Paralelización de la detección de una matriz copositiva mediante la evaluación de las facetas de un simplex unidad

J.M.G. Salmerón, L.G. Casado[1], y E.M.T. Hendrix[2],

Resumen— **El problema de encontrar el mínimo de un problema de optimización cuadrática estándar permite determinar cuando la matriz simétrica involucrada en la formulación es copositiva. El espacio de búsqueda del valor mínimo se realiza en un simplex unidad. Recientemente se ha desarrollado un algoritmo que evalúa las facetas de un simplex unidad para determinar si la función cuadrática es positiva en una faceta. Se desarrollaron diferentes tests para descartar facetas de la búsqueda donde se puede verificar que la función cuadrática es positiva o para reducir la dimensión de una faceta al ser la función cuadrática monótona en ella. Para una matriz simétrica de dimensión n el simplex unidad tiene $2^n - 1$ facetas por lo que su número crece exponencialmente con n. Aunque los test mencionados permiten reducir el número de facetas a evaluar, este número puede ser muy elevado para algunas matrices copositivas. Debido a que la evaluación de una faceta puede realizase de forma independiente de la evaluación de otra faceta, este problema es un buen candidato a ser resuelto de forma paralela. En la paralelización del algoritmo hay que tener en cuenta que varias facetas comparten subfacetas que no deben ser evaluadas si se probó que la función cuadrática es positiva en al menos una de las facetas padre. En este estudio se muestra una posible implementación paralela teniendo también en cuenta un uso eficiente de la memoria.**

Palabras clave— **Matriz copositiva, simplex unidad, faceta.**

I. Introducción

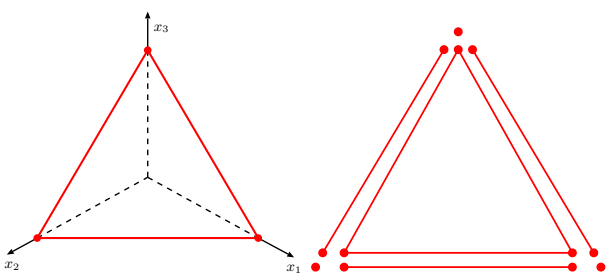
La copositividad juega un papel importante en la optimización combinatoria y cuadrática. Si se establece un problema de optimización lineal sobre el cono copositivo se obtienen reformulaciones exactas de problemas combinatorios. Como ejemplo, podemos mostrar el problema del Clique máximo, que es un problema NP-Completo [1]. Sea $\mathbf{1}$ el vector con todos los elementos a uno en la dimensión apropiada, $\omega(G)$ el número Clique de un grafo G, $I = \mathbf{1}\mathbf{1}^T$ la matriz con todos sus elementos iguales a uno, $t \in \mathbb{N}$ un escalar, y $Q = I - A_G$ una matriz obtenida de la matriz de adyacencia A_G del grafo G. El objetivo de este problema de programación copositiva es encontrar el menor valor de t tal que $tQ - I$ es copositiva, i.e., está en el conjunto $\mathcal{C}$ de las matrices copositivas,

$$\omega(G) = \text{mín}\{t : \ tQ - I \in \mathcal{C}\}. \tag{1}$$

Una matriz simétrica A de $n \times n$ se certifica que no es copositiva cuando $\exists x \in S_n, \ x^T Ax < 0$, en el simplex unidad

$$S_n = \{x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i = 1; \ x_i \geq 0, \quad i = 1, \ldots, n\}. \tag{2}$$

Se define una faceta $F_k \subseteq S_n$ como un subconjunto de los vectores unitarios e_i que están identificados por el vector binario $z \in \{0,1\}^n$, donde $z_i = 1$ si e_i está incluido en el conjunto de vértices. El número de vértices en la faceta está determinado por $m = \sum_i z_i$, el cual también determina su número de facetas (incluyéndose ella misma y los vértices) $k = 2^n - 1$. Por lo tanto, $z = \mathbf{1} = (2^n - 1)_2$ se refiere al simplex unidad indexado como F_{2^n-1}. Una faceta F_k es en realidad un símplice unidad de dimensión m (S_m) cuyas componentes están determinadas por z $\left(m = \sum_i z_i, \ z = (k)_2\right)$. La figura 1a) muestra un simplex unidad de dimensión 3. La figura 1b) intenta mostrar sus facetas: el propio simplex, tres lados y tres puntos. Los conjuntos de cuatro puntos más cercanos son en realidad el mismo punto en el espacio, donde solo una componente vale uno y las demás valen cero. Además de la matriz identidad



(a) Simplex unidad 3D. (b) Facetas simplex 3D.

