

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Estructuras de Datos Compactas para Homología Persistente en Complejos Cubicales

PROPUESTA DE TEMA DE MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

Valeria Serrano

MODALIDAD:
Memoria

PROFESOR GUÍA:
Gonzalo Navarro
PROFESOR GUÍA 2:
Rolando Kindelan

SANTIAGO DE CHILE
2026

1. Introducción

El Análisis Topológico de Datos (*Topological Data Analysis*, TDA) se ha consolidado durante los últimos años como un paradigma fundamental para la extracción de características multiescala en datos complejos. Su punto de partida es la topología, que estudia aquellas propiedades de los espacios que permanecen invariantes bajo transformaciones continuas, como la conectividad, la compacidad o la existencia de agujeros. Para cuantificar estas propiedades, la homología asocia a cada espacio una familia de grupos abelianos que miden ciclos que no son fronteras de celdas de dimensión superior. Sin embargo, en aplicaciones reales los datos no forman un único espacio topológico fijo, sino que admiten múltiples escalas de análisis. La **homología persistente** extiende la homología clásica considerando una filtración, i.e., una familia creciente de espacios parametrizada por una escala. De este modo, resume la estructura topológica subyacente de un conjunto de datos, inducida por su geometría o por una función de densidad o intensidad, a través del estudio de la aparición (*birth*) y desaparición (*death*) de rasgos topológicos, como componentes conexas, túneles, cavidades y vacíos de mayor dimensión, a medida que varía un parámetro de filtración, que puede ser una escala de distancia o un umbral sobre una función de densidad o intensidad.

En el procesamiento de imágenes digitales (2D), tomografías de volumen (3D) y secuencias espacio-temporales (4D), los datos se representan mediante grillas regulares de píxeles o vóxeles, según el caso. Sobre estas grillas se construyen los **complejos cubicales**, que corresponden a complejos formados por celdas elementales de dimensión k: 0-celdas (vértices), 1-celdas (aristas), 2-celdas (cuadrados o caras), 3-celdas (cubos) y, en general, d-celdas (hipercubos). A diferencia de los complejos simpliciales que requieren triangulación sobre una grilla (por ejemplo, la de Freudenthal), los complejos cubicales preservan la geometría ortogonal de la imagen sin necesidad de subdividir el espacio. No obstante, el cálculo algebraico de la homología persistente sobre complejos cubicales enfrenta una importante limitación computacional: la explosión del consumo de memoria RAM cuando el tamaño o la dimensión del volumen aumentan.

Desde el punto de vista algorítmico, el proceso del cálculo de homología persistente sobre un complejo cubical se compone de tres fases secuenciales:

- **Filtración u Ordenamiento:** Se asigna un orden estricto de aparición a todas las celdas del complejo (vértices, aristas, caras y cubos) en función de la función de intensidad o densidad subyacente, usualmente bajo una V-filtración o *lower-star*, donde el valor de una celda equivale al máximo valor de sus vértices incidentes.
- **Mantención de Componentes Conexas (H_0 y H_{d-1}):** Se rastrean los nacimientos y muertes de componentes mediante el algoritmo de *Union-Find*. Gracias a dualidad de Alexander en espacios cubicales, las clases de la dimensión máxima H_{d-1} también se puede resolver mediante *Union-Find* sobre el grafo dual de la grilla.
- **Reducción de la Matriz de Borde (H_k intermedios):** Para las dimensiones intermedias, como H_1 en 3D, se construye la matriz de borde ordenada según la filtración y se realiza una reducción gaussiana columna por columna sobre el campo de Galois $\mathbb{F}_2 = \{0, 1\}$. Los pivotes sobrevivientes determinan los pares de persistencia (tiempo de nacimiento y muerte de cada rasgo).

Para un volumen 3D de tamaño 1024^3 el número de celdas cubicales en la grilla es de aproximadamente $2^d \cdot n^d \approx 8{,}6 \cdot 10^9$ celdas. Almacenar y manipular esta cantidad masiva de elementos produce un colapso en la memoria RAM principal, forzando al sistema a realizar intercambio de páginas a disco (*swapping*) o provocando errores por agotamiento de memoria (*out-of-memory*).

Actualmente, el peak de consumo de memoria RAM se concentra en tres ejes del sistema:

- **Filtración Ordenada:** Almacenar la secuencia explícita ordenada de miles de millones de celdas y sus valores.
- **Estado de *Union-Find*:** La estructura de datos que mantiene la partición dinámica de las celdas duales en componentes conexas. En el caso dual permite rastrear las componentes del grafo dual y, por dualidad, las clases de homología H_{d-1}.
- ***Fill-in* de la matriz reducida:** La acumulación de nuevos elementos no nulos creados por las operaciones de suma de columnas sobre $\mathbb{F}_2$ durante la reducción gaussiana.

