Universidad Autónoma Metropolitana Azcapotzalco

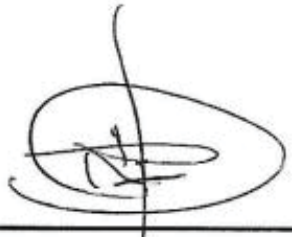
Lic. En Ingeniería en Computación

Generación de Delta-Matroides no binarios y no ternarios mediante cómputo paralelo

Modalidad: Proyecto Tecnológico

Frausto Tamayo Diego Leonardo
2132004045

Trimestre 18-I



Asesora: *Dra. María Guadalupe Rodríguez Sánchez*

Coasesor: *Dr. Adán Geovanni Medrano Chávez*

Declaratoria

Yo, María Guadalupe Rodríguez Sánchez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco

Firma

Yo, Adán Geovanni Medrano Chávez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco

Firma

Yo, Diego Leonardo Frausto Tamayo, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Resumen

Uno de los problemas en lo que respecta a la teoría de estructuras matroidales, es la $GF(2)$ y $GF(3)$ representabilidad de delta-matroides con matrices simétricas. Para la caracterización de estos delta-matroides ya existe un primer gran paso: la generación de obstrucciones con conjunto base de tamaño del conjunto base $N = 4$, pero aun existe la posibilidad de que existan más obstrucciones cuyo tamaño de su conjunto base es $N = 5$, $N = 6$ y $N = 7$, existe la conjetura de que no hay obstrucciones con conjunto base mayor que $N = 7$. En este trabajo se generan las obstrucciones con $N = 4$ verificando así los resultados precedentes a este proyecto y se generan también las extensiones con conjunto base $N = 5$ avanzado en la caracterización de delta-matroides $GF(2)$ y $GF(3)$ representables, todo esto con el apoyo de un modelo de cómputo paralelo que puede ser la clave para obtener la solución de problema, si se cuentan con los recursos computacionales adecuados.

Tabla de contenido

Introducción

Los matroides son estructuras estudiadas en el campo de la matemática combinatoria, estos son una generalización del concepto de independencia línea, inicialmente fueron propuestos por Whitney [1] en 1935, los delta-matroides fueron introducidos por A. Bouchet [2], como una generalización de los matroides. Un delta-matroide $D = (V, \mathcal{F})$ es una pareja formada por un conjunto V, llamado el conjunto base y una familia $\mathcal{F}$ de subconjuntos de V llamados conjuntos factibles, los cuales cumplen el axioma de intercambio simétrico:

$Si\ F_1, F_2$ son elementos de $\mathcal{F}$ y $\exists x \in (F_1 \Delta F_2)$, entonces $\exists y \in (F_1 \Delta F_2)$ tal que $F_1 \Delta \{x, y\} \in \mathcal{F}$.

Si el axioma se cumple para toda pareja F_i, F_j tal que $i, j \in [1, |\mathcal{F}|]$, entonces D es un delta-matroide.

Ejemplo:

Sea $D = (V, \mathcal{F})$ donde:

V : es un conjunto de enteros; $V = \{1,2,3,4\}$.

$\mathcal{F}$: es una familia de subconjuntos de V; $\mathcal{F} = \{\{1,2\}, \{3,4\}, \{1,2,3,4\}\}$.

El concepto de independencia lineal es fundamental en la formulación de la teoría de las estructuras matroidales. Conceptos propios de la misma son los conjuntos independientes, bases, circuitos entre otros, formando en cuerpo matemático sólido, por lo que sus aplicaciones y relaciones con otras áreas de la matemática discreta han fomentado investigaciones muy fructíferas en los últimos 30 años. Una de las áreas donde más se relacionan los matroides y Delta-matroides es la teoría de gráficas, un ejemplo importante es la liga de los Delta-matroides con los circuitos eulerianos en gráficas.

En este proyecto generamos aquellos delta-matroides que no poseen una representación matricial en el campo $GF(2)$ y $GF(3)$ para el caso simétrico. El problema de reprensentabilidad de estas estructuras es muy difícil tanto desde el punto de vista teórico como computacional, pues las operaciones necesarias para generar y estudiar los datos crecen de manera exponencial; en este proyecto solo se resuelve de manera acotada y con cómputo paralelo el problema, para algunas cardinalidades del conjunto base de los delta-matroides en cuestión, para aprovechar y optimizar el uso de recursos computacionales.

Objetivos

Objetivo general

Diseñar e implementar mediante cómputo paralelo un algoritmo para obtener una lista de extensiones a partir de delta-matroides que son obstrucciones en el campo $GF(2)$ y que permitirán encontrar nuevas obstrucciones que caractericen a delta-matroides no binarios y no ternarios de tamaño $N \geq 4$.

Objetivos específicos

- Diseñar e implementar un módulo (MDDM) que determine si una familia de subconjuntos es un delta-matroide.
- Diseñar e implementar un módulo (MGCP) que genere el conjunto potencia de un delta-matroide (con acceso concurrente).
- Diseñar e implementar un módulo (MGEV) que genere extensiones por borrado y contracción.
- Diseñar e implementar un módulo (MER) que descarta los delta-matroides que son representables en $GF(3)$.
- Diseñar e implementar un módulo (MEI) que evalúe isomorfismo entre delta-matroides

Marco teórico

Conceptos fundamentales de Delta-Matroides

Delta-Matroides Delta-Equivalentes:

Un delta-matroide D_1 es delta-equivalente a otro D_2 (ambos con el mismo conjunto base V) si se puede obtener D_2 a partir de D_1 mediante la operación ***twisting*** (Δ)**,** la cual consiste en trasformar la familia de factibles de D_1 aplicando diferencia simétrica a cada conjunto factible de D_1 con un conjunto $X \subseteq V$.

Por ejemplo:

Consideremos el siguiente delta-matroide:

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

Y el Factible $X = \{2,4\}$.

Por lo tanto:

$\mathcal{F}\Delta X = \{\{2,4\}, \{1,2,4\}, \{2,3,4\}, \{2\}, \{1,2\}, \emptyset, \{2,3\}, \{1\}, \{3\}\}$.

Se puede llamar a $F\Delta x$ como F' y se ordena lexicográficamente:

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

F' es un Delta-matroide ya la que la operación *twisting* da como resultado otro delta-matroide, además F y F' son delta-equivalentes. Podemos llegar de $\mathcal{F}$ a $\mathcal{F}'$ con $\mathcal{F}\Delta\{2,4\}$ y viceversa.

Huella de un Delta-Matroide

Es una herramienta auxiliar que usamos para clasificar eficientemente conjuntos de delta-matroides, puede ser definida formalmente como:

Sea un delta-matroide $D = (V, \mathcal{F})$ la huella de D será el vector α con $V + 1$ componentes $\alpha = (\alpha_0, \alpha_1, \dots, \alpha_{|V|})$, donde $\alpha_i = |\{F \in \mathcal{F}: |F| = i\}|$. Es decir, a cada componente α_i de α le corresponde el número de conjuntos factibles que tiene cardinalidad i $(i = 0,1, \dots \dots, |V|)$.

Ejemplo:

Se tiene el delta-matroide: $D = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{2,4\}, \{1,2,4\}, \{2,3,4\}\}$ su huella será el vector donde cada componente i es el número de subconjuntos de cardinalidad i en D es decir:

$$h(D) = (1,3,3,2,0)$$

note que en D hay un subconjunto de cardinalidad 0 ($\emptyset$), tres subconjuntos de cardinalidad 1 ($\{1\}, \{2\}, \{3\}$), tres subconjuntos de cardinalidad 2 ($\{1,2\}, \{2,3\}, \{2,4\}$), dos subconjuntos de cardinalidad 3 ($\{1,2,4\}, \{2,3,4\}$), y cero subconjuntos de cardinalidad 4.

Isomorfismo de Delta-Matroides

Un Delta-Matroide $D_1 = (V_1, \mathcal{F}_1)$ es isomorfo a otro $D_2 = (V_2, \mathcal{F}_2)$ sí se puede establecer una función biyectiva φ de V_1 a V_2 tal que para todo X subconjunto de V_1, el conjunto $\varphi(X)$ es un conjunto factible en D_2 si y solo si X es un conjunto factible en D_1. Consideremos el siguiente ejemplo.

$$D_1 = \{\emptyset, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$
$$D_2 = \{\emptyset, \{1,3\}, \{1,5\}, \{3,5\}\{1,3,5\}\}$$

Función Biyectiva:
1 ------- 5
2 ------- 3
3 ------- 1

Si para cada elemento en los subconjuntos de D_1 los sustituimos por su correspondiente en la función biyectiva descrita obtendremos un delta-matroide idéntico a D_2.

Operaciones de menores

A los delta-matroides se les puede aplicar dos tipos de operaciones, llamadas operaciones de menores, estas son operaciones que pueden hacerse sobre la familia de subconjuntos factibles de un delta-matroide D_O. Las operaciones de menores son: borrado y contracción. Para comprenderlas, tomemos el siguiente ejemplo:
Sea $D = (V, \mathcal{F})$ con $V = \{1,2,3\}$ y $\mathcal{F} = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$ un delta-matroide.
Sea $x = 1 \in V$.