Fig. 1: Simplex unidad y facetas 3D.

$E_m = (e_1, \ldots, e_m)$ en el espacio m dimensional, se usará también la matriz $D_m = E_m - \frac{1}{m}\mathbf{1}\mathbf{1}^T$ con las posibles direcciones sobre las facetas del simplex unidad.

Un problema relacionado es el de la programación cuadrática estándar (StQP)

$$f^* := \min_{x \in S_n} f(x) := x^T Ax. \tag{3}$$

Donde si $f^* < 0$ entonces A no es copositiva y si $f^* \geq 0$ entonces A es copositiva.

Existen algoritmos, como los de ramificación y acotación [2], que permiten certificar que una matriz no es copositiva o que es ϵ-copositiva. Mediante el

[1]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: {josemanuel,leo}@ual.es.
[2]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: eligius@uma.es.

refinamiento de símplices, la certificación de la ϵ-copositividad requiere mucha más computación que verificar que una matriz no es copositiva. En [3], donde se usó computación grid, se mostró la verificación de matrices no copositivas con una dimensión de hasta varios miles. Sin embargo, la certificación ϵ-copositiva de una matriz solo pudo realizarse para dimensiones de hasta $n = 22$ en un tiempo razonable.

Se sabe que el óptimo de StQP para una matriz no-positiva semi-definida se encuentra en una faceta. Sin embargo, los algoritmos de ramificación y acotación espaciales antes mencionados evalúan puntos en el interior relativo de S_n. Por ello se desarrolló un algoritmo que busca de forma sistemática el punto con el menor valor de la función cuadrática en cada una de las $2^n - 1$ facetas.

Este artículo tiene la siguiente organización. En la sección II se describen los desarrollos matemáticos para determinar cuando una matriz es positiva semi-definida y la función cuadrática (3) es monotona en una faceta. La sección III muestra una primera versión secuencial del algoritmo y sus desventajas, además de una versión mejorada basada en niveles. La sección IV presenta una posible versión paralela de la versión secuencial basada en niveles. Finalmente la sección V muestra los resultados preliminares de la primera versión del algoritmo secuencial y el potencial de mejora para desarrollar la versión paralela.

II. Caracterización del mínimo de un StQP

Para evaluar f en la faceta F_k es útil considerar la matriz A_k.

Definición 1: Dada una matriz A simétrica de $n \times n$ y un vector binario z, A_k es una matriz de $m \times m$ con $m < n$ y las filas, columnas i de A, seleccionadas si $z_i = 1$.

Por lo tanto, la optimización de $\min_{x \in F_k} x^T Ax$ en una faceta es equivalente a $\min_{x \in S_m} x^T A_k x$.

A. Caracterización del óptimo relativo interior

Un antiguo resultado de la programación cuadrática muestra que el mínimo de una función cuadrática indefinida (en la que estamos interesados) puede encontrarse en el límite del espacio factible [4]. Como tal, esto es interesante para el problema (3) ya que el mínimo no se encuentra en el interior de S_n. Sin embargo, puede estar en el interior de una de sus facetas. Se define el interior de $\breve{S}_m$ de S_m como

$$\breve{S}_m = \Big\{ x \in \mathbb{R}^m \mid \sum_{j=1}^{m} x_j = 1;\ x_j > 0,\ \ j = 1, \ldots, m \Big\}.$$

Proposición 1: Si $\exists x^* \in \operatorname{argmin}_{S_n} f(x) \in \breve{S}_n$, entonces $D_n A x^* = 0$ y $H = D_n A D_n$ es una matriz semi-definida.

