International Journal of Information Science and Technological Applications-UAS
IJISTA
ISSN (en trámite)
https://revistas.uas.edu.mx/index.php/IJISTA

Mayo 2025
Vol.1.
Número:1

Investigación y diseño de software basado en la teoría de matroides orientadas

Designing software based on oriented matroid Theory.

Emmanuel Roberto Estrada Aguayo[1], Carlos Francisco Montoya Urzúa[2]

[1]Facultad de Informática / Universidad Autónoma de Sinaloa, México.
[2]Departamento de Sistemas y Computación, Instituto Tecnológico de Ciudad Guzmán, México.

Recibido: Enero 2025, **Aceptado:** Marzo 2025, **Publicado**: Mayo 2025

Resumen:

Este trabajo presenta el diseño de un software en C++ orientado al análisis de matroides orientadas, estructura fundamental de la geometría combinatoria que extiende el concepto de independencia lineal. Se abordan nociones clave de la teoría; como base, circuito, matroide uniforme y chirotope (χ), que describe la orientación de la matroide. El desarrollo se basa en una fórmula propuesta por el Dr. J. A. Nieto para construir una matriz que represente a una matroide orientada a partir de n elementos y su correspondiente chirotope. La implementación computacional genera matrices cuadradas $n \times n$, pero el análisis muestra que, en general, estas no satisfacen las condiciones necesarias de dependencia lineal esperadas en matroides de rango $r < n$, pues presentan determinantes distintos de cero, lo cual contradice su validez como representaciones sobre $\mathbb{R}$. Como resultado emergente, se propone que la submatriz conformada por las primeras r filas de la matriz generada podría ser una representación válida de la matroide, hipótesis apoyada por ejemplos computacionales. Se concluye que el software constituye un avance funcional en el estudio computacional de matroides orientadas, y se identifican líneas futuras para validar formalmente la fórmula base, optimizar su complejidad algorítmica y caracterizar la representabilidad computacional de este tipo de estructuras.

Palabras Clave: matroide, matriz, chirotope, software.

Abstract:

This work presents the design of C++ software aimed at the analysis of oriented matroids, a fundamental structure in combinatorial geometry that extends the concept of linear independence. Key notions of the theory are addressed, such as bases, circuits, uniform matroids, and the chirotope (χ), which describes the orientation of the matroid. The development is based on a formula proposed by Dr. J. A. Nieto to construct a matrix representing an oriented matroid from n elements and its corresponding chirotope. This computational implementation generates square matrices of size $n \times n$, but analysis shows that, in general, these do not satisfy the necessary conditions of linear dependence expected in matroids of rank $r < n$, as they yield nonzero determinants, contradicting their validity as representations over $\mathbb{R}$. As an emergent result, it is proposed that the submatrix formed by the first r rows of the generated matrix may constitute a valid representation of the matroid, a hypothesis supported by computational examples. It is concluded that the software represents a functional advance in the computational study of oriented matroids, and future lines of research are identified to formally validate the base formula, optimize its algorithmic complexity, and characterize the computational representability of such structures.
Index Terms: matroid, matrix, chirotope, software.

Estrada Aguayo, E.R., Montoya Urzúa, C.F. (2025). Investigación y diseño de software basado en la teoría de matroides orientadas. *International Journal of Information Science and Technological Applications-UAS IJISTA,* 1(1), 32-45. Doi (en trámite)

**IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)**

1. Introducción

La Teoría de Matroides es una rama de la geometría combinatoria que se basa en la estructura conocida como matroide, que ofrece una generalización del concepto de independencia lineal entre vectores. Por tal motivo, esta encuentra su valor matemático dentro del algebra lineal, pero también dentro de la teoría de grafos; donde permite estudiar los conjuntos de caminos acíclicos (bosques) y cíclicos de un grafo.

Dentro de las Ciencias de la Computación, los matroides han sido utilizados para generar algoritmos de solución a varios problemas de optimización; y en el caso de los matroides orientados, han destacado por su aplicación dentro del campo de la geometría computacional [2].

La generación de representaciones matriciales de matroides orientadas constituye un problema relevante tanto desde el punto de vista teórico como computacional.

Las matroides orientadas incorporan a la teoría de matroides axiomas relacionados con la orientación de conjuntos de vectores. Estas ofrecen un marco potente para modelar estructuras en geometría combinatoria, optimización y teoría de arreglos.

A pesar de su riqueza conceptual, el tratamiento computacional de estas estructuras es aún escaso. Existen muy pocas herramientas de software que permitan construir explícitamente matrices que representen una matroide dada, y son aún menos aquellas que operan sobre matroides orientadas.

Esta escasez limita el acceso a experimentación computacional, simulación de ejemplos complejos y verificación empírica de conjeturas, lo cual es crucial tanto en contextos educativos como en la investigación avanzada.

Por tal motivo, el presente trabajo tiene como objetivo principal el diseño de software que permita calcular la matriz representación de una matroide orientada, a partir de datos que caractericen la matroide seleccionada por el usuario; tales como rango, dimensión y el mapeo de la función chirotope.

Fundamentándose en este objetivo general, se establecen los siguientes objetivos específicos aplicables al proyecto son:

1. Desarrollar un prototipo funcional en C++ capaz de generar representaciones matriciales de matroides orientadas a partir de un chirotope dado.
2. Validar computacionalmente la consistencia de las matrices generadas, mediante la comparación con propiedades teóricas esperadas de matroides de bajo rango.
3. Analizar las limitaciones de la fórmula implementada y proponer hipótesis para su corrección o mejora, sustentadas en los resultados experimentales obtenidos.
4. Evaluar el potencial educativo y de investigación del software desarrollado mediante ejemplos aplicados en el contexto de la teoría de matroides orientadas.

Se espera que este proyecto sea de gran utilidad para el estudio de la teoría de matroides y su aplicación en otros campos.

En años recientes, la teoría de matroides ha encontrado nuevas aplicaciones dentro de la cosmología, el análisis y la edición genética, y el desarrollo de la Inteligencia Artificial [6]. Por tanto, software que ayude a entender los conceptos básicos de la teoría de matroides, será altamente beneficioso para estudiantes y profesionistas de estas ramas del conocimiento.

La mayoría de los algoritmos sobre matroides suelen ser computacionalmente costosos (Big-O grande), al basarse principalmente en realizar operaciones durante varios recorridos sobre una misma matriz. Por tal motivo el presente software fue codificado en C++, que es un lenguaje que puede parecer limitado al usarse en investigaciones científicas, pero es reconocido por su velocidad y menor consumo de recursos comparado a los lenguajes de programación de alto nivel. Además, C++ es uno de los lenguajes de programación que más facilita la creación de librerías externas para usar sus funciones en software escrito en un lenguaje distinto (como Java o Python).

Por último, se reconoce que la idea original de este código fue creada por el Licenciado en Física Daniel Aldana Virgen. Sobre la base de su trabajo, el sistema fue actualizado, optimizado, y en algunos aspectos corregido, para alcanzar un uso más general y mayor amabilidad con el usuario.