Operación de menor por borrado (\ ): Dado un elemento $x \in V$, $D \backslash x$ denotará el Delta-

matroide cuyos conjuntos factibles son todos los conjuntos factibles de D que no contienen a x. Respecto al ejemplo: el conjunto de factibles es $\mathcal{F}\backslash 1 = \{\emptyset, \{2\}, \{3\}, \{2,3\}\}$.

*Operación de menor por contracción (*):* Dado un elemento $x \in V$, la operación $D * x$ denotará el Delta-matroide cuyos conjuntos factibles se forman a partir de todos los factibles que sí contienen a x, pero eliminando x de los factibles: en el ejemplo, el conjunto de factibles de $D * 1\ es\ \mathcal{F} * 1 = \{\{2\}, \{3\}, \{2,3\}\}$. Note que $\{2\}$ viene del factible $\{1,2\}$ en $\mathcal{F}$, $\{3\}$ del factible $\{1,3\}$ y $\{2,3\}$ del factible $\{1,2,3\}$ también en $\mathcal{F}$.

Conceptos sobre conjuntos, campos y representabilidad de Delta-Matroides

Conjunto potencia

Es el conjunto formado por todos los subconjuntos de V denotado por $P(V_K)$ donde $K = |V|$. Por ejemplo, consideremos:

$V = \{1,2,3,4\}$.

Los elementos del conjunto potencia $P(V_4)$ es:

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

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

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

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

$$\emptyset$$

Campos y Delta-Matroides

En álgebra moderna se le llama cuerpo o campo a una estructura definida sobre un conjunto finito o infinito $\boldsymbol{K}$ sobre el cual se pueden definir las operaciones adición y multiplicación con las siguientes propiedades:

- ***K*** es cerrado para la adición y la multiplicación:

$$\forall\, a, b\ \in K : a + b = c \mid c \in K$$

$$\forall\, a, b\ \in K : a * b = c \mid c \in K$$

- Conmutatividad para la adición y la multiplicación:

$$\forall\, a,\ b\ \in K,\ a\ +\ b\ =\ b\ +\ a$$

$$a \cdot b\ =\ b \cdot a$$

- Asociatividad de la adición y multiplicación

$$\forall\, a,\ b,\ c\ \in\ K,\ a\ +\ (b\ +\ c) =\ (a\ +\ b) +\ c$$

$$a \cdot (b \cdot c)\ =\ (a \cdot b) \cdot c.$$

- Existencia de un elemento para la adición y la multiplicación: para la adición y la multiplicación:

$$\exists\, un\, 0 \mid \forall\, a\ \in K : a\ + 0 = a$$

$$\exists\, un\, 1 \mid \forall\, a\ \in K : a\ *\ 1 = a$$

- Existencia del elemento inverso:

$$\forall\, a\ \in K, \exists\, -a \in K\ tal\ que - a + a = 0$$

$$\forall\, a\ \in K, a^{-1}\ \in K\ tal\ que\ a^{-1} * a = 1$$

El conjunto de los números **Reales** es un ejemplo de campo, en este proyecto se trabajó sobre los campos finitos $GF(2)$ y $GF(3)$ a los cuales también llamamos $campo\ binario$ y $campo\ ternario$ respectivamente:

$GF(2)$ posee dos elementos, nosotros usamos 0 y 1 como los elementos del campo. Las tablas de adición y multiplicación se dan a continuación:

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

GF(3) posee tres elementos, nosotros usamos -1 , 0 y 1 como los elementos del campo. Las tablas de adición y multiplicación se dan a continuación:

+	0	1	-1
0	0	1	-1
1	1	-1	0
-1	-1	0	1

×	0	1	-1
0	0	0	0
1	0	1	-1
-1	0	-1	1

Cuando A. Bouchet propuso los delta-matroides [2], describió las propiedades de representabilidad de estas estructuras. Los matroides pueden representarse con matrices formadas por arreglos maximales de vectores linealmente independientes vinculados a las bases del matroide. En forma similar, los delta-matroides pueden ser representados con una matriz simétrica o anti simétrica, cuyas *submatrices principales* no singulares correspondan a los conjuntos factibles, en un campo dado, por ejemplo:

M	1	2	3	4	5
1	1	0	1	1	0
2	0	0	0	1	1
3	1	0	1	1	1
4	1	1	1	0	0
5	0	1	1	0	0

D ={ Ø,
{1},{3},
{1,4},{2,4},{2,5},{3,4},{3,5},
{1,2,4},{1,2,5},{1,3,5},{2,3,4},{2,3,5},
{1,2,4,5},{1,3,4,5},
{1,2,3,4,5}
}

Imagen 1: ilustración de un delta-matroide y su matriz de representacion

El delta-matroide D, tiene como representación matricial a la matriz M a la izquierda, cada conjunto factible en D, tiene asociada una submatriz en M que es no singular, es decir, que su determinante es diferente de cero. De esta forma, se dice que un delta-matroide D es binario si es posible encontrar una matriz de representación con entradas en el campo $GF(2)$ y ternario si las entradas de la matriz están en $GF(3)$, con la propiedad anterior. D puede ser binario y ternario al mismo tiempo.

Al aplicar una operación de menor sobre un delta-matroide D_O, se obtiene un delta-matroide menor D_m, de tal forma que D_O contiene como menor a D_m. Si D_m no es representable sobre un campo K, D_O hereda de D_m la propiedad de **no representabilidad sobre K**, por lo tanto, si D_m no es representable sobre K esto implicará que cualquier delta-matroide que contenga como menor a D_m, tampoco será representable sobre el campo K.

A. Bouchet y A. Duchamp en 1991 [3] caracterizaron a los Delta-matroides binarios mediante una lista de cinco menores excluidos u obstrucciones:

Teorema: sea $D = (V, \mathcal{F})$ un delta-matroide, D será binario si y sólo si no contiene como menor a alguno de los siguientes delta-matroides:

$S_1 = \{\emptyset, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

$S_2 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$

$S_3 = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}$

$S_4 = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}$

$S_5 = \{\emptyset, \{1,2\}, \{1,4\}, \{2,3\}, \{3,4\}, \{1,2,3,4\}\}$

donde S_i, $i = 1,2,\ldots,5$; es la lista de los conjuntos factibles de los cinco menores excluidos. Los delta-matroides $S_i, i = 1,2,3,5$; no son $GF(2)$ representables, pero si tienen una representación sobre el campo $GF(3)$. Sus extensiones pueden ser o no $GF(3)$ representables. Se muestran a continuación:

S_1 tiene como representación matricial en $GF(3)$ a la matriz

$A_1 =$

	1	2	3
1	0	1	1
2	1	0	1
3	1	1	0

S_2 tiene como representación matricial en $GF(3)$ a la matriz

$A_2 =$

	1	2	3
1	−1	1	0
2	1	1	−1
3	0	−1	−1

S_3 tiene como representación matricial en $GF(3)$ a la matriz

$$A_3 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & -1 \\ 3 & 1 & -1 & -1 \end{array}$$

S_5 tiene como representación matricial en $GF(3)$ a la matriz

$$A_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & -1 \\ 3 & 1 & 1 & 0 & 0 \\ 4 & 1 & -1 & 0 & 0 \end{array}$$

Cómputo Paralelo

El cómputo paralelo es un paradigma computacional para resolver problemas cuya principal premisa es ejecutar múltiples tareas de manera simultánea en un problema descompuesto bajo el principio de *"divide y vencerás"*, de tal forma que el tiempo de la múltiple ejecución de tales tareas simultaneas resulte en una mejor optimización que resolverlas de manera secuencial. Desde la época arcaica de las computadoras hasta la actualidad se han hecho grandes avances en tecnología de procesadores para lograr paralelizar el trabajo que realiza una computadora, tan es así, que hoy en día es bastante común (o casi la norma) encontrarnos con arquitecturas de procesadores que tienen un diseño multinúcleo y, en general, con el hardware necesario para explotar en mayor o en menor medida el paralelismo, incluso en procesadores de propósito general. Es aquí donde radica un punto importante en la realización de este proyecto, es muy común que en la investigación científica (en este caso en las matemáticas discretas) se presenten problemas donde el potencial de paralelizar sea muy grande, por lo que mostrar ejemplos, pautas y metodologías para paralelizar la ejecución de un algoritmo para resolver un problema en matemáticas es una gran contribución al campo de la ingeniería, dado que la tendencia en la tecnología está apuntando cada vez más a este camino.

Taxonomía de Flynn

Es una forma de clasificar el diseño de arquitecturas de computadoras paralelas, esto mediante la forma de procesar las instrucciones contra la forma de procesar los datos. Esta clasificación contempla cuatro categorías:

- **SISD**: se procesa únicamente una instrucción y únicamente un dato por ciclo de reloj.
- **MISD**: Cada unidad ejecuta una instrucción distinta con el mismo dato (tiene poca utilidad en la práctica).
- **SIMD**: Todas las unidades ejecutan la misma instrucción, cada unidad procesa un dato distinto y todas las unidades operan simultáneamente.