La proposición 1 puede aplicarse a cualquier faceta $F_k \subset S_n$.

Corolario 1: Si $\exists x^* \in \operatorname{argmin}_{F_k} f(x) \in \breve{F}_k$, entonces $\exists y^* \in \breve{S}_m$, $D_m A_k y^* = 0$ y $D_m A_k D_m$ es positiva semi-definida.

Consideremos la matriz de vértices V con su correspondiente cobertura convexa (símplice Δ)

$$\begin{aligned} \Delta &= \{x = V\lambda, \textstyle\sum \lambda_i = 1, \lambda_i \geq 0, i = 1, \ldots, n\} \\ \Delta &= \{x = V\lambda, \lambda \in S_n\} \end{aligned} \tag{4}$$

y $Q = V^T AV$, tal que $f(x) = x^T Ax$ corresponde con $\lambda^T V^T AV \lambda = \lambda^T Q \lambda$. Usando está notación, el Corolario 1 se transforma en $DQ\lambda^* = DVAx^* = 0$. Aunque es complicado, este análisis puede extenderse a las facetas de menor dimensión. Sin embargo, la cuestión es ¿en qué faceta se encuentra el mínimo? Para ello, hay que tener en cuenta las derivadas direccionales DAx y $DQ\lambda$.

B. Consideraciones sobre monotonía

Proposición 2: Sea V una matriz de vértices, Δ de la ecuación (4), la matriz D con columnas d_i y $Q = V^T AV$. Si $d_i^T Q \geq 0$, entonces $\exists x^* \in \operatorname{argmin}_{\Delta} f(x)$ en la faceta que corresponde con el valor de $\lambda_i = 0$.

La importancia de este resultado es que, para el problema (3), si se cumple la condición de la proposición 2, se puede reducir la investigación de la copositividad de A a una faceta F_k de S_n que tiene una menor dimensión. Por lo tanto, podemos buscar un valor negativo de f en el área más prometedora de S_n reduciendo S_n a F_k. Cuando $d_i^T Q \geq 0$, la faceta F_k está definida por una matriz $\hat{V}$ de vértices de $m \times n$ que se obtiene quitando el vértice v_i de V. Entonces $\hat{Q} = \hat{V}^T A \hat{V}$. Esto se puede extender de forma directa a facetas de menor dimensión que si cumplen la Proposición 2 usando $\hat{Q}$ y el correspondiente D_m, se pueden reducir a una de sus sub-facetas.

III. Algoritmo secuencial

El Algoritmo 1 enumera las facetas F_k de S_n, marcando como *Chequeadas* aquellas donde no puede estar el mínimo de StQP (3) o el mínimo en la faceta F_k es positivo, hasta que se haya chequeado la lista completa de facetas o se haya encontrado un punto cuyo valor es negativo.

Si la matriz A no contiene valores negativos entonces es copositiva (línea 1). Esto también ocurre con las sub-facetas F_k, donde si A_k no contiene valores positivos, el optimo interior de F_k es positivo y las sub-facetas de F_k se marcan como chequeadas (línea 5).

Se usa para la Proposición 2 para chequear si f es monótona creciente en una de las componentes de la faceta (línea 12) y en caso afirmativo se reduce a una de sus sub-facetas de dimensión menor. Si no es el caso, se evalúa si H_k es positiva semi-definida (línea 15) condición necesaria para tener un óptimo interior. Para que una matriz sea positiva semi-definida sus autovalores deben ser positivos. Si H_k no es es positiva semi-definida, la solución de StQP (3) solo puede estar en sus sub-facetas. En el caso de que H_k sea positiva semi-definida, hay que encontrar el punto estacionario relativo interior de F_k que resuelva $D_m A_k x = 0$ en S_m (línea 16). Para ello hay que