2. Trabajos Relacionados

2.1 Antecedentes de proyectos de software relacionados a la teoría de matroides.

En décadas recientes se han generado un puñado de herramientas de software orientadas para el estudio de matroides, aunque pocas se enfocan específicamente en matroides orientadas.

Se procede a enumerar las herramientas de software relacionadas más destacadas:

- **Polymake**, una plataforma extensible desarrollada en C++ y Perl; está orientada principalmente a la geometría de poliedros. Esta ofrece un módulo para

IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)

el análisis estructural de matroides que permite calcular bases, circuitos y polinomios de Tutte; pero no contempla representaciones orientadas a partir de chirotope [9].

- **Oid** es un sistema que ofrece una interfaz interactiva para experimentar con matroides, aunque su enfoque se restringe a representaciones sobre campos finitos. Actualmente no ofrece ninguna aproximación a matroides orientadas [10].
- **SageMath**, un sistema de álgebra computacional basado en Python cuenta con una sólida implementación de matroides, permitiendo su construcción y análisis desde matrices o funciones de rango. Sin embargo, su soporte para matroides orientadas es limitado, y no ofrece una funcionalidad específica para derivar representaciones matriciales de una matroide [11].
- **Maple** es un sistema de algebra computacional ampliamente usado. En su versión del año 2024, ha incorporado módulos dedicados al estudio de matroides, pero mantiene un enfoque general y no aborda aspectos orientados [12].

Finalmente, existen propuestas orientadas a aplicaciones específicas, como la planificación de caminos en robótica, que emplean matroides orientadas en su modelado, pero sin ofrecer herramientas reutilizables [13].

Estos antecedentes reflejan que, aunque existen ya diversas herramientas para el estudio de matroides, se identifica una carencia de software específico que facilite el análisis de matroides orientadas, especialmente en lo que respecta a su representación matricial y la validación de propiedades asociadas.

El desarrollo de herramientas especializadas en este ámbito podría llenar este vacío y proporcionar un recurso valioso para investigadores y educadores interesados en la teoría de matroides orientadas.

Una vez abordado el conocimiento de las herramientas de software que anteceden al presente proyecto, se presentan los conceptos básicos de la teoría de matroides presentados en [1],[2], y [3]. Sobre estos conceptos se sustenta la investigación.

2.2 Matroide

Una matroide es una estructura de la geometría combinatoria que extiende el concepto de independencia lineal. Su definición técnica es la siguiente:

Definición 1. *[2] Una matroide M es un par $(E, \mathcal{B})$, donde E es un conjunto finito no vacío y $\mathcal{B}$ es una colección no vacía de subconjuntos de E (llamados bases) que satisfacen las siguientes propiedades:*

- $\emptyset \in \mathcal{B}$
- *Ninguna base contiene propiamente a otra base;*
- *Si B_1 y B_2 son bases y si b es cualquier elemento de B_1, entonces existe un elemento g de B_2 con la propiedad de que $(B_1 - \{b\}) \cup \{g\}$ también es una base.*

El conjunto E se nombra conjunto soporte de la matroide.

El conjunto $\mathcal{B}$ se nombra conjunto hereditario.

Todo conjunto que sea elemento de $\mathcal{B}$, y cualquier subconjunto de algún conjunto elemento de $\mathcal{B}$, se denomina conjunto linealmente independiente.

2.3 Circuitos de una matroide

Definición 2. *[1][2] Un circuito de una matroide es un conjunto linealmente dependiente con la cardinalidad mínima posible.*

El conjunto de circuitos de una matroide suele denotarse como $\mathcal{C}$, y posee las siguientes propiedades:

1. *$\emptyset \notin \mathcal{C}$. El vacío es considerado un conjunto linealmente independiente, por tanto, no es un circuito.*
2. *Ningún circuito de una matroide contiende a otro circuito. Por tanto, si $C_1, C_2 \in \mathcal{C}$, y $C_1 \subseteq C_2$, entonces $C_1 = C_2$.*
3. *Si $C_1 \in \mathcal{C}$, entonces $C_1 \notin \mathcal{B}$ y $(C_1 - x) \subseteq b, b \in \mathcal{B}$ para todo $x \in C_1$. O sea que un circuito nunca es una base de la matroide, pero al quitarle cualquier elemento al circuito, este es un conjunto linealmente independiente (una base o un subconjunto de una base).*
4. *Si $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$ y $e \in C_1 \cap C_2$, entonces existe un $C_3 \in \mathcal{C}$ tal que $C_3 \subseteq (C_1 \cap C_2) - e$.*

2.4 Matroide Uniforme $U_{k,n}$

Definición 3. [1][2][3] *Una matroide uniforme (denotada como $U_{k,n}$) con $k < n$, es una matroide sobre un conjunto soporte E de n elementos, donde cualquier subconjunto de E con k elementos distintos es una base de la matroide (k es el rango de la matroide).*

IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)

$$U_{k,n} = (E, \mathcal{B}) \quad (1)$$

$$E = 1,2,3,\dots,n \quad (2)$$

$$Sea\ b \subseteq E,\ entonces\ |b| = k \Leftrightarrow b \in \mathcal{B} \quad (3)$$

Por ejemplo, la matroide $U_{2,4}$ es conformada por los siguientes conjuntos:

$$U_{2,4} = (E, B) \quad (4)$$

$$E = \{1,2,3,4\} \quad (5)$$

$$\mathcal{B} = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\} \quad (6)$$

2.5 Matroide Orientada

Definición 4. [1][3] *Una matroide orientada M es un tipo de matroide definida por un par $M = (E, \chi)$, donde E es el conjunto soporte de la matroide (un conjunto finito de elementos); y χ (denominado chirotope, chirotopo, quirotopo o tipo de orden) es una función que hace un mapeo sobre las permutaciones de cada k-tupla de elementos pertenecientes a E, al conjunto $\{-1,0,1\}$, muchas veces expresado como $\{-,0,+\}$, que satisface la definición siguiente.*

2.5.1 Chirotope

Un chirotope (también denominado en español como quirotopo o chirotopo) es una estructura algebraica asociada con la teoría de matroides orientados. Proporciona una manera de describir la orientación de los elementos en una matroide, lo que permite extender varias propiedades y resultados de la teoría de matroides a una versión orientada. El concepto de chirotope se originó en la teoría de matroides y geometría combinatoria.