- **MIMD**: Cada unidad ejecuta una instrucción distinta Cada unidad procesa un dato distinto Todas las unidades operan simultáneamente

En este proyecto se trabajó con una maquina bajo la arquitectura **MIMD** por lo que se diseñó un algoritmo que pueda aprovechar los recursos de paralelismo que brinda esta arquitectura, también cabe mencionar, que no se trabajó sobre un paralelismo real sino más bien con concurrencia, de acuerdo con las características de esta arquitectura y los estándares para manejar el paralelismo dados por el SO de la máquina.

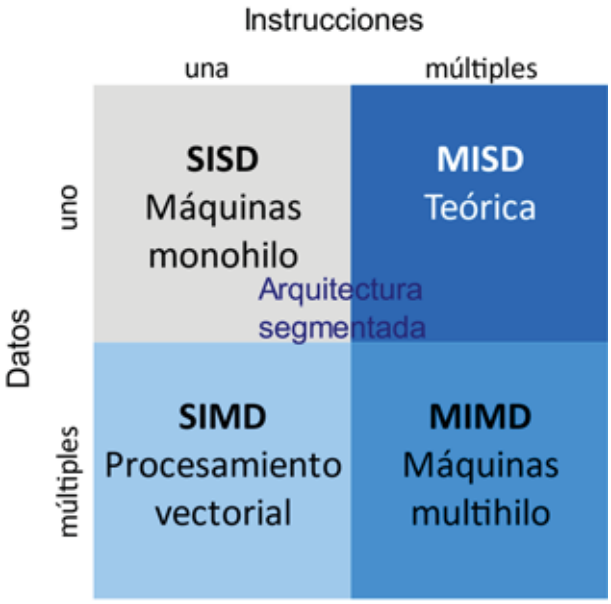


Imagen 2: Diagrama de clasficacion Flynn de arquitecturas paralelas

Concurrencia a nivel de tareas

Consiste en diseñar una forma de resolver un problema mediante la segmentación en etapas, de esta manera esas etapas pueden reordenarse y combinarse en grupos que luego son ejecutadas en paralelo sin cambiar el resultado del programa. Para que una tarea pueda ser descompuesta en múltiples etapas, se es necesario que las subtareas que resulten de esa descomposición tengan dependencia de datos entre ellas, es decir que una etapa dependa de la salida de otra etapa de forma que se cree un flujo de datos, y así cada etapa se puede ejecutar de manera simultánea.

Concurrencia a nivel de datos

Esta estrategia se basa en operar concurrentemente un subconjunto de datos del problema en cuestión, este además debe cumplir el requisito de no tener dependencia de datos, de esta manera se puede asignar múltiples hilos a operar una porción de los datos que necesiten ser procesados. Usualmente todo el conjunto de datos se encuentra en una estructura de datos que es accesible a todas las unidades o hilos de procesamiento que realizan sus cálculos y tareas de manera simultánea. Es muy común encontrar que aplicaciones científicas y de ingeniería presenten esta condición: la independencia de datos, como lo es nuestro caso.

Riesgos de concurrencia

Los riesgos en la concurrencia es un problema de la arquitectura paralela de un procesador segmentado, ya que dependiendo del problema que se trate, este puede requerir de instrucciones que no pueden ser ejecutadas al mismo tiempo, debido a su

naturaleza; los podemos asociar en tres tipos: riesgos de datos, riesgos de salto o de control y riesgos estructurales [4].

- **Riesgos de datos**: ocurren cuando una tarea se encuentra atascada en un paso porque necesita del resultado de otra tarea para completarse, en otras palabras, este riesgo surge en la condición de una tarea con dependencia de datos.
- **Riesgos estructurales**: estos se producen por la incapacidad de la arquitectura de hardware de ejecutar cierta combinación de instrucciones en el mismo ciclo de reloj.
- **Riesgos de salto o de control**: este surge en las situaciones en las cuales existe una bifurcación en la ejecución de instrucciones, debido a el resultado de otra tarea, de manera que existe una cierta dependencia de un dato aun no disponible y el procesador no sabría que bifurcación tomar.

Una de las metodologías para prevenir los riesgos es el llamado método de **inserción de burbujas**, el cual consiste en primera instancia en determinar si existe un riesgo, y si es así entonces insertar No operaciones o operaciones **NOP**, que permita al procesador esperar hasta que la independencia o la estructura de hardware esté disponible sin ningún riesgo. Para los riesgos de control existen técnicas como **Ejecución especulativa** o **Predictor de saltos** con mecanismos y metodologías que predicen o aproximan el resultado de una bifurcación permitiéndole al procesador proceder y no insertar NOP.

HyperThreading

El término *hyperthreading* es el nombre que INTEL da a su implementación de la tecnología ***Multithreading* Simultáneo** la cual consiste en la ejecución de múltiples hilos de programas en el mismo procesador físico, para lograrlo, esta tecnología se basa en aprovechar los periodos de no operaciones (que la lógica de control debería de insertar para evitar los riesgos de concurrencia) cambiando de contexto, es decir ejecutar otro hilo, y de esta manera aprovechar ciclos de reloj del procesador que de otra forma serian desperdiciados.

Desarrollo del proyecto

Para desarrollar este proyecto se escribió un programa en el lenguaje de programación orientado a objetos C++ al que denominamos PT-GeneracionDM, que realiza los cálculos necesarios para obtener las obstrucciones de delta-matroides no representables sobre $GF(2)$ y $GF(3)$ para ello requerí el uso de las siguientes herramientas:

- El compilador G++ con el estándar de compilación C++14.
- La API OpenMP 4.5.
- El editor de código Visual Studio Code.
- La herramienta "ProyectoDelta: paquete computacional para la investigación en DeltaMatroides" [5], software para hacer algunos cálculos secundarios.

El desarrollo de este proyecto constó de dos fases: escritura del programa secuencial y paralización del programa secuencial. Además, el programa PT-GeneracionDM, está programado con los mecanismos necesarios para realizar dos tipos de ejecución: Modelo Secuencial y Modelo paralelo; esto con el objetivo de hacer un análisis del rendimiento ganado.

Descripción del algoritmo para la generación de extensiones

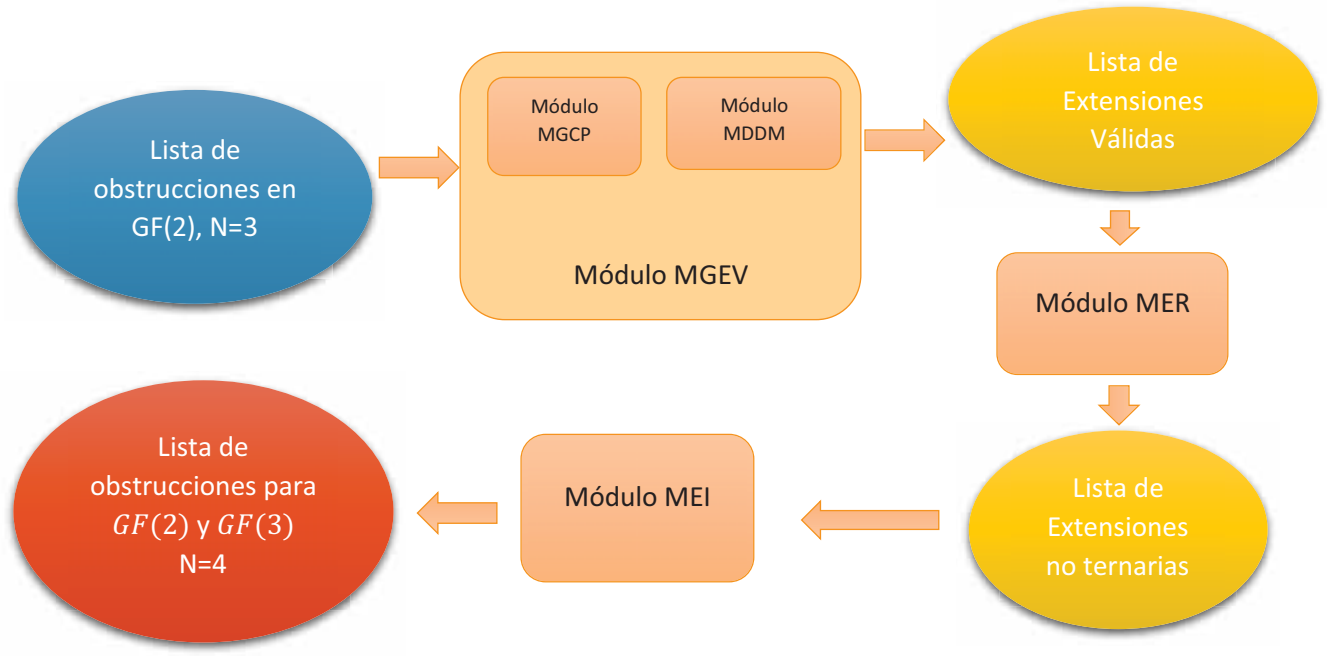


Imagen 3: Diagrama de módulos

Para generar las extensiones no binarias y no ternarias de tamaño 4 se requiere de las obstrucciones que caracterizan a delta-matroides binarios de A. Bouchet y A. Duchamp [3], concretamente las tres primeras: S_1, S_2 y S_3, esta lista de 3 delta-matroides es la entrada

del programa. Se requiere exactamente estos delta-matroides ya que al ser obstrucciones en el campo $GF(2)$ estas heredarán tal propiedad a cualquier extensión que se construya a partir de ellas.