Algoritmo 1 Test de copositividad basado en facetas

Entrada: A: matriz simétrica de $n \times n$.
if $A \geq 0$ **then**
  return A es copositiva
if $A_{i,i},\ i = 1, \ldots, n$, o $f(\frac{1}{n}\mathbf{1})$ es negativo **then**
  return A no es copositiva
$k = 2^n - 1$
Marcar las facetas $F_i,\ i = k, \ldots, n$ como no chequeadas.
while $k > 2$ **do**
  if F_k no chequeada **then**
    if $A_k \geq 0$ **then**
      Marcar las sub-facetas de F_k como chequeadas
      break
    if $\exists i,\ D_{mi}^T A_k > 0$, i.e. monótona creciente en la dirección i **then**
      Marcar las sub-facetas que no tengan $x_i = 0$ como chequeadas
    else
      if H_k es positiva semi-definida **then**
        if $\exists x^* \in S_m : D_m A_k x^* = 0$ **then**
          if $x^{*T} A_k x^* < 0$ **then**
            return A no es copositiva
          else
            Marcar las sub-facetas de F_k como chequeadas
  $k = k - 1$
return A es copositiva

resolver un problema de programación lineal que maximiza el valor del elemento mínimo, de forma que se tiene una solución en S_m o es cierto que no existe una solución de

$$\begin{aligned} &\text{máx}\, g, \\ &\text{s.t: } D_m A_k x = 0, \\ &\quad \mathbf{1}^T x = 1, \\ &\quad g \leq x_i,\ i = 1, \ldots, m. \end{aligned} \tag{5}$$

Si el resultado de (5) es $g < 0$, entonces no existe una solución en S_m y la solución de StQP (3) no está ni en F_k ni en sus sub-facetas.

El algoritmo 1 presenta dos principales inconvenientes: i) las facetas no se visitan de mayor a menor dimensión y ii) la lista completa de facetas chequeadas puede llegar a consumir mucha memoria.

En cuanto al primer inconveniente, la Figura 2 muestra el conjunto de facetas para n=4. Si se visitan en orden descendiente, se evaluaría antes F_{12} que tiene una dimension menor que F_7, que en caso de ser descartada junto con sus sub-facetas (F_3, F_5, F_6) y las sub-facetas de estas, descartarían más facetas del árbol que en el caso de descartar F_{12}. El número de niveles del árbol es n, estando S_n en el nivel n. Cada nivel tiene $\binom{n}{\text{nivel}}$ elementos. El nivel indica el número de bits igual a uno en los índices binarios de las facetas de ese nivel que, como se indicó anteriormente, se corresponde con las coordenadas que pueden tomar un valor en el rango [0,1].

En cuanto al segundo inconveniente podemos mostrar un ejemplo. Para $n = 28$, si se usa un `char` para almacenar si la faceta F_k ha sido chequeada o no, se necesitarían $2^n - 1$ bytes $\approx$ 2 GB.

Estos inconvenientes hacen pensar que es mejor trabajar con solo dos niveles, el actual y el siguiente. Las facetas del siguiente nivel se marcarían como chequeadas (True) dependiendo de las facetas

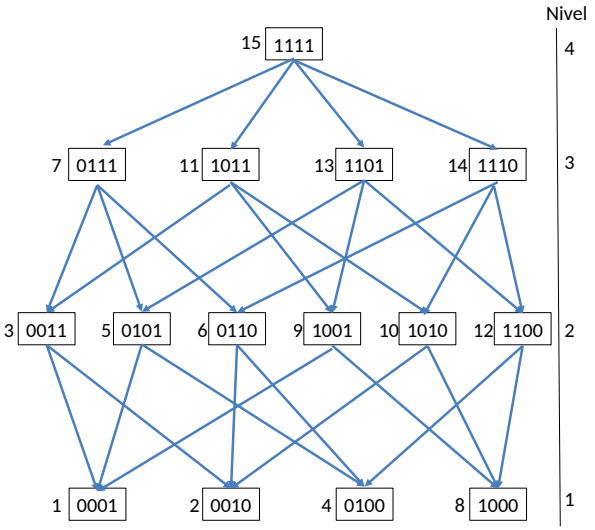


Fig. 2: Árbol de facetas para $n = 4$.

del nivel actual. Por lo tanto se usarán dos vectores de booleanos: NA (Nivel Actual) y NS (Nivel Siguiente) con $\binom{n}{nivel}$ y $\binom{n}{nivel-1}$ elementos respectivamente, donde *nivel* es el nivel actual. Se necesitarán las funciones k=IndexFaceta(n, nivel, i) que devuelve el indice de la faceta en la posición i del nivel, e i=PosNivelFaceta(n,nivel,k) que devuelve la posición de la faceta k en el vector de su nivel. Por ejemplo, en la Figura 2, 5=PosNivelFaceta(4,2,10) y 10=IndexFaceta(4,2,5). Como puede observarse en la Figura 2 cada faceta del nivel j tiene j subfacetas en el nivel $j - 1$.