Definición 5. [3] *Dado un conjunto finito E de n elementos, un chirotope es una función que asigna a cada k-tupla ordenada de elementos de E un valor de $\{+1,-1,0\}$, de manera que se cumplen ciertas propiedades que reflejan la orientación de los elementos. Para matroides completos, se asume que $k = n$.*

Formalmente, un chirotope de rango k sobre un conjunto finito $\mathbb{E}$ es una función

$$\chi: \mathbb{E}^k \to \{+,-,0\} \quad (7)$$

que satisface las siguientes propiedades:

1. **Toma una tupla de tantos elementos de $\mathbb{E}$ como el rango de la matroide**.
2. **No es constante** 0. Dicho valor aparece exclusivamente cuando la tupla de elementos NO es una base de la matroide a la que describe.
3. **Antisimetría o alternancia**: Para cualquier permutación σ de $\{1,2,\dots,k\}$,

$$\chi\left(e_{\sigma(1)}, e_{\sigma(2)}, \dots, e_{\sigma(k)}\right) = sign(\sigma) \cdot \chi(e_1, e_2, \dots, e_k) \quad (8)$$

   donde $sign(\sigma)$ es la signatura de la permutación σ. Esto es similar al signo del determinante de una matriz, que alterna entre las distintas formas de ordenas los vectores que conforman dicha matriz.

4. **Propiedad de Graßman-Plücker (o propiedad de intercambio)**: Para cualquier conjunto de $k+1$, elementos $e_1, e_2, \dots, e_{k+1}$ en E, se cumple

$$\sum_{i=1}^{k+1} (-1)^i \chi(e_1, \dots, \hat{e_i}, \dots, e_{k+1}) \chi(e_i, f_1, \dots, f_{k-1}) = 0 \quad (9)$$

   donde $\hat{e_i}$ denota que el elemento e_i se omite.

2.6 Representación matricial de una matroide

Definición 6. [1][2] *La **representación matricial de una matroide** es una forma de representar la matroide mediante una matriz donde cada columna de esta está asociada a un elemento del conjunto soporte de la matroide E. Así pues, la columna 1 representa al elemento 1 del conjunto E de la matroide, la columna 2 representa al elemento 2 y así sucesivamente.*

Esta matriz codifica la información sobre las dependencias lineales y las relaciones entre los subconjuntos del conjunto E. Para todo subconjunto e del conjunto soporte E, si sus elementos son linealmente independientes, entonces los vectores columna de la matriz asociados a los elementos de e también son linealmente independientes. Si los elementos de e son linealmente dependientes, también lo es el conjunto de vectores columna asociados a sus elementos.

Cabe aclarar que las representaciones matriciales de una matroide dada son especificas a un determinado campo de Galois. Ya que dos vectores pueden ser linealmente independientes en un campo de Galois especifico, y no serlo en otros.

Por norma general, si un elemento de E no es parte de ninguna base de la matroide dada, este se representa con el vector nulo o $\vec{0}$.

**IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)**

No todas las matroides pueden asociarse a una matriz. Aquellas que si tienen una matriz que las caracteriza son nombradas matroides representables [1][2][14].

3. Metodología.

Este estudio se desarrolló bajo un enfoque cualitativo con un diseño metodológico de tipo *investigación-acción tecnológica*, orientado a proponer y evaluar una herramienta computacional que facilite el análisis estructural de matroides orientadas, al tiempo que se reflexiona críticamente sobre sus fundamentos teóricos. Este enfoque es adecuado para investigaciones donde el desarrollo de soluciones tecnológicas se vincula estrechamente con el avance del conocimiento disciplinar.

Se emplearon tres técnicas principales: análisis documental, observación sistemática y pruebas computacionales controladas. El análisis documental consistió en la revisión de literatura especializada sobre matroides y matroides orientadas, con énfasis en su representación mediante matrices y las propiedades algebraicas asociadas. La observación sistemática se llevó a cabo durante el proceso iterativo de desarrollo del software, registrando comportamientos relevantes, limitaciones técnicas y respuestas del sistema. Las pruebas computacionales permitieron aplicar el algoritmo a distintos ejemplos para contrastar su funcionamiento con resultados teóricos conocidos.

La muestra del estudio estuvo conformada por una colección de instancias de matroides orientadas, principalmente de bajo y mediano rango ($r \leq 4$), extraídas de fuentes académicas o generadas *ad hoc*, que sirvieron como casos de prueba para evaluar la validez de las representaciones obtenidas.

El instrumento principal fue un prototipo de software desarrollado en C++ durante la investigación, complementado con una bitácora de ejecución que permitió documentar sistemáticamente los resultados obtenidos, anomalías encontradas y ajustes realizados. Como se mencionó en la introducción, la elección de C++ se basó en su eficiencia en la ejecución de algoritmos intensivos en operaciones matriciales.

Para asegurar la validez del estudio, se aplicó triangulación metodológica: los resultados computacionales se contrastaron con teorías existentes, y se sometieron a revisión por parte de especialistas en matemáticas aplicadas y computación. Esta triangulación permitió fortalecer la confiabilidad de los hallazgos y establecer fundamentos sólidos para futuras mejoras del sistema.

En los apartados subsecuentes se incluyen aspectos teóricos relevantes para el desarrollo del sistema de software que fueron encontrados por medio de la investigación bibliográfica o por medio del análisis de los resultados guardados en la bitácora de pruebas.

3.1 Cálculo de la representación matricial de una matroide orientada

Antes de iniciar esta sección, cabe recalcar que en [1] se aclara que no todas las matroides son representables por una matriz sobre un determinado campo $\mathbb{F}$.

La fórmula propuesta en [3] para el cálculo de una matriz cuadrada $A = \{a_{i,j}\}$, que funcione de representación a una matroide orientada es (en notación tensorial):

$$a_{i,j} = \chi_{ikl}\chi_{jkl} \qquad (10)$$

Se puede generalizar por la siguiente fórmula para una matroide orientada M, con un chirotope χ que tome r elementos del conjunto soporte de la matroide E, es que toma en cuenta todas las permutaciones con repeticion de r elementos de E:

$$a_{i,j} = \frac{1}{MCD\{A\}} \sum_{x_1=1}^{n} \sum_{x_2=1}^{n} \sum_{x_3=1}^{n} \cdots \sum_{x_{r-1}=1}^{n} \chi\,(i, x_1, x_2, x_3, \ldots, x_{r-1}) * \chi(j, x_1, x_2, x_3, \ldots, x_{r-1}) \qquad (11)$$

Sobre la fórmula anterior se sustenta la programación del software.

Apréciese que se usan $r-1$ sumatorias e índices, ya que el primer indice en la función chirotope siempre corresponderá, primero a la fila del elemento en la matriz y segundo a su columna.

Por ejemplo, el cálculo de la matriz $a_{i,j}$ para una matroide $U_{4,n}$, con un chirotope $\chi: E^4 \to \{-1,0,1\}$, quedarí a como:

$$a_{i,j} = \sum_{k=1}^{n} \sum_{l=1}^{n} \sum_{m=1}^{n} \chi\,(i,k,l,m) * \chi(j,k,l,m) \qquad (12)$$

Esta última fórmula puede escribirse para revisar únicamente las permutaciones diferentes de cero:

$$a_{i,j} = \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{n} \sum_{\substack{l=1 \\ l\neq i \\ l\neq j \\ l\neq k}}^{n} \sum_{\substack{m=1 \\ m\neq i \\ m\neq j \\ m\neq k \\ m\neq l}}^{n} \chi(i,k,l,m) * \chi(j,k,l,m) \quad (13)$$

3.2 Propiedades particulares de la matriz A en una matroide orientada

3.2.1 En una matroide orientada $U_{r,n}$, la diagonal principal $a_{i,i}$ tiene el mismo valor en todos sus elementos, y este valor es:

$a_{i,i} = \frac{(n-1)!}{(n-r)!}$ (14)

Prueba. Tomando la fórmula de los sumatorios, es fácil ver que el producto de chirotopes sólo tiene dos valores posibles, 1 y 0 ; cuando $i = j$:

$$\chi(i,x_1,x_2,x_3,\dots,x_{r-1}) * \chi(j,x_1,x_2,x_3,\dots,x_{r-1})$$
$$= \chi(i,x_1,x_2,x_3,\dots,x_{r-1}) * \chi(i,x_1,x_2,x_3,\dots,x_{r-1})$$
$$= \chi(i,x_1,x_2,x_3,\dots,x_{r-1})^2$$
$$= \{(-1)^2, 0, 1^2\}$$
$$= \{1,0\} \quad (15)$$

Por consecuencia directa de la definición de matroide uniforme, el valor 0 aparecerá exclusivamente y siempre cuando la permutacion de r elementos que tomará χ incluye repeticiones. Por tanto, el número de casos donde χ no es 0, es: $\frac{n!}{(n-r)!}$. Sin embargo, al ser el primer elemento (i) fijo, las permutaciones se reducen a $\frac{(n-1)!}{(n-1-(r-1))!} = \frac{(n-1)!}{(n-r)!}$; para todo valor de i, y por consecuencia, este es el valor de todo elemento de la diagonal principal.

3.2.2 La matriz A es siempre una matriz simétrica.

Prueba. Partiendo de que la matriz simétrica posee la particularidad: $a_{i,j} = a_{j,i}$, se procede a hacer el cálculo de ambos elementos. Es fácil comprobar que el producto de los chirotopes χ siempre dará el mismo valor para toda permutación de elementos de E, al ser sólo un cambio de orden en los factores.

$$\mathbf{a_{i,j}} = \sum_{x_1=1}^{n} \sum_{x_2=1}^{n} \sum_{x_3=1}^{n} \dots \sum_{x_{r-1}=1}^{n} \chi(\mathbf{i},x_1,x_2,x_3,\dots,x_{r-1}) * \chi(\mathbf{j},x_1,x_2,x_3,\dots,x_{r-1}) \quad (16)$$

$$\mathbf{a_{j,i}} = \sum_{x_1=1}^{n} \sum_{x_2=1}^{n} \sum_{x_3=1}^{n} \dots \sum_{x_{r-1}=1}^{n} \chi(\mathbf{j},x_1,x_2,x_3,\dots,x_{r-1}) * \chi(\mathbf{i},x_1,x_2,x_3,\dots,x_{r-1}) \quad (17)$$

3.3 Técnica empleada para almacenar el mapeo del chirotope en el programa

La necesidad de almacenar el mapeo del chirotope durante la ejecución del programa supuso un reto mayor al previsto. Para esto, se almacenaron los signos del mapeo del chirotope dentro de un arreglo. Cualquier subconjunto de k elementos del conjunto soporte E tiene una entrada en el mapeo del chirotope, y el signo de esa entrada es almacenado en el arreglo. Cada tupla de k elementos (ordenados ascendentemente) pasa como un vector $\vec{a}$ a la función $f(\vec{a})$ (que será descrita más adelante), el valor devuelvo por esta función es la posición del arreglo donde se almacena el signo correspondiente a la tupla. Está técnica para almacenar registros de forma ordenadas se conoce como hashing, y la función $f(\vec{a})$ se denominar á como FUNCIÓN HASH DE COMBINACIONES.

Definición 7. ***Función Hash de combinaciones.*** *Sea $\vec{a}$ un vector de dimensíon k (índices de 1 a k), compuesto por el conjunto $A = \{a|$ a es un número entero entre 1 y $n\}$, donde los componentes de $\vec{a}$ se encuentran ordenados ascendentemente y sin repetición.*

Sea $f(\vec{a})$ una función sobre el conjunto de posibles valores que tome el vector $\vec{a}$: $f: A^k \to [0, \frac{n!}{k!(n-k)!} - 1]$

Es simple inferir que existen $\binom{n}{k}$ formas distintas de elegir el vector $\vec{a}$. Para asignarle un número de 0 a $\binom{n}{k} - 1$ a cada uno de estos posibles vectores $\vec{a}$, se puede emplear la siguiente función:

$$f(\vec{a}) = \sum_{i=1}^{k} \sum_{\substack{j=1+a_{i-1} \\ Si\ i=1 \to j=1}}^{a_i} \binom{n-j}{k-i} \quad (18)$$

Si como en el lenguaje C, los índices de un vector inician en 0 y no en 1, la fórmula pasa a:

$$f(\vec{a}) = \sum_{i=0}^{k-1} \sum_{\substack{j=1+a_{i-1} \\ Si\ i=0 \to j=1}}^{a_i} \binom{n-j}{k-i-1} \quad (19)$$

Esta fórmula es implementada por la función combinatorialHash mostrada a continuación. Cabe aclarar que la función combinatorial (n, k) es una implementación del coeficiente de Newton para obtener del total de combinaciones de k elementos de un total de n.

3.3.1 implementación de la función combinatorialHash en el lenguaje C++

```
int combinatorialHash (int a [], int k, int n) {
   int hash = 0;
   int j = 0;

   for (int i = 0; i < k; i++) {
      if (i > 0) j = a[i-1] + 1;
      else j = 1;

      for (; j < a[i]; j++) {
         hash += combinatorial (n - j, k - 1 - i);
      }
   }

   return hash;
}
```

La demostración de que esta fórmula es unívoca (con correspondencia uno-a-uno) y que tiene el rango especificado son pormenores que escapan al tema central del trabajo; además de ser sumamente extensos. Por tanto, no serán expuestos en el presente escrito.

3.4 Código fuente de los programas desarrollados durante la investigación

3.4.1 Matroide Uniforme $U_{2,4}$

```
//LIBRARY HEADERS
#include <iostream>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <iomanip>

using namespace std;

//PROTOTYPES
int chirotope (int,int);

int main(){
   system("cls");

   //DECLARATION VARIABLES
   int k = 0;
   int suma = 0;
   int maxVal = 4;
   double g[4][4];
   int chi1 = 0, chi2 = 0;

   cout<<"MATRIZ PARA LA MATROIDE U_2,4\n\n";

   //COMPUTING SMALL MATROID
   for (int i = 0; i < 4; ++i) {
      for (int j = 0; j < 4; ++j) {
         suma = 0;
         k = 1;
         cout<<"\nOPERACIONES PARA g"<<i + 1<<","<<j + 1<<"\n";
         while (k < 5) {
            if(k != (i + 1) && k != (j + 1)) {
               chi1 = chirotope (i + 1, k);
               chi2 = chirotope (j + 1, k);
               cout<<"chi ("<< i + 1 <<", "<<k<<") * chi ("<<j + 1<<", "<<k<<") = "
                  <<chi1<<" * "<< chi2<<" = "<< (chi1 * chi2)<< "\n";
               suma += chi1 * chi2;
            }
            k++;
         }
         g[i][j] = suma;
      }//for-j
   }//for-i
   //PRINTING MATRIX
   cout << "\n\n\n";
   for (int i = 0; i < 4; ++i) {
      for (int j = 0; j < 4; ++j) {
         cout<< setw(6)<< g[j][i]<< "  ";
      }//for-j
      cout << "\n";
   }//for-i
   getch();
   return 0;
}

//RETURN ASSOCIATED VALUE TO 2 ELEMENTS
int chirotope (int x0, int x1)  {
   int x[] = {x0, x1};
   int signo = 0;
   int aux = 0;
   int inversiones = 0;

   if (x0 == x1) return 0;

   for (int i = 0; i < 3; ++i) {
      for (int j = i + 1; j < 4; ++j) {
         if (x[i] > x[j]) {
            ++inversiones;
            aux = x[i];
            x[i] = x[j];
            x[j] = aux;
         }
      }//for-j
   }//for-i

   signo = 1;
   if(x[0] == 1 && x[1] == 4){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 3){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 4){
      signo = -1;
   } else if(x[0] == 3 && x[1] == 4){
      signo = -1;
   }

   return signo * pow(-1, inversiones);
}
```

3.4.2 Matroide Uniforme $U_{4,8}$

```
//LIBRARY HEADERS
#include <iostream>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <iomanip>

using namespace std;

//PROTOTYPES
int chirotope (int,int,int,int);

int main(){
   system("cls");
   //DECLARING VARIABLES
   int k = 0;
   int l = 0;
   int m = 0;
   int suma = 0;
   int maxVal = 8;
   int g[8][8];
   //int chi1 = 0, chi2 = 0;

   cout<<"MATRIZ PARA LA MATROIDE U_4,8\n\n";
   //COMPUTING SMALL MATROID
   for (int i = 0; i < maxVal; ++i) {
      for (int j = 0; j < maxVal; ++j) {
         k = 1;
         l = 1;
         m = 1;
         suma = 0;

         while (k <= maxVal) {
            suma += chirotope (i + 1,k,l,m) * chirotope (j + 1,k,l,
m);
            ++m;
            if (m == maxVal + 1) {
               m = 1;
               ++l;
            }//if-m

            if (l == maxVal + 1) {
               l = 1;
               ++k;
            }//if-l
         }//while-k

         g[i][j] = suma/24;
         cout<< setw(6)<< g[i][j]<< "  ";
      }//for-j
      cout << "\n";
   }//for-i

   getch();
   return 0;
}

//RETURN ASSOCIATED VALUE TO 4 ELEMENTS
int chirotope (int x0, int x1, int x2, int x3)  {
   int x[] = {x0, x1, x2, x3};
   int signo = 0;
   int aux;
   int inversiones = 0;

   for (int i = 0; i < 3; ++i) {
      for (int j = i + 1; j < 4; ++j) {
         if (x[i] > x[j]) {
            ++inversiones;
            aux = x[i];
            x[i] = x[j];
            x[j] = aux;
         } else if (i != j && x[i] == x[j]) {
            return 0;
         }
      }//for-j
   }//for-i

   signo = 1;

   if(x[0] == 1 && x[1] == 2 && x[2] == 4 && x[3] == 7){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 2 && x[2] == 5 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 2 && x[2] == 5 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 2 && x[2] == 6 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 2 && x[2] == 6 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 3 && x[2] == 4 && x[3] == 5
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 3 && x[2] == 4 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 3 && x[2] == 6 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 3 && x[2] == 6 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 4 && x[2] == 5 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 4 && x[2] == 5 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 4 && x[2] == 6 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 1 && x[1] == 5 && x[2] == 7 && x[3] == 8
){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 3 && x[2] == 4 && x[3] == 5
){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 3 && x[2] == 4 && x[3] == 6
){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 3 && x[2] == 4 && x[3] == 7
){
      signo = -1;
   } else if(x[0] == 2 && x[1] == 3 && x[2] == 4 && x[3] == 8
){
      signo = -1;
```

```
    } else if(x[0] == 2 && x[1] == 3 && x[2] == 6 && x[3] == 7
){
        signo = -1;
    } else if(x[0] == 2 && x[1] == 4 && x[2] == 6 && x[3] == 7
){
        signo = -1;
    } else if(x[0] == 2 && x[1] == 5 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 2 && x[1] == 6 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 4 && x[2] == 5 && x[3] == 7
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 4 && x[2] == 5 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 4 && x[2] == 6 && x[3] == 7
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 4 && x[2] == 6 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 5 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 3 && x[1] == 6 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 4 && x[1] == 5 && x[2] == 6 && x[3] == 7
){
        signo = -1;
    } else if(x[0] == 4 && x[1] == 5 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    } else if(x[0] == 4 && x[1] == 6 && x[2] == 7 && x[3] == 8
){
        signo = -1;
    }

    return signo * pow(-1, inversiones);
}
```