El algoritmo está dividido en tres etapas:

- Generación de extensiones
- Descartar delta-matroides ternarios
- Depuración de isomorfos y delta-equivalentes

Cada etapa produce una salida (una lista de delta-matroides) que es tomada por la siguiente etapa, como es descrito en la Imagen 3: Diagrama de módulos. Se diseño un macro módulo para cada etapa del algoritmo: módulo MGEV (generación de extensiones), módulo MER (descartar delta-matroides ternarios) y MEI (depuración de isomorfos y delta-equivalentes). Adicionalmente el módulo MGEV está compuesto por dos módulos de menor complejidad el módulo MGCP y módulo MDDM, cuya funcionalidad será descrita más adelante. Ahora se describirá con mayor detalle los algoritmos específicos de cada etapa.

Generación de Extensiones

El módulo MGEV (módulo generador de extensiones válidas) es el encargado de solventar esta etapa, además para ello hace uso de dos sub módulos, MGCP (módulo genera conjunto potencia) y MDDM (módulo determina delta-matroide).

De manera intuitiva, sea $D = (V_k, F)$, un delta-matroide con $V_k = \{1,2,\dots,k\}$. En esta etapa el módulo MGEV que genera una extensión por borrado del delta-matroide D, añade un elemento $k+1$ al conjunto potencia $V_k + 1$ (que es proporcionado por el módulo MGCP) a la familia de subconjuntos $\mathcal{F}$ y determina si tras esta operación, $\mathcal{F}$ sigue siendo un delta-matroide (con ayuda del módulo MDDM), si es así, se añadirá a la lista de extensiones válidas. Realicemos un ejemplo para entenderlo mejor:

Tomamos a el delta-matroide:

$$S_1 = (V, \mathcal{F}) \mid V = \{1,2,3\},\ \mathcal{F} = \{\emptyset, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

Ahora el módulo MGCP se encargará de generar una lista con todos los subconjuntos de $P(V_4)$ que contienen a 4, es decir:

$$P(V_4) - P(V_3) = \{\{4\}, \{1,4\}, \{2,4\}, \{3,4\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}\}.$$

El siguiente paso es añadir un subconjunto X de $P(V_4) - P(V_3)$ a la familia de subconjuntos $\mathcal{F}$ en S_1:

Sea $X = \{\{4\}, \{1,2,4\}, \{2,3,4\}\}$ un subconjunto de $P(V_4) - P(V_3)$ entonces:

$$D_{new} = F \cup X$$

$$D_{new} = \{\emptyset, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\} \cup \{\{4\}, \{1,2,4\}, \{2,3,4\}\}$$

$$D_{new} = \{\emptyset, \{4\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{1,2,4\}, \{2,3,4\}\}$$

Finalmente, el módulo MDDM determina si la extensión resultante D_{new} es o no un delta-matroide, si D_{new} pasa la prueba, es añadido a la lista de extensiones válidas. El Módulo MGEV prueba con todos los subconjuntos de $P(V_4) - P(V_3)$ diferentes, además cabe mencionar que cada delta-matroide que aparezca en la lista de extensiones válidas, contiene como menor a $S_i, i = 1,2,3$; por lo que heredarán la propiedad de no representabilidad en $GF(2)$.

Descartar Delta-Matroides ternarios

En esta etapa el módulo MER recibe dos parámetros como entrada, la lista de extensiones válidas que genera el módulo MGEV (que es un dato dependiente) y la matriz de representación en $GF(3)$ del delta-matroide S_i (que es un dato independiente). La principal tarea de este módulo, es descartar los delta-matroides que están en la lista de extensiones válidas y que son ternarios, es decir, aquellos delta-matroides que poseen una representación matricial en $GF(3)$, ya que como todos estos delta-matroides contienen como menor a alguna obstrucción S_i no es posible que tengan una representación matricial con entradas en el campo $GF(2)$, pero puede existir una presentación matricial en $GF(3)$. Para lograr esto, el módulo MER generara una lista denominada "L*ista de extensiones ternarias*" con todos los delta-matroides ternarios en $V + 1$ que contienen como menor a S_i con $i = 1,2,3$. La estrategia para generar las extensiones ternarias es diferente a la de generar las extensiones no binarias presentes en la lista de extensiones válidas arrojada por el módulo MGEV, ya que, las extensiones ternarias poseen una representación matricial en el campo $GF(3)$:

Sean $D = (V, \mathcal{F})$ un delta-matroide y $A = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}$ una matriz simétrica con entradas en $GF(3)$ tal que A es la representación matricial de D. Podemos extender la matriz A con un vector θ con $V + 1$, θ_i tal que $\theta_i \in \{0,1,-1\}$.

Se construye la matriz B que es una extensión de A, como sigue:

Sea B extensión de $A \mid B = \begin{pmatrix} a & b & c & \boldsymbol{\theta_1} \\ b & d & e & \boldsymbol{\theta_2} \\ c & e & f & \boldsymbol{\theta_3} \\ \boldsymbol{\theta_1} & \boldsymbol{\theta_2} & \boldsymbol{\theta_3} & \boldsymbol{\theta_4} \end{pmatrix}$

Dado que B es una matriz simétrica con entradas en el campo $GF(3)$ existe un Delta-Matroide E que es ternario y posee como representación matricial a B y además $E\backslash V + 1 = D$, donde $V + 1$ es la nueva etiqueta. En otras palabras, el trabajo del módulo MER para generar todos los delta-matroides ternarios, se resume en la generación de todas las posibles combinaciones de las entradas del vector θ tomadas de $\{1,2,3,\dots,k,k+1\}$ y calcular el delta-matroide de la matriz resultante. De esta forma generaremos todos los delta-matroides ternarios que contiene como menor S_i.

Además, el módulo MER depura los delta-matroides repetidos de la lista de extensiones ternarias, finalmente busca los delta-matroides que están en la lista de extensiones válidas y que son extensiones ternarias, si son encontrados serán descartados, de modo que quedarán solo los delta-matroides que no son ternarios y dado que son extensiones de S_i tampoco son binarios.

Depuración de isomorfos y delta-equivalentes

Esta es la última etapa del algoritmo; consiste en depurar los delta-matroides que en un sentido combinatorio son equivalentes, aunque no sean exactamente el mismo, para ello primero descarta delta-matroides que sean isomorfos entre sí. Para resolver el problema de isomorfismo de dos delta-matroides, podemos probar todas las permutaciones de etiquetas en el conjunto base V de uno de ellos y aplicar esa permutación como una biyección, si resulta en un delta-matroide idéntico a el otro, entonces son isomorfos, si ninguna de las permutaciones posibles da una biyección válida entonces no son isomorfos. Sin embargo, dado que el algoritmo es de orden $O(N!)$, diseñé un algoritmo de *backtracking*, que contempla la frecuencia con la que aparecen los elementos de V en la familia de subconjuntos. Consideremos el siguiente ejemplo:

$D_1 = \{\emptyset, \{1\}, \{3\}, \{1,2\}, \{1,4\}, \{3,4\}, \{1,3,4\}\}$

$D_2 = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{2,4\}, \{3,4\}, \{2,3,4\}\}$

Vector de tablas de Frecuencia Tab

	1	2	3	4
Frecuencia	4	1	3	3

	1	2	3	4
Frecuencia	1	4	3	3

No tienen sentido las biyeciones que van de una etiqueta con frecuencia n en D_1 a una etiqueta con frecuencia m en D_2 con $m \neq n$, si pensamos en el algoritmo recursivo clásico para generar permutaciones, este no ejecutaría las llamadas recursivas que generen

biyecciones inválidas. Respecto al ejemplo, supongamos que queremos encontrar una biyección válida que para $D_1 \to D_2$:

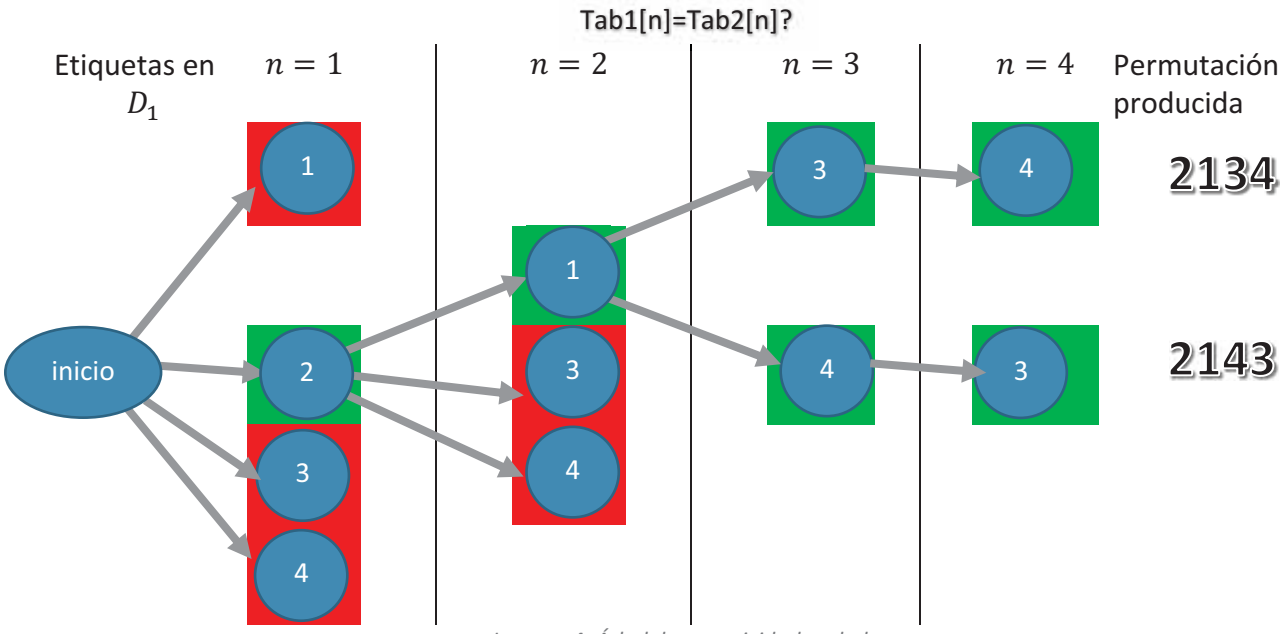


Imagen 4: Árbol de recursividad podado

Si tomamos la permutación producida 2143 y generamos su biyección φ:

$$\varphi: P \to Q = \begin{bmatrix} 1 & \to & 2 \\ 2 & \to & 1 \\ 3 & \to & 4 \\ 4 & \to & 3 \end{bmatrix} \text{ tal que } P, Q \text{ son familias de subconjuntos}$$

y la aplicamos a D_1 :

$$\varphi: D_1 \to Q = \{\emptyset, \{2\}, \{4\}, \{1,2\}, \{2,3\}, \{3,4\}, \{2,3,4\}\}.$$

Ahora nos fijamos en $D_2 = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{2,4\}, \{3,4\}, \{2,3,4\}\}$ notamos que $Q \neq D_2$, sin embargo, aún tenemos la permutación 2134, su función biyectiva φ:

$$\varphi: P \to Q = \begin{bmatrix} 1 & \to & 2 \\ 2 & \to & 1 \\ 3 & \to & 3 \\ 4 & \to & 4 \end{bmatrix} \text{ tal que } P, Q \text{ son familias de subconjuntos}$$

y la aplicamos a D_1 :

$$\varphi: D_1 \to Q = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{2,4\}, \{3,4\}, \{2,3,4\}\}.$$

Si nos fijamos en $D_2 = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{2,4\}, \{3,4\}, \{2,3,4\}\}$ notamos que $Q = D_2$ lo que implica que existe una función biyectiva φ tal que $\varphi: D_1 \to D_2 \therefore D_1 \, y \, D_2 \, son \, isomorfos.$

Como se puede apreciar en el ejemplo anterior, el algoritmo de *backtracking* diseñado, reduce enormemente las permutaciones que tienen que ser analizadas. Cabe mencionar que el módulo MEI solo genera las permutaciones si D_1 y D_2 tienen la misma huella.

Además, de depurar los isomorfos MEI también elimina los delta-matroides delta-equivalentes, esto calculando toda la clase de equivalencia de uno y comparándolos isomorfamente con el resto de la lista.

Análisis de complejidad

Como se ha mencionado con anterioridad, la mayoría de las operaciones crecen de manera exponencial, sin embargo, cabe mencionar que el cuello de botella en todo el proceso de la generación de extensiones se encuentra en la primera etapa "Generación de extensiones válidas", por tanto, tomaremos esta etapa como el referente para determinar la complejidad del algoritmo con base a el teorema de la suma[6] en complejidad de algoritmos. Recordemos que el módulo MGEV es el encargado de solventar esta etapa, y que este módulo prácticamente realiza 3 tareas: Generar subconjuntos, validar subconjuntos, y guardar resultados. Me he tomado la libertad de despreciar el tiempo de "guardar resultados" ya que esto significa una asignación y se resuelve en tiempo lineal, entonces analicemos las tareas "Generar subconjuntos" y "validar subconjuntos".

Primero que nada, consideremos la entrada de nuestro programa como N, donde N es el número de elementos del conjunto base de los delta-matroides que se van a construir.

- **Generar subconjuntos**: el problema de la generación de subconjuntos ha sido estudiado ampliamente y el problema tiene una complejidad de 2^n, los subconjuntos se van a generar sobre los elementos del conjunto potencia que para el caso $N = 4$ son 16, recordemos que el conjunto potencia son todos los subconjuntos del conjunto base es decir 2^N, entonces esto implica que tendremos que generar todos los subconjuntos de 2^N elementos por lo que la complejidad de este problema crece de manera $O(2^{2^N})$.
- **Validación de subconjuntos**: una vez que se tiene construido un subconjunto del conjunto potencia, para validar que sea un delta-matroide se tienen que tomar todas las parejas de subconjuntos factibles, eso significa tomar todas las combinaciones del número de subconjuntos factibles en dos, es decir: $C_{n,2} = \binom{n}{2}$, dado que tenemos que analizar el peor de los casos, tomaremos el caso en que están todos los subconjunto del conjunto potencia, es decir 2^N elementos, para formar las combinaciones de 2^N en dos, es decir: $C_{2^N,2} = \binom{2^N}{2}$. Ahora para cada combinación se calcula su diferencia simétrica, de acuerdo a la documentación disponible de C++[7], esto tiene una complejidad lineal $(2n)$ en términos de los tamaños de los

contenedores, para nuestro caso, considerando el peor de los casos como N. Tras estas consideraciones podemos inferir que la complejidad de validar subconjuntos es: $O\left(2N \cdot C_{2^N,2} = \binom{2^N}{2}\right)$.

Finalmente, por el teorema de la multiplicación podemos asumir que la complejidad de esta tarea es de:

$$O\left(2^{2^N+1} \cdot N \cdot \binom{2^N}{2}\right)$$

Descripción técnica de objetos y recursos

Para crear el programa PT-GeneracionDM se hizo uso del paradigma de programación orientado a objetos, esto para implementar el diseño de módulos de la propuesta de este proyecto de integración. Con ello se diseñaron los objetos DeltaMatroide y Huella (como atributo de DeltaMatroide):

Imagen 5:Diagrama de Clases DeltaMatroide y Huella

Es posible construir un delta-matroide de dos formas principales, una es introduciendo la familia de subconjuntos factibles del delta-matroide y otra mediante la matriz de representación, que calcula la familia de subconjuntos factibles mediante tal matriz. En

ambos casos se construye un objeto huella como atributo de delta-matroide con la familia de subconjuntos.

Puntualidades de los atributos y métodos de la clase DeltaMatroide:

- El objeto `Factible` es una definición de tipo (`typedf`) del objeto `vector<unsigned>`.
- Por tanto, las familias de subconjuntos son modeladas con vectores de vectores de enteros sin signo (`vector< vector<unsigned>>`).
- Matrix es también una definición de tipo (`typedf`) del objeto `vector<vector<int>>`.
- EL atributo conjunto_V se considera como la etiqueta V en la notación $[V]$ que significa $V = \{1,2, \ldots . , V\}$.
- El atributo `tabla_Frecuencias`, es un vector que se construye con las frecuencias de los elementos de $[conjunto\_V]$ en la `familia_subconjutnos`, este trabajo lo realiza el método `CalculaTablaFrecuencias()` y es invocado por el constructor una vez el atributo `familia_subconjutnos` está disponible.
- Los métodos `ContruyeBiyeccion()` y `AnalizaPermutaciones()` ayudan a el método `IsIsomorfo()` a determinar isomorfismo en delta-matroides.
- Los métodos `AnalizaSubMatrices()`, `IsFactible()` y `CalculaDeterminante()` son para calcular la familia de subconjuntos de un delta-matroide a partir de la matriz de representación, y son usados por el constructor que recibe un objeto Matrix, como parámetro.
- Los métodos `Twsting()` y `GetClaseEquivalencia()` generan delta-matroides delta-equivalentes de esta instancia.

Los macro módulos MGEV, MER y MEI heredan del objeto padre Modulo que posee los métodos `HuellaEnLista()` y `ObtenBloqueHuellas()` y que permiten hacer búsquedas en vectores de DeltaMatroides (`vector<DeltaMatroide>`) por bloques de huellas, ya que los tres módulos hacen búsquedas en listas estos métodos son heredados.

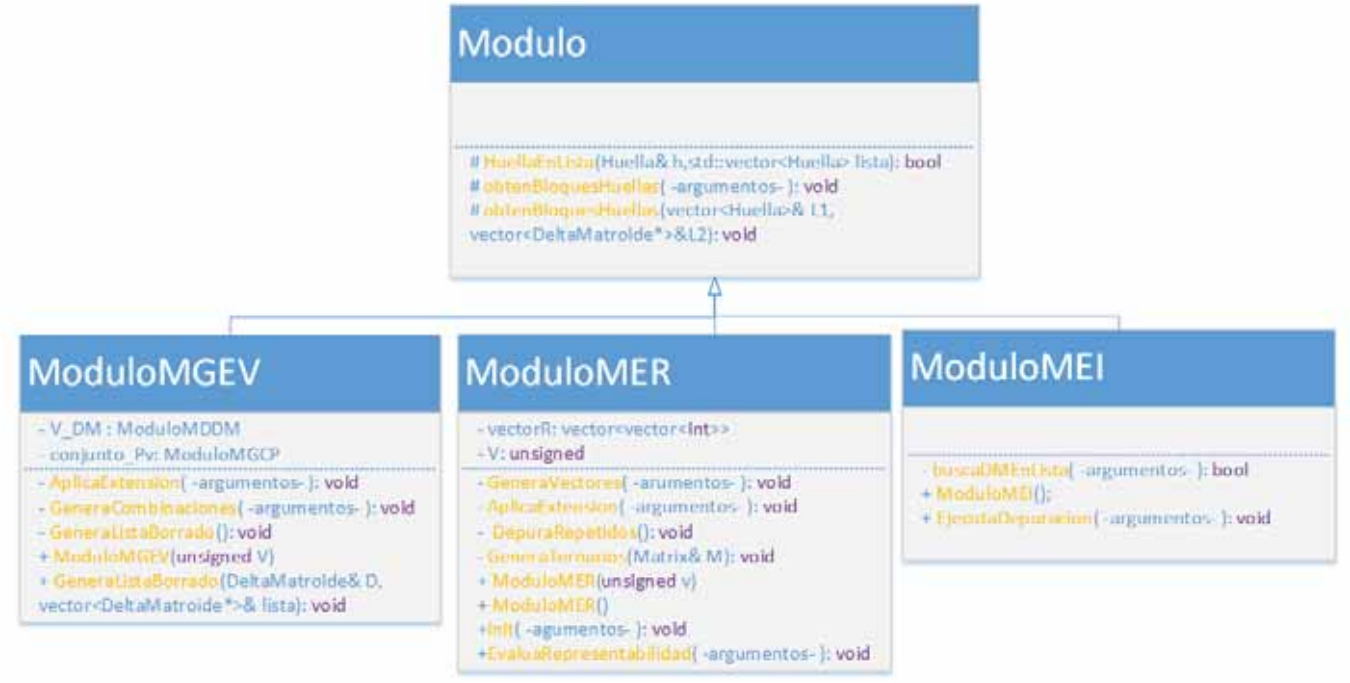


Imagen 6:Diagrama de clases de módulos

Puntualidades de los atributos y métodos de la clase MGEV:

- El atributo `conjunto_Pv` es una instancia de la clase MGCP que genera y provee el conjunto potencia.
- EL método `GeneraCombinaciones()` genera todos los subconjuntos de $P(V_4) - P(V_3)$ que el atributo `conjunto_Pv` proporciona.
- El método `AplicaExtension()` agrega uno de los subconjunto que el método `GeneraCombinaciones()` generó, a la familia de subconjuntos del objeto DeltaMatroide que se esté extendiendo.
- `GeneraListaBorrado()` recibe un objeto DeltaMatroide y genera todas sus posibles extensiones con ayuda de los métodos anteriores.
- El atributo `valida_DeltaMatroide` es una instancia de la clase MDDM y es usado por el método `GeneraListaBorrado()` para determinar que extensiones son extensiones válidas

Puntualidades de los atributos y métodos de la clase MER:

- El atributo `vectorR` guarda todas las combinaciones del vector extensión θ descrito en "Descartar Delta-Matroides ternarios" que serán para extender la matriz de representación. Son generadas por el método `GeneraVectores()`.
- El método `AplicaExtension()` inserta una combinación presente en el atributo `vectorR` como una nueva columna y un nuevo renglón en la matriz de representación.
- El método `GeneraTernarios()` hace uso del trabajo de los dos métodos anteriores para generar toda la *"Lista de extensiones ternarias"*.

- `DepuraRepetidos()` elimina todos los elementos repetidos de la *"Lista de extensiones ternarias".*
- Finalmente `EvaluaRepresentabilidad()` elimina todos los delta-matroides ternarios en la lista entrante gracias a la *"Lista de extensiones ternarias"*

Puntualidades de los atributos y métodos de la clase MEI:

Este módulo no posee atributos su principal objetivo es coordinar los métodos para determinar isomorfismo y delta-equivalencia, es decir las operaciones necesarias para depurar la información combinatoria repetida:

- `EjecutaDepuracion()` es el método principal, recibe como parámetros una vector de delta-matroides que será depurado y un vector vacío donde se guarda el resultado de la depuración.
- `BuscaDMEnLista()` es un método usado por `EjecutaDepuracion()` para tomar cada elemento del vector entrante y buscar isomorfos en el resto del vector, descartando todos aquellos que den la misma información combinatoria.

Parámetros de uso

El Programa PT-GeneracionDM está programado para efectuar dos modos de ejecución: secuencial y paralelo, además la entrada de este programa debe de estar en un archivo de texto plano, y al terminar su ejecución escribirá los delta-matroides ternarios generados en otro archivo de texto plano. Para interactuar y cargar estas configuraciones PT-GeneracionDM puede recibir parámetros que indiquen su modo de ejecución y configuración:

- **-P**: indica a PT-GeneracionDM que ejecute el modo paralelo, además el siguiente parámetro después de -P debe ser un entero n ($2 \leq n \leq$ número de núcleos de procesamiento disponibles) que indique el número de hilos que se lanzarán simultáneamente (por omisión se ejecuta el modo secuencial).
- **-S:** hace que PT-GeneracionDM ejecute la versión secuencial sin lanzar ningún hilo concurrente.
- **-I**: Indica a PT-GeneracionDM de que archivo va a tomar la entrada, por esto, tras el parámetro -I debe seguir una cadena de caracteres que representan el nombre del archivo de entrada (por default se intenta abrir el archivo "entrada").
- **-O**: permite introducir un nombre especifico para el archivo de salida con los delta-matroides ternarios, por ello tras el parámetro -O deberá seguir una cadena de caracteres (por default el archivo se llamará: "Ternarios.txt").
- **-D**: activa la depuración de minimales, un paso extra en el cálculo de las extensiones (por default está desactivado).

Modelo de concurrencia por datos

Dado que la entrada del algoritmo es una lista de delta-matroides y para cada lista se generan extensiones propias, este es un dato independiente y pueden ser operados simultáneamente. En el modo de ejecución -P se calculan las extensiones de n delta-matroides simultáneamente, dado que para el caso de extensiones con conjunto base de tamaño 4 solo hay 3 delta-matroides, se pueden lanzar 3 hilos independientes que calcularan las extensiones de S_1, S_2 y S_3:

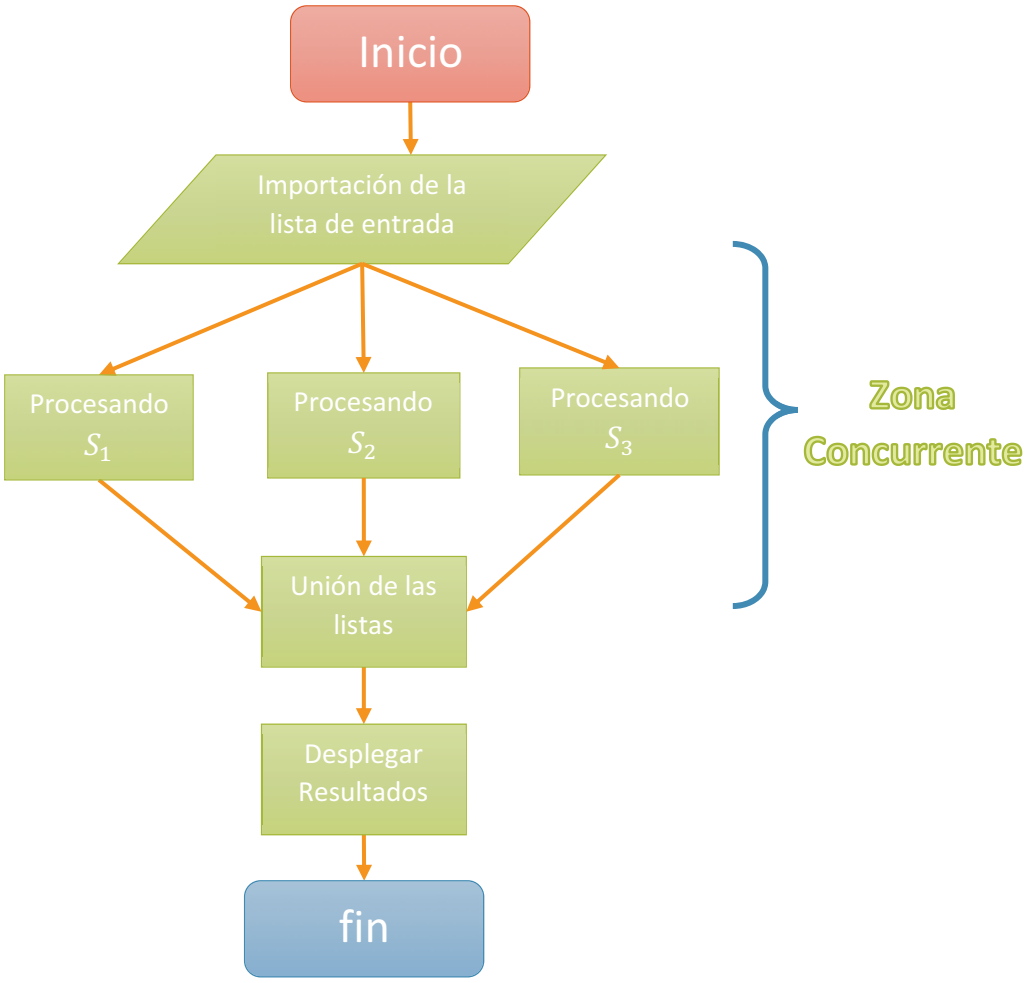


Imagen 7: Diagrama de flujo señalando la concurrencia

El programa PT-GeneracionDM esta parametrizado para lanzar tantos hilos como el usuario le indique, e itera sobre la lista entrante, aunque para generar las extensiones de tamaño cuatro solo sean tres elementos; en los siguientes niveles serán bastante más, en $N = 5$ la lista de entrada es de 63 delta-matroides (de acuerdo con los resultados de $N = 4$) y dados los recursos de hardware con los que contamos, se pueden lanzar hasta 8 hilos que operen concurrentemente los 63 delta-matroides

Resultados

Extensiones obtenidas

La siguiente lista de delta-matroides contiene los resultados de la etapa $n = 4$, que caracteriza a delta-matroides no binarios y no ternarios con conjunto base de 4 elementos; por tanto, son obstrucciones para representabilidad sobre los campos $GF(2)$ y $GF(3)$ con matrices simétricas:

$D1 = \{\emptyset, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

$D2 = \{\emptyset, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

$D3 = \{\emptyset, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

$D4 = \{\emptyset, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}\}$

$D5 = \{\emptyset, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

$D6 = \{\emptyset, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{2,3,4\}\}$

$D7 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}\}$

$D8 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,4\}, \{1,3,4\}, \{1,2,3,4\}\}$

$D9 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,4\}, \{1,2,3,4\}\}$

$D10 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3,4\}\}$

$D11 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

$D12 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}$

$D13 = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,2,3,4\}\}$

$D14 = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,3,4\}\}$

$D15 = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{1,2,3\}, \{1,2,3,4\}\}$

$D16 = \{\emptyset, \{4\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$

Para la etapa $N = 5$ se obtuvieron un total de 2,382 extensiones que están incluidas en el archivo de texto plano ResultadosEtapa5. Cabe mencionar que **estas extensiones no son obstrucciones ya que no son minimales**, estas podrían contener como menor a alguna de las obstrucciones D_i enlistas anteriormente. Sin embargo, son resultados útiles que ayudan al avance en la caracterización de representabilidad.

Informe de rendimiento Ganado

Con los recursos computacionales con los que contamos logramos resolver el problema para el caso $N = 4$ y el caso $N = 5$.

Para el caso $N = 4$ el programa paralelo resultó ser **2.12** veces más eficiente que la versión secuencial, los cálculos en la versión secuencial tardaron **777.2** milisegundos en

promedio para concluir, mientras que la versión paralela le tomo **367.4** milisegundos en promedio. A continuación, se muestra una tabla con los tiempos de 30 ejecuciones:

Ejecuciones de PT-GeneracionDM con $N = 4$			
Numero de Ejecución	Tiempo en Milisegundos		Eficiencia $E = \frac{T(S)}{T(P)}$
	Secuencial	Paralelo (3 hilos)	
1	773	367	2.11
2	771	367	2.10
3	768	367	2.09
4	784	366	2.14
5	772	366	2.11
6	787	369	2.13
7	776	369	2.10
8	774	369	2.10
9	770	370	2.08
10	774	367	2.11
11	788	369	2.14
12	786	366	2.15
13	786	366	2.15
14	778	369	2.11
15	777	366	2.12
16	777	366	2.12
17	771	366	2.11
18	781	366	2.13
19	776	368	2.11
20	788	366	2.15
21	773	367	2.11
22	772	371	2.08
23	777	366	2.12
24	776	367	2.11
25	778	368	2.11
26	782	367	2.13
27	780	369	2.11
28	775	366	2.12
29	783	370	2.12
30	775	366	2.12
Media	**777.6**	**367.4**	**2.12**

Tabla 1: tabla con los resultados de la evaluación de rendimiento $N = 4$

En la siguiente gráfica podemos ver con mayor detalle los tiempos de ejecución y el rendimiento ganado del programa paralelo frente al secuencial, como se puede apreciar el tiempo de ejecución medido en milisegundos de la versión secuencial (presente en azul) casi duplica el tiempo de ejecución de la versión paralela (representado en verde) a lo largo de las 30 ejecuciones que realizamos para evaluar el rendimiento ganado.

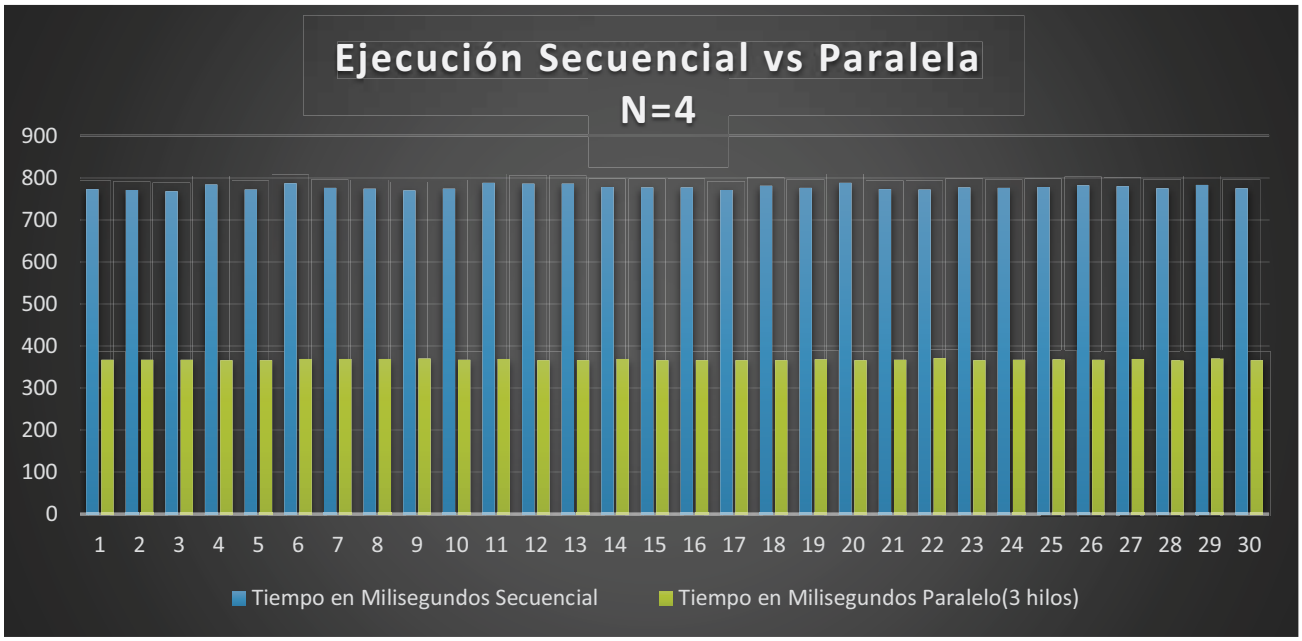


Imagen 8:grafica con los tiempos de ejecución $N = 4$

Ahora para el caso de $N = 5$, notamos un incremento en el rendimiento ganado, en esta etapa cada ejecución tarda una media de 8780208.2 milisegundos es decir 2.4 horas en la versión secuencial mientras que la versión paralela tarda una media de 2692696.7 es decir 44.8 minutos, para esta etapa realizamos solo 10 ejecuciones como se puede ver en la siguiente tabla:

Ejecuciones de PT-GeneracionDM con $N = 5$			
Numero de ejecución	Secuencial	Paralelo (8 hilos)	Eficiencia $E = \frac{T(S)}{T(P)}$
1	8771291	2774436	3.16
2	8746216	2759891	3.17
3	8772029	2729756	3.21
4	8781897	2613404	3.36
5	8801699	2770456	3.18
6	8774414	2763174	3.18
7	8797733	2664971	3.30
8	8770465	2647441	3.31

9	8890439	2615433	3.40
10	8695899	2588005	3.36
Media	**8780208.2**	**2692696.7**	**3.26**

Tabla 2: tabla con los resultados de la evaluación de rendimiento $N = 5$

Al igual que en la etapa $N = 4$, en la siguiente imagen se muestra una gráfica con las diez ejecuciones paralelas y secuenciales en las que se puede apreciar la cantidad de tiempo en milisegundos consumidos por las versiones secuenciales (en azul) y paralela (en verde).

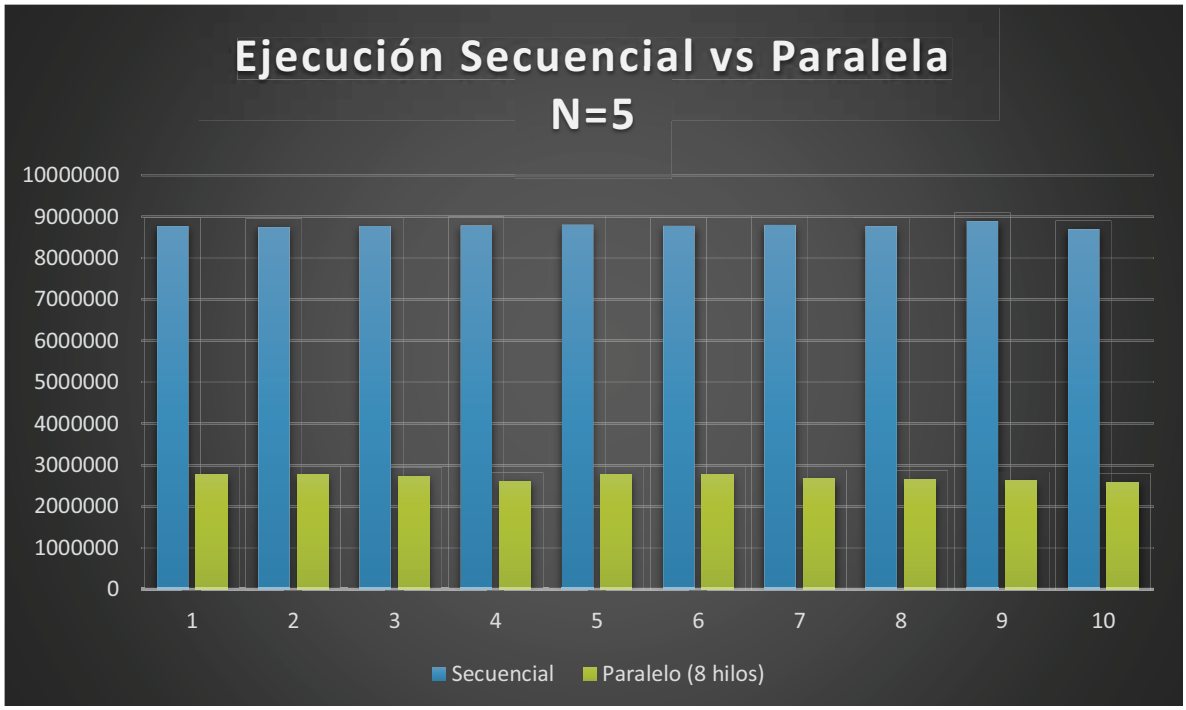


Imagen 9: grafica con los tiempos de ejecución N=5

Note como el rendimiento ganado de la versión paralela frente a la secuencial se ha incrementado en esta etapa, en parte fue gracias a que en esta etapa existen más datos para procesar lo que permite tener más hilos ejecutando distintos subconjuntos de datos.

Modelo de *Pipeline*

Dada la naturaleza del algoritmo planteado para generar las extensiones, que se encuentra segmentado en etapas, se diseñó un modelo de paralelismo *pipeline* a la par que el modelo de concurrencia por datos, originalmente se tenía planeado enfrentar ambos modelos de paralelismo para el informe de rendimiento ganado, sin embargo, tras las primeras pruebas el modelo de *pipeline* resultó ser muy deficiente, casi un 50 por ciento más lento que la versión secuencial. Durante el análisis de complejidad que se realizó en este proyecto se determinó que la etapa de "Generación de extensiones válidas" representaba un cuello de botella importante. Fue entonces que entendimos porque el

modelo de *pipeline* tiene un mal rendimiento, ya que ese cuello de botella obstruye (analógicamente claro) al flujo de datos en el modelo de pipeline, provocando que las unidades encargadas de las etapas después del cuello de botella estén en receso. Tras estos primeros resultados, decidimos no seguir trabajando este modelo.

Conclusiones

A lo largo de la realización de este proyecto se me han presentado distintos retos que me han hecho poner a prueba mis conocimientos y habilidades, los cuales al mismo tiempo me han servido como una retroalimentación y una oportunidad para aprender mas y refinar mis capacidades. Finalmente, los resultados arrojados por este proyecto, por un lado, vienen a solventar sobradamente los requerimientos planteados en los objetivos, con lo cual logramos generar una lista de delta-matroides que son obstrucciones para representabilidad en los campos $GF(2)$ y $GF(3)$, mediante un modelo de cómputo paralelo que constó de la implementación de cinco módulos (MDDM, MGCP, MGEV, MER y MEI) que son un conjunto de algoritmos para hacer cálculos con delta-matroides mediante un diseño segmentado en etapas, que permitió hacer un uso más eficiente de recursos aprovechando las bondades de paralelismo de las arquitecturas de computadoras modernas. Por otro lado, respecto al significado simbólico o valor de los resultados arrojados de este proyecto en términos matemáticos y computacionales me gustaría acotar las siguientes conclusiones:

- Las 16 extensiones coinciden con el trabajo de la Dra. G. Rodríguez verificando por una parte sus resultados y validando los algoritmos empleados en este proyecto.
- De 16 extensiones siete de ellas contienen como menor al delta-matroide DT [8], que tiene una representación sobre $GF(3)$ con una matriz simétrica. Dependiendo del enfoque; estas podrían no ser obstrucciones al no ser minimales.
- Dados los resultados de rendimiento, podemos decir con certeza que calcular los delta-matroides de conjunto base de tamaño 6 es posible con más recursos computacionales.
- Si es posible calcular las extensiones de conjunto base $N = 7$, podría resolverse por completo el problema de $GF(2)$ y $GF(3)$ representabilidad con matrices simétricas.
- Tras el análisis de complejidad que se realizó en este proyecto podemos concluir que resolver el caso $N = 7$, significa hacer 2^{64} validaciones de delta-matroides, lo que es inabordable, al menos con una computadora promedio, quizás con la ayuda de súper cómputo sea posible.
- Tras las evidencias presentadas; es imposible demostrar la conjetura de que no existen obstrucciones con conjunto base mayor que $N = 7$, **con medios computacionales**. Para tatar de verificar la conjetura se tendrían que calcular las extensiones de $N = 8$ y esperar que este paso no arrojará ninguna extensión, pero esto es inabordable con la tecnología actual.

- Podemos asumir que, dado que el rendimiento del modelo de paralelismo *pipeline* resulto ser deficiente, el mejor camino para este problema son los modelos basado en el paralelismo de datos dada la naturaleza de independencia.

Los resultados del rendimiento ganado fueron aceptables, se logró una gran eficiencia, sin embargo, tomando en cuenta los análisis presentes en este trabajo, todo apunta a que es posible un rendimiento incluso mayor, sobre todo si se usan tecnologías de arquitecturas diferentes como las SIMD, podemos concluir que puede implementarse un mejor modelo de paralelismo para avanzar un siguiente paso en la caracterización de delta-matroides no binarios y no ternarios.

Referencias bibliográficas:

[1] H. Whitney, "On the Abstract Properties of Linear Dependence", *Am. J. Math.*, vol. 57, núm. 3, pp. 509–533, 1935.

[2] A. Bouchet, "Greedy algorithm and symmetric matroids", *Math. Program.*, vol. 38, núm. 2, pp. 147–159, jun. 1987.

[3] A. Bouchet, "Representability of Δ-matroids over GF(2)", *Linear Algebra Its Appl. - LINEAR ALGEBRA APPL*, vol. 146, pp. 67–78, feb. 1991.

[4] D. A. Patterson y J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Elsevier, 2011.

[5] D. Frausto E. Valenzuela y O. Saucedo, "ProyectoDelta: paquete computacional para la investigación en Delta-Matroides". UAM, 2017.

[6] S. Baase y A. V. Gelder, *Algoritmos computacionales: introducción al análisis y diseño*. 2002.

[7] "set_symmetric_difference - C++ Reference". [En línea]. Disponible en: http://www.cplusplus.com/reference/algorithm/set_symmetric_difference/. [Consultado: 22-mar-2018].

[8] G. Rodríguez, "GF(3)-Representabilidad de Delta-Matroides", CINVESTAV, 1999.