Podría pensarse que usar dos vectores de `char` no reduce mucho la memoria necesaria ya que los vectores actual y siguiente de mayor tamaño se encontrarían en los niveles de la mitad del árbol. En la Figura 2, entre los niveles 3 y 2 ocupan 10 de los 15 `char` totales, pero esta diferencia se agranda conforme lo hace n.

El algoritmo 2 muestra el pseudocódigo para chequear la copositividad de una matriz simétrica basado en facetas del simplex unidad por niveles.

IV. Versión paralela

En esta sección se estudia la version paralela del algoritmo 2. El esquema se basa en un modelo maestro-esclavo donde el maestro reparte aquellas facetas del nivel actual que deben evaluarse entre los distintos esclavos. Un esclavo recibirá del master el (los) índice(s) del vector NA (Nivel Actual) marcados con False y evaluará las facetas correspondientes, devolviendo los índices del vector NS (Nivel Siguiente) que

Algoritmo 2 Test de copositividad basado en facetas por niveles

Entrada: A: matriz simétrica de $n \times n$.
1: **if** $A \geq 0$ **then**
2: **return** A es copositiva
3: **if** $A_{i,i},\ i = 1, \ldots, n$, o $f(\frac{1}{n}\mathbf{1})$ es negativo **then**
4: **return** A no es copositiva
5: NA_1=False ▶ Nivel Actual. False=No Chequeada
6: NS_i=False, $i = 1, \ldots, n$. ▶ Nivel Siguiente
7: **for** nivel=n to 2 **do**
8: **for** i=1 to $\binom{n}{\text{nivel}}$ **do**
9: k=IndexFaceta(n,nivel,i)
10: **if** NA_i==True or $A_k \geq 0$ **then**
11: Marcar las subfacetas de F_k en NS como True.
12: **else**
13: **if** $\exists i,\ D_{mi}^T A_k > 0$, i.e. monótona creciente en la dirección i **then**
14: Marcar las sub-facetas de F_k que no tengan $x_i = 0$ en NS como True.
15: **else**
16: **if** H_k es positiva semi-definida **then**
17: **if** $\exists x^* \in S_m : D_m A_k x^* = 0$ **then**
18: **if** ${x^*}^T A_k x^* < 0$ **then**
19: **return** A no es copositiva
20: **else**
21: Marcar las sub-facetas de F_k en NS como True.
22: **if** nivel>2 **then**
23: NA=NS
24: NS_i=False, $i = 1, \ldots, \binom{n}{\text{nivel}-2}$
25: **return** A es copositiva

deben establecerse a True (Chequeados).

La principal limitación de la versión paralela son las barreras que existen en la terminación de cada nivel, ya que no se debería empezar a procesar el siguiente nivel sin haber terminado el anterior, si se quiere evitar evaluar una faceta que podría no ser necesario tener que evaluarla en el nivel siguiente. Se podría pensar en una reordenación de los indices del nivel actual que permitan saber hasta qué indice del nivel siguiente no va a haber mas modificaciones, pero se deja como estudio para trabajos futuros.

Otra limitación del paralelismo es que el master tras recibir los índices del siguiente nivel (NS) de los esclavos que deben marcarse a True, los tiene que marcar y esta operación se hace de forma secuencial.

Existe un compromiso entre cuantos indices del nivel actual se le dan a cada esclavo y el número de escrituras secuenciales en el nivel siguiente que debe realizar el maestro. Mientras que enviar un solo indice al esclavo mejora el balanceo de la carga entre esclavos, enviar más de un indice hace que el esclavo elimine los índices repetidos, reduciendo así el número de escrituras secuenciales del maestro. El maestro también debe marcar los indices del siguiente nivel que correspondan a subfacetas de la faceta marcada como chequeada en el nivel actual.

Resumiendo, el trabajo del esclavo es evaluar una(s) faceta(s) y devolver si ha encontrado un punto negativo o los indices de las facetas del siguiente nivel a marcar como chequeadas. El trabajo del maestro es repartir las facetas a evaluar entre los esclavos y marcar los índices del siguiente nivel como chequeados (True) para las facetas del nivel actual que estén a True y los indices devueltos por los esclavos. Como el maestro es el único que escribe en la lista del siguiente nivel no se necesita exclusión mutua. Se podría pensar en un acceso paralelo a la lista del siguiente nivel ya que solo se realizan escrituras pero como muestra la Figura 2 el acceso a una faceta del nivel inferior desde otra del nivel superior no es uniforme por lo que es difícil mantener una localidad espacial de los datos.

El Algoritmo 3 muestra el pseudocódigo del máster y el Algoritmo 4 el del esclavo. Para no complicar los algoritmos, el maestro solo envía un índice cada vez a un esclavo.

Algoritmo 3 Maestro: Test de copositividad.

Entrada:
A: matriz simétrica de $n \times n$,
p: número de esclavos.
1: **if** $A \geq 0$ **then**
2: **return** A es copositiva
3: **if** $A_{i,i},\ i = 1, \ldots, n$, o $f(\frac{1}{n}\mathbf{1})$ es negativo **then**
4: **return** A no es copositiva
5: NA_1=False ▶ Nivel Actual. False=No Chequeada
6: NS_i=False, $i = 1, \ldots, n$. ▶ Nivel Siguiente
7: **for** pW=1 to p **do**
8: Inicializa Esclavo$_{pW}(A)$
9: **for** nivel=n to 2 **do**
10: Generar LFalseNA, LTrueNA. ▶ Listas con indices de NA a False y True
11: nLF=tamaño(LFalseNA)
12: nLT=tamaño(LTrueNA)
13: iLF=1; iLT=1 ▶ Contadores de las listas
14: pW=0 ▶ Número de esclavos trabajando
15: **while** $iLF < nLF$ and $pW < p$ **do**
16: $pW++$
17: Enviar a Esclavo$_{pW}$: nivel, LFalseNA$_{iLF}$
18: iLF++
19: **while** $iLT < nLT$ or $iLF < nLF$ **do**
20: **if** Resultados de esclavo: PuntoNegativo, LIndNS **then** ▶ No Bloqueante
21: **if** PuntoNegativo==True **then**
22: FinalizarEsclavos() ▶ Enviarles nivel=0
23: **return** A no es copositiva
24: **else**
25: **if** $iLF < nLF$ **then**
26: Enviar a este esclavo: nivel, LFalseNA$_{iLF}$
27: iLF++
28: **else**
29: $pW--$
30: Poner a True los indices en NS de LIndNS.
31: **else** ▶ No hay interacción con los esclavos
32: **if** $iLT < nLT$ **then**
33: Poner a True en NS los indices de las subfacetas de LTrueNA$_{iLT}$.
34: iLT++
35: **while** $pW > 0$ **do** ▶ resultados de esclavos pendientes
36: **if** Resultados de esclavo: PuntoNegativo, LIndNS **then** ▶ No Bloqueante
37: **if** PuntoNegativo==True **then**
38: **return** A no es copositiva
39: **else**
40: Poner a True en NS los indices de LIndNS
41: $pW--$
42: **if** nivel>2 **then**
43: NA=NS
44: NS_i=False, $i = 1, \ldots, \binom{n}{\text{nivel}-2}$
45: FinalizarEsclavos() ▶ Enviarles nivel=0
46: **return** A es copositiva

Al igual que el el Algoritmo 2, el maestro visita y actualiza cada nivel (ver las líneas 9 y 42). Para cada nivel se generan dos listas LFalseNA y LTrueNA con los indices del nivel actual (NA) que tienen los valores False y True respectivamente (ver línea 10). La primera tarea del maestro es la de alimentar con

trabajo a los esclavos de forma no bloqueante. Esto se realiza en el while de la línea 15. Una vez alimentados los esclavos, se está a la espera de los resultados (línea 20) que pueden ser: i) se ha encontrado un punto negativo (línea 21) o ii) los indices del siguiente nivel a marcar como chequeados (True), en cuyo caso, si hay trabajo, primero se le asigna un nuevo trabajo al esclavo (línea 26) y después se marcan los índices recibidos del siguiente nivel (línea 30). Puede ocurrir que los esclavos estén trabajando y no se haya recibido ningún resultado. En ese caso el maestro se dedica a establecer como chequeados los indices del nivel siguiente determinados por la lista LIndTrue del nivel actual (línea 32). Una vez que se han mandado todos los indices de las facetas a evaluar a los esclavos y se han marcado todos los indices del siguiente nivel, según LIndTrue, puede ocurrir que queden resultados de los esclavos por recibir y procesar. Esto se chequea en la línea 35. Una vez procesados los resultados pendientes, se puede empezar a procesar el siguiente nivel (ver línea 42).

El maestro finaliza su ejecución cuando se ha encontrado un punto negativo o se han chequeado todas las facetas en todos los niveles (el nivel 1 se chequeó en la linea 3). En el último caso la matriz es copositiva. Antes de finalizar, el maestro envía a los esclavos el nivel con valor 0 para que estos terminen su ejecución.

Algoritmo 4 Esclavo: Test de copositividad.

Entrada:
A: matriz simétrica de $n \times n$
1: Recibir del maestro: nivel, indice ▶ Bloqueante
2: **while** nivel> 0 **do**
3: LIndNS={∅} ▶ Indices de las sub-facetas en el siguiente nivel a marcar como True
4: k=IndexFaceta(n,nivel,indice)
5: **if** $A_k \geq 0$ **then**
6: Almacenar los índices de las sub-facetas de F_k en LIndNS
7: **break**
8: **if** $\exists i$, $D_{mi}^T A_k > 0$, i.e. monótona creciente en la dirección i **then**
9: Almacenar los indices de las sub-facetas de F_k que no tengan $x_i = 0$ en LIndNS
10: **else**
11: **if** H_k es positiva semi-definida **then**
12: **if** $\exists x^* \in S_m : D_m A_k x^* = 0$ **then**
13: **if** $x^{*T} A_k x^* < 0$ **then**
14: Enviar al maestro: True, {∅}
15: **else**
16: Almacenar los índices de las sub-facetas de F_k en LIndNS
17: Enviar al maestro: False, LIndNS
18: Recibir: nivel, indice ▶ Bloqueante
19: **return**

El esclavo se inicializa recibiendo la matriz A, tal como muestra el Algoritmo 4. El esclavo recibe el nivel y el indice de la faceta a procesar. Básicamente realiza las mismas evaluaciones sobre la faceta que se realizan en los algoritmos secuenciales 1 y 2, pero almacena en LIndNS los índices a poner como True en el siguiente nivel, en caso de no encontrar un punto negativo (línea 16). Tras realizar el trabajo, el esclavo está a la espera de recibir los datos para procesar más facetas y finaliza su ejecución cuando recibe un nivel con valor 0.

Como se ha dicho anteriormente mandar más de una faceta al esclavo permite que el maestro realice menos escrituras en el nivel siguiente. Otra posible estrategia es partir el vector del nivel actual en trozos. Cada trozo se le envía a un esclavo siguiendo un modelo parecido al usado en el algoritmo paralelo anterior, ademas del indice de inicio del trozo en el vector del nivel actual. El trozo de vector recibido por el esclavo puede contener facetas chequeadas y sin chequear. Si la faceta está chequeada, el esclavo genera los indices de las subfacetas en el nivel siguiente a marcar como chequeadas y si no está chequeada la evalúa. La evaluación de una faceta por parte del esclavo puede dar como resultado un punto negativo o una lista de subfacetas en el siguiente nivel que se unirá a la lista ya existente de otras facetas chequeadas o evaluadas anteriormente, eliminando los posibles duplicados. Esa lista se enviará finalmente al maestro. En este esquema los esclavos descargan de trabajo al maestro ya que ahora el maestro solo trocea el vector del nivel actual y actualiza los indices del nivel siguiente a partir de las listas recibidas de los esclavos.

V. Resultados preliminares

Hasta la fecha solo se ha realizado la implementación secuencial del Algoritmo 1 en MatLab. El problema resuelto es el de Clique máximo Jhonson8-2-4 [5] que tiene una matriz de adyacencia de 28 dimensiones y el clique máximo es de 4, es decir el menor valor de t para el cual la matriz $tQ - I$ en (1) es copositiva es $t = 4$.

El algoritmo se ejecutó en un ordenador con un Intel i5 y 8 GB de RAM. El tiempo de ejecución fue de unas 9 horas y 20 minutos para garantizar que la matriz $tQ - I$ es copositiva para $t = 4$. De las aproximadamente $268 \cdot 10^6$ facetas, no fue necesario chequear unas $177 \cdot 10^3$. El test de monotonía (Proposición 2) fue positivo solo unas 12.750 veces, se resolvieron unos $5 \cdot 10^3$ sistemas líneales (5) y se generaron y evaluaron 1.797 mínimos locales.

Aun usando un lenguaje de programación de alto nivel como MatLab en secuencial, se ha podido resolver un problema de 28 dimensiones. Sin embargo, otros autores usaron un grid de 26 clusters en 6 países solo pudieron verificar la copositividad de una matriz con una dimensión de hasta 22, con un límite de tiempo de 20.000 sg. (5,6 horas) [3]. Esto nos hace pensar que los algoritmos secuenciales basados en facetas son más eficientes que aquellos basados en Ramificación y Acotación usando un refinamiento espacial.

Debido al tiempo de ejecución del algoritmo secuencial, parece interesante abordar la implementación de la versión paralela descrita en este estudio. Se pretende seguir utilizando MatLab como entorno de programación paralelo [6]. Se estudiaran cuales de las posibilidades que ofrece MatLab para el desarrollo de algoritmos paralelos es más interesante para resolver este problema. En principio el uso de `parfor`

para recorrer el vector del nivel actual y actualizar el del siguiente nivel no es viable debido a que no se garantiza el acceso exclusivo de escritura en el vector del siguiente nivel.

Los resultados de la ejecución paralela sobre el problema Jhonson8-2-4 y otros tomados del reto de DIMACS [5] se presentarán en las XXIX Jornadas de Paralelismo que se celebrarán del 12 al 14 de Septiembre de 2018 en Teruel, organizadas por la Sociedad de Arquitectura y Tecnología de Computadores.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Ciencia e Innovación (TIN2015-66688) y financiado en parte por el Fondo Europeo de Desarrollo Regional (ERDF). J.M.G. Salmerón forma parte del programa estatal FPU.

Referencias

[1] I. M. Bomze, M. Dür, E. de Klerk, C. Roos, A. J. Quist, and T. Terlaky, "On copositive programming and standard quadratic optimization problems," *Journal of Global Optimization*, vol. 18, no. 4, pp. 301–320, 2000.

[2] S. Bundfuss and M. Dür, "Algorithmic copositivity detection by simplicial partition," *Linear Algebra and its Applications*, vol. 428, no. 7, pp. 1511–1523, 2008.

[3] J. Žilinskas and M. Dür, "Depth-first simplicial partition for copositivity detection, with an application to maxclique," *Optimization Methods and Software*, vol. 26, no. 3, pp. 499–510, 2011.

[4] Reiner Horst and Hoang Tuy, *Global Optimization. Deterministic Approaches*, Springer, 3rd edition, 1996.

[5] "Clique benchmark instances," http://cse.unl.edu/∼tnguyen/npbenchmarks/clique.html.

[6] "Matlab. parallel computing toolbox. user's guide," 2018, https://www.mathworks.com/help/pdf_doc/distcomp/distcomp.pdf.