

```
20.    for polinomio in polinomios:
21.      for x in toro:
22.        for i in range(int(scipy.special.binom(r, d))):
23.          if polinomio[i] != 0:
24.            flag = True
25.            for j in monomios[i]:
26.              coord = (coord * x[j]) % q
27.            coord = (coord * polinomio[i]) % q
28.            coordt = (coordt + coord) % q
29.            coord = 1
```

ISSN 2395-9797

Si al evaluar nuestro polinomio en un punto del polinomio obtenemos 0, aumentamos en uno la cantidad de ceros que tendrá el vector que resulta al aplicar la función evaluación en él polinomio.

```
30.         if (coordt == 0) and (flag == True):
31.           ceros += 1
32.         coordt = 0
33.         flag = False
```

En caso de que el peso de Hamming (es decir, la cantidad de ceros) de nuestro vector sea menor que el mínimo peso de Hamming que hayamos obtenido con anterioridad, actualizamos este mínimo.

```
34.       if (q - 1)**r - ceros < dist:
35.         dist = (q - 1)**r - ceros
36.       ceros = 0
```

Una vez calculado el peso de Hamming mínimo de los vectores hacemos que la función regrese este valor, pues sabemos qué es igual a la distancia mínima del código.

```
38.     return dist
```

Referencias

1. D. Jaramillo, M. Vaz Pinto, and R. H. Villarreal, *Evaluation codes and their basic parameters*, Des. Codes Cryptogr. **89** (2021), 269--300.

2. J. Martínez-Bernal, Y. Pitones and R. H. Villarreal, *Minimum distance functions of complete intersections*, J. Algebra Appl. **17** (2018), no. 11, 1850204 (22 pages).

3. M. Kreuzer and L. Robbiano, *Computational linear and commutative algebra*, Springer, Cham, 2016.