3.4.3 Programa final

```
//LIBRARY HEADERS
#include <iostream>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <string>
#include <iomanip>

using namespace std;


//PROTOTYPES
void compute_print_matrix (int elements, int rank, char chirotop
e_mapping[]);
int chirotope(int tuple[], int elements, int rank, char chirotope_
mapping[]);
void print_chirotope (int elements, int rank, char chirotope_map
ping[]);
int combinatorial(int n, int k);
int combinatorialHash (int a[], int length, int max);
long fact(int);
void printArray(int a[], int);

int main(){
    int op = 0;
    int op2 = 0;
    int length_chirotope;
    int rank;
    int elements;
    int change_element;
    char value;

    system("cls");

    do {
        cout<<"\n\n\n\nMENU DE CHIROTOPES\n\n";
        cout<<"[ 1 ] CREAR UN NUEVO CHIROTOPE\n";
        cout<<"[ 2 ] SALIR \nOPCION ELEGIDA: ";
        cin>>op;

        switch (op) {
            case 1://CHIROTOPE
                cout<<"\n\nCREACION DE NUEVA MATROIDE";
                cout<<"\nINGRESE EL NUMERO DE ELEMENTO
S DE LA MATROIDE: ";
                cin>>elements;

                cout<<"INGRESE EL RANGO DE LA MATROIDE:
";
                cin>>rank;

                length_chirotope = combinatorial(elements, rank);
                //char chirotope_mapping [length_chirotope];
                char chirotope_mapping[600];

                for (int i = 0; i < length_chirotope; i++){
                    if(i % 3 == 0) chirotope_mapping[i] = '+';
                    else if(i % 3 == 1) chirotope_mapping[i] = '0';
                    else chirotope_mapping[i] = '-';
                }
                print_chirotope (elements, rank, chirotope_mapping);
                compute_print_matrix (elements, rank, chirotope_ma
pping);

                //CHANGE CHIROTOPE
                do {
                    cout<<"\n\nCAMBIAR SIGNOS DEL CHIROTOP
E?\n";
                    cout<<"[ 1 ] SI\n";
                    cout<<"[ 2 ] NO \nOPCION ELEGIDA: ";
                    cin>>op2;
                    if(op2 == 2) {
                    } else {
                        system("cls");
                        print_chirotope (elements, rank,
                        chirotope_mapping);
                        cout<<"\n\nTECLEA EL ELEMENTO
                        A CAMBIAR DEL CHIROTOPE: ";
                        cin>>change_element;

                        cout<<"\nVALOR DE REEMPLAZO ['-', '0', '+']
: ";
                        cin>>value;

                        if(value != '-' && value != '+') value = '0';
```

```
                chirotope_mapping[change_element] = value;


                print_chirotope (elements,
                rank, chirotope_mapping);
              }
            } while(op2 != 2);//do-while
            compute_print_matrix (elements, rank, chirotope_ma
pping);
          break;
          case 2:
            cout<<"\n\nFINALIZANDO PROGRAMA...\n";
            //exit(0);
            return 0;
          break;
          default: cout<<"\n\nOPCION NO DISPONIBLE\n";
        }//switch
    } while (op != 2);
    //getch();
    cin.ignore();
    return 0;
}//int main

void compute_print_matrix (int elements, int rank, char chirotop
e_mapping[]){
    //DECLARING VARIABLES
    int aux = 0;
    int suma = 0;
    int g[elements][elements];
    int lim = pow(elements, rank);
    int k = 0;
    int tuple[rank];
    int aux_chirotope;

    //COMPUTING SMATROID
    cout<<"\n\nMATRIZ DE LA MATROIDE:\n";
    for (int i = 0; i < elements; ++i) {
        for (int j = 0; j < elements; ++j) {
            suma = 0;

for (int j2 = 0; j2 < lim; j2++){
                tuple[0] = i + 1;

                i2 = j2;
                for (int k = rank - 1; k > 0; k--){
                  tuple[k] = (i2 % elements) + 1;
                  i2 = i2 / elements;
                }

                aux_chirotope = chirotope(tuple, elements, rank, chir
otope_mapping);
                tuple[0] = j + 1;
                aux_chirotope *= chirotope(tuple, elements, rank, ch
irotope_mapping);
                suma += aux_chirotope;
}
            g[i][j] = suma;
            cout<< setw(6)<< g[i][j]<< "  ";
        }//for-j
        cout << "\n";
    }//for-i
}//void calcular_matriz

//RETURN ASSOCIATED VALUE TO A TUPLE
int chirotope(int tuple0[12], int elements, int rank, char chirotop
e_mapping[])  {
    int signo = 0;
    int aux = 0;
    int inversiones = 0;
    int hash = 0;
    int tuple[12];

    for (int i = 0; i < rank; ++i) {
        tuple[i] = tuple0[i];
    }//for-i


    for (int i = 0; i < rank - 1; ++i) {
        for (int j = i + 1; j < rank; ++j) {
            if (tuple[i] > tuple[j]) {
                ++inversiones;
                aux = tuple[i];
                tuple[i] = tuple[j];
                tuple[j] = aux;
            } else if (tuple[i] == tuple[j] && i != j) {
                return 0;
            }
        }//for-j
    }//for-i

    hash = combinatorialHash(tuple,rank, elements);
    if(chirotope_mapping[hash] == '+') signo = 1;
    else if(chirotope_mapping[hash] == '-') signo = -1;
    else return 0;

    signo *= pow(-1, inversiones);
    return signo;
}//int chirotope

void print_chirotope (int elements, int rank, char chirotope_map
ping[]) {
    cout<<"\n\nMAPEO DEL CHIROTOPE:\n";
    int tuple[rank];
    for(int i = 0; i < rank; i++){
        tuple[i] = i + 1;
    }

    int hash;
    int nck = combinatorial(elements, rank) - 1;
    int i = 0;
    int aux;
    while(hash < nck){
        hash = combinatorialHash(tuple, rank,elements);
        cout<<hash<<")   {";
        /*for(int j = 0; j < rank; j++){
            cout<<tuple[j]<<"";
            if(j < rank - 1) cout<<", ";
        }*/
        printArray(tuple, rank);
        cout<<"}   --> "<<chirotope_mapping[hash]<<"\n";
        for(i = rank - 1; i >= 0; i--){
            if(tuple[i] <= elements - rank + i) break;
        }

        aux = tuple[i];
        for (int j = 0; j < rank-i; j++){
            tuple[j+i] = aux + j + 1;
        }
    }
}//void print_chirotope
```

**IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)**

```
int combinatorial(int n, int k){
   if (n < 0 || k < 0) return 0;
   else if (n == k) return 1;
   else if (k == 1 || k == n - 1) return n;
   else if (k == 0) return 1;
   else if (n == 0) return 0;
   else return (combinatorial(n - 1, k - 1) + combinatorial(n - 1,
k));
}

int combinatorialHash (int a[], int length, int max){
   int hash = 0;
   int j = 0;

   for (int i = 0; i < length; i++){

      if (i > 0) j = a[i-1] + 1;
      else j = 1;
      for (; j < a[i]; j++){
         hash += combinatorial(max - j, length - 1 - i);
      }
   }

   return hash;
}

// Returns factorial of n
long fact(int n) {
    if(n==0)
    return 1;
   long res = 1;
   for (int i = 2; i <= n; i++)
      res = res * i;
   return res;
}

void printArray (int a[], int length){
   for (int i = 0; i < length; i++){
      cout<<""<<a[i]<<"";
      if(i < length - 1) cout<<", ";
   }//FOR
   //cout<<"\n";
}
```

4. Resultados

El software desarrollado fue puesto a prueba con distintas instancias de matroides uniformes, obteniendo matrices cuya validez como representaciones orientadas fue posteriormente evaluada.

Algunos ejemplos de representaciones matriciales calculadas con el software se muestran a continuación:

4.1 Matriz cuadrada para la matroide uniforme $\boldsymbol{U}_{4,8}$

$$\begin{matrix} 35 & 4 & 8 & -6 & 16 & 6 & 8 & -8 \\ 4 & 35 & 8 & 6 & -8 & 18 & -8 & -4 \\ 8 & 8 & 35 & 10 & -4 & -6 & 16 & -8 \\ -6 & 6 & 10 & 35 & -6 & 4 & 10 & 18 \\ 16 & -8 & -4 & -6 & 35 & 6 & 8 & 4 \\ 6 & 18 & -6 & 4 & 6 & 35 & -6 & 10 \\ 8 & -8 & 16 & 10 & 8 & -6 & 35 & 8 \\ -8 & -4 & -8 & 18 & 4 & 10 & 8 & 35 \end{matrix} \quad (20)$$

4.2 Matriz cuadrada para la matroide uniforme $\boldsymbol{U}_{3,6}$

$$\begin{matrix} 20 & 12 & 0 & 12 & 0 & -4 \\ 12 & 20 & -12 & 0 & 4 & 0 \\ 0 & -12 & 20 & 4 & 0 & -12 \\ 12 & 0 & 4 & 20 & -12 & 0 \\ 0 & 4 & 0 & -12 & 20 & -12 \\ -4 & 0 & -12 & 0 & -12 & 20 \end{matrix} \quad (21)$$

4.3 Matriz cuadrada para la matroide uniforme $\boldsymbol{U}_{2,6}$

$$\begin{matrix} 5 & 4 & -2 & -4 & 2 & 0 \\ 4 & 5 & -4 & -2 & 0 & 2 \\ -2 & -4 & 5 & 0 & 2 & -4 \\ -4 & -2 & 0 & 5 & -4 & 2 \\ 2 & 0 & 2 & -4 & 5 & -4 \\ 0 & 2 & -4 & 2 & -4 & 5 \end{matrix} \quad (22)$$

4.4 Matriz cuadrada para la matroide uniforme $\boldsymbol{U}_{2,4}$

$$\begin{matrix} 3 & -2 & 2 & 0 \\ -2 & 3 & 0 & 2 \\ 2 & 0 & 3 & 2 \\ 0 & 2 & 2 & 3 \end{matrix} \quad (23)$$

4.5 Matriz cuadrada para la matroide uniforme $\boldsymbol{U}_{3,4}$

$$\begin{matrix} 6 & -2 & -2 & -2 \\ -2 & 6 & -2 & -2 \\ -2 & -2 & 6 & -2 \\ -2 & -2 & -2 & 6 \end{matrix} \quad (24)$$

IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)

5. Análisis de Resultados

Un análisis preliminar demostró que ninguna de las matrices proporcionadas por el software cumple con las propiedades de la matroide que se espera represente.

Se demostró que, si una matroide tiene un rango r menor o igual a $n-1$ (siendo n el número de elementos de la matroide), tendrá al menos un conjunto linealmente dependiente de cardinalidad $r+1$.

$$\begin{aligned} M &= (\mathcal{E}, \mathcal{B}) \\ n &= |\mathcal{E}| \\ r &= rank(M) = |b|, \forall b \in \mathcal{B} \\ Si\ r &\leq n-1 \\ &\Longrightarrow \exists C \mid C \subseteq \mathcal{E}\ , |C| \geq r+1 \\ &\Longrightarrow C \notin \mathcal{B}, y\ C \subsetneq b, \forall b \in \mathcal{B} \\ &\Longrightarrow C\ no\ es\ un\ conjunto\ l.i.^{*} \end{aligned}$$

*Linealmente independiente

(25)

Por lo anterior, su representación como una matriz cuadrada (de tamaño $n * n$) sobre el campo de los números reales $\mathbb{R}$, debe tener obligatoriamente un determinante igual a 0 (implicación directa de que existe un conjunto linealmente dependiente dentro del conjunto de vectores columna de una matriz). De no ser así, la matriz no es una representación válida de la matroide; puesto que todos los vectores son independientes.

Este último hallazgo, a la luz de las pruebas computacionales que corroboran que dicha fórmula no siempre devuelve matrices con determinante nulo, muestra que se requiere de una propiedad emergente para garantizar (o en su caso, descartar) la universalidad de la fórmula del doctor J. A. Nieto; y, por tanto, del software basado en su teoría.

Como parte del análisis de los resultados arrojados por el programa, se llegó a la siguiente conjetura:

Conjetura 1. *Se teoriza que la submatriz generada tomando sólo las primeras r filas de la matriz cuadrada calculada originalmente, será siempre una representación valida de la matroide. Sin embargo, se requiere una demostración estricta de esto para poder afirmarlo con total certeza.*

Ejemplos que sustentas la conjetura anterior son las siguientes matrices rectangulares de dimensiones $r * n$ (recuérdese que r es el rango de la matriz y n el número de elementos), que son submatrices de las originalmente calculadas con el software y que sí representan a sus matroides correspondientes:

Matroide uniforme $\mathbf{U}_{4,8}$

$$\begin{matrix} 35 & 4 & 8 & -6 & 16 & 6 & 8 & -8 \\ 4 & 35 & 8 & 6 & -8 & 18 & -8 & -4 \\ 8 & 8 & 35 & 10 & -4 & -6 & 16 & -8 \\ -6 & 6 & 10 & 35 & -6 & 4 & 10 & 18 \end{matrix}$$

(26)

Matroide uniforme $\mathbf{U}_{3,6}$

$$\begin{matrix} 20 & 12 & 0 & 12 & 0 & -4 \\ 12 & 20 & -12 & 0 & 4 & 0 \\ 0 & -12 & 20 & 4 & 0 & -12 \end{matrix}$$

(27)

Matroide uniforme $\mathbf{U}_{2,6}$

$$\begin{matrix} 5 & 4 & -2 & -4 & 2 & 0 \\ 4 & 5 & -4 & -2 & 0 & 2 \end{matrix}$$

(28)

Matroide uniforme $\mathbf{U}_{3,4}$

$$\begin{matrix} 6 & -2 & -2 & -2 \\ -2 & 6 & -2 & -2 \\ -2 & -2 & 6 & -2 \end{matrix}$$

(29)

Matroide uniforme $\mathbf{U}_{2,4}$

$$\begin{matrix} 3 & -2 & 2 & 0 \\ -2 & 3 & 0 & 2 \end{matrix}$$

(30)

Esta última matriz, a través de operaciones entre renglones y entre columnas, puede transformarse en la representación de la matroide $U_{2,4}$ ofrecida en la mayoría de libros y artículos [2][3][8]:

$$\begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{matrix}$$

(31)

Este resultado sugiere que, si bien el modelo de generación propuesto no garantiza representaciones válidas en forma cuadrada, sí podría funcionar como un generador confiable de representaciones rectangulares, lo cual constituye una línea de investigación futura.

**IJISTA, Vol. 1, Núm. 1, Mayo 2025,
ISSN (en trámite)**

6. Conclusiones

El desarrollo de herramientas computacionales especializadas en estructuras matemáticas abstractas como las matroides orientadas representa un desafío interdisciplinar con múltiples aplicaciones potenciales en áreas como la geometría combinatoria, la optimización y la ciencia de datos.

En este contexto, el presente trabajo introduce un software escrito en C++ que permite generar representaciones matriciales de matroides orientadas a partir de un conjunto de elementos y su chirotope (χ), apoyado en una fórmula propuesta por el Dr. J. A. Nieto.

Dicho software es un esfuerzo pionero que, aunque incipiente, apunta a cubrir la falta de herramientas capaces de generar representaciones matriciales de matroides dadas, las cuales serían de gran utilidad tanto para la educación como para la investigación en este campo.

Su desarrollo atiende un vacío real en la literatura computacional sobre matroides orientadas. Además, sienta bases para futuros proyectos más integrados que vinculen teoría y práctica mediante algoritmos de verificación estructural y visualización dinámica de representaciones.

Respecto al primer prototipo ideado por D. Aldana, fue posible optimizar el algoritmo y, sobre todo, el uso de memoria del programa. Además de que se desarrollaron métodos que permiten extender el uso del programa a matroides de cualquier rango menor a 12, mientras que el prototipo original solo admitía matroides con rango 3 o 7.

Hasta la fecha, existen pocas herramientas que permitan calcular representaciones matriciales de matroides, y aún menos enfocadas en matroides orientadas. Algunas bibliotecas matemáticas computacionales permiten explorar propiedades de matroides (como SageMath [11] u Oid), pero no ofrecen funcionalidades específicas para generar representaciones matriciales directas de forma automatizada. Esta ausencia resalta la necesidad de contar con software especializado, y el trabajo aquí desarrollado busca cubrir parcialmente esa brecha.

Esta carencia actual de herramientas para trabajar con matroides orientadas puede atribuirse, en parte, a la complejidad de formalizar computacionalmente los axiomas de orientación y su traducción a estructuras matriciales.

Además, estudios sobre representaciones matriciales de matroides suelen centrarse en su existencia y condiciones algebraicas (por ejemplo, representabilidad sobre ciertos campos), pero rara vez en métodos constructivos automatizados. Por tanto, este desarrollo representa un paso inicial hacia la sistematización computacional del análisis de matroides orientadas desde una perspectiva constructiva y aplicable.

Como parte de la investigación, se encontraron resultados que exigen un mayor análisis sobre la fórmula propuesta por el doctor J. A. Nieto en [3], para así complementar su investigación y saber si dicha fórmula funciona para generar matrices de dimensiones $r * n$, saber en qué condiciones la fórmula es válida, o directamente refutarla.

Una de las principales limitaciones del enfoque presentado en este trabajo radica, justamente, en su dependencia de una fórmula cuya validez aún no ha sido formalmente demostrada. Aunque el software desarrollado permite generar matrices a partir de un chirotope y observar propiedades relevantes de matroides orientadas, la ausencia de una base teórica rigurosa que garantice la representabilidad de las matrices generadas compromete su aplicabilidad general.

Esto delimita el alcance del software a un uso exploratorio o educativo, más que a aplicaciones formales en investigación matemática avanzada. Además, la implementación actual no cuenta con un sistema de verificación automático que identifique inconsistencias algebraicas o violaciones de axiomas matroidales, o incluso que verifique si la matroide es representable antes de intentar encontrar su representación matricial. Esto plantea desafíos al momento de escalar su uso a instancias más complejas o de mayor dimensión.

Finalmente, este proyecto no solo representa una contribución funcional en la creación de herramientas específicas para el estudio computacional de matroides orientadas, sino que también plantea interrogantes relevantes que pueden enriquecer la comprensión teórica y práctica de estas estructuras. Se espera en el futuro encontrar resultados matemáticos que ayuden a complementar la teoría sobre la representación de matroides, tales como:

1. Generar un algoritmo que permita identificar computacionalmente cuando una matroide es o no es representable.

2. Ofrecer una demostración o refutación rigurosa de la conjetura expresada anteriormente, sobre las matrices de tamaño $r * n$ calculada con la fórmula del doctor J. A. Nieto, y sobre la propia fórmula.

3. Encontrar un algoritmo para calcular representaciones matriciales de cualquier matroide, con un sustento matemático estricto.

7. Referencias

[1] Oxley, James, *Matroid Theory*, 2nd edn, Oxford Graduate Texts in Mathematics (Oxford, 2011; online edn, Oxford Academic, 17 Dec. 2013), https://doi.org/10.1093/acprof:oso/9780198566946.001.0001, accessed 3 May 2025.

[2] J. Oxley, WHAT IS A MATROID? 1991, pp. 1–45.

[3] N. G. J. Antonio and Marín Miriam Christina, Elementos de la Teoría de Matroides. El Colegio de Sinaloa, 2005.

[4] R. F. M. Jean Cardinal and C. Hidalgo-Toscano, "Chirotopes of Random Points in Space are Realizable on a Small Integer Grid," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:210859350

[5] J. Geelen and P. Nelson, "The number of lines in a matroid with no U2, n-minor," European Journal of Combinatorics, vol. 50, pp. 115–122, 2015, doi: https://doi.org/10.1016/j.ejc.2015.03.026.

[6] C. Merino, M. Ramírez-Ibáñez, and G. Sanchez, "The Tutte Polynomial of Some Matroids," International Journal of Combinatorics, vol. 2012, Feb. 2012, doi: 10.1155/2012/430859.

[7] J. Valero and I. Lizarazo, "Multispectral image classification from axiomatic locally finite spaces-based segmentation," UD y la geomática, no. 13, 2019, doi: 10.14483/23448407.15230.

[8] J. A. Nieto, Oriented Matroid Theory as a Mathematical Framework for M-Theory. 2006. [Online]. Available: https://arxiv.org/abs/hep-th/0506106

[9] E. Gawrilow y M. Joswig, "Polymake: a framework for analyzing convex polytopes," en Polymake Documentation, [En línea]. Disponible en: https://polymake.org/doku.php/documentation/latest/matroid. [Accedido: 10-may-2025].

[10] S. Kingan, "Oid: a software system for matroids," [En línea]. Disponible en: https://userhome.brooklyn.cuny.edu/skingan/matroids/software.html. [Accedido: 10-may-2025].

[11] The Sage Developers, Matroid Theory – SageMath Documentation, versión 10.6, 2025. [En línea]. Disponible en: https://doc.sagemath.org/html/en/reference/matroids/index.html. [Accedido: 10-may-2025].

[12] Maplesoft, "Matroids – Maple Help," [En línea]. Disponible en: https://www.maplesoft.com/support/help/Maple/view.aspx?path=Matroids. [Accedido: 10-may-2025].

[13] A. Solana Hernández, "Path planning strategies for navigation of mobile robots," Procedia Computer Science, vol. 133, pp. 746–753, 2018. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2214914718305130. [Accedido: 10-may-2025].

[14] Woodall. "Types of matroids". MathsNet: School of Mathematical Sciences Intranet Pages. Accedido el 12 de mayo de 2025. [En línea]. Disponible: https://www.maths.nottingham.ac.uk/plp/pmadw/PG/matroid.ch3.pdf