La presente propuesta de memoria busca abordar este cuello de botella mediante el diseño, implementación y evaluación de estructuras de datos compactas aplicadas sobre los tres ejes mencionados. Para aislar el impacto de las estructuras de datos respecto a las heurísticas algorítmicas, la memoria mantendrá fijo el algoritmo del estado del arte de referencia [3] junto con sus tres componentes algorítmicas centrales:

- Dualidad de Alexander para el cálculo de dimensiones extremas (H_0 y H_{d-1}) vía *Union-Find* dual.
- *Pruning* de aristas duales directas de persistencia cero.
- *Lookup tables* precalculadas para clasificar configuraciones de vecindarios locales.

Al sustituir las representaciones de memoria tradicionales por estructuras compactas adaptadas a la regularidad cubical, se busca reducir el peak de consumo de memoria RAM sin alterar los algoritmos de reducción ni el resultado final, permitiendo procesar volúmenes masivos en computadores convencionales que actualmente resultan inviables por limitaciones de memoria. Cualquier mejora adicional en tiempo de ejecución sería un beneficio secundario.

2. Situación Actual

Dentro de la literatura de homología persistente sobre complejos cubicales, la evolución de los algoritmos ha estado guiada por la necesidad de reducir la complejidad en espacio y tiempo. A continuación se revisan las principales representaciones y optimizaciones propuestas, para luego identificar la oportunidad que motiva esta memoria.

Evolución del Estado del Arte

Representación Explícita [7]: El trabajo de Wagner et al. introdujo la estructura *CubeMap*, una grilla densa que almacena explícitamente el estado y valor de cada celda del complejo cubical. Si bien evitó la triangulación simplicial, que expandía la cantidad de símplices por un factor de $d!$, *CubeMap* requiere almacenar arreglos explícitos que crecen exponencialmente con la dimensión d, resultando inviable para volúmenes grandes.

Representación Implícita [4]: Siguiendo la filosofía de *Ripser* para complejos de Vietoris-Rips, *CubicalRipser* eliminó la necesidad de almacenar el complejo en memoria. En su lugar, calcula las coordenadas y conexidades de las celdas *on-the-fly* usando aritmética de coordenadas. Asimismo, incorpocó la dualidad de Alexander (mediante la opción `--top_dim`) para resolver la dimensión superior H_{d-1} con *Union-Find* en lugar de reducción matricial.

Flash Cubical [3]: Corresponde al estado del arte actual en cómputo de homología persistente cubical sobre V-filtraciones en imágenes 2D y volúmenes 3D. *Flash Cubical* combina tres técnicas aritméticas clave:

- **Dual Union-Find:** Extiende la dualidad de Alexander construyendo un grafo dual donde los vóxeles son vértices duales y las caras inter-vóxel son aristas duales, agregando un vértice especial "infinito". Esto permite resolver tanto H_0 como H_{d-1} usando *Union-Find* en tiempo casi lineal, reservando la reducción matricial únicamente para la dimensión intermedia H_1 en 3D.
- **Local Edge Pruning:** Antes de ejecutar *Union-Find*, identifica y elimina aristas positivas y negativas de persistencia cero en la filtración semi-refinada, reduciendo drásticamente el número de elementos a ordenar y las operaciones de búsqueda (*find*).
- **Lookup Tables:** Aprovechando la regularidad geométrica del complejo cubical se precalculan las 15.935 configuraciones posibles del vecindario de 26 vecinos de un vóxel. Durante la ejecución, la clasificación local de las celdas se obtiene por consulta directa a esta tabla, evitando cómputos repetitivos.

A pesar de estas optimizaciones algorítmicas, *Flash Cubical* almacena las secuencias de celdas restantes y el estado de conjuntos disjuntos utilizando estructuras de memoria genéricas, lo que genera peaks de RAM de varios gigabytes en volúmenes estándar de la literatura (128^3 o 256^3).

Análisis de la Memoria en Complejos Cubicales y Oportunidad

Un diagnóstico fundamental de este trabajo es la siguiente propiedad geométrica: en una grilla cúbica regular, el complejo topológico en sí mismo es "gratis" desde el punto de vis-

ta del almacenamiento. Las adyacencias, fronteras y cofronteras entre celdas no requieren almacenamiento explícito, se recalculan mediante aritmética de coordenadas $O(1)$.

Por lo tanto, la memoria peak en el estado del arte no se consume en la topología estática de la grilla, sino en tres ejes del estado dinámico y ordenado:

- **Filtración Ordenada:** Almacenar explícitamente la secuencia ordenada de celdas para guiar el barrido del algoritmo.
- **Estado de *Union-Find*:** Los arreglos de punteros de padre y rangos para el bosque de componentes en el grafo dual.
- ***Fill-in* de la Matriz reducida:** La inserción de nuevos elementos no nulos durante la suma de columnas sobre $\mathbb{F}_2$.

Estructuras de Datos Compactas Fundamentales

En el área de Estructuras de Datos Compactas [5] se han desarrollado técnicas avanzadas para almacenar secuencias, conjuntos esparsos y arreglos en espacio cercano a su entropía teórica, manteniendo soporte para operaciones `access`, `rank` y `select` en tiempo constante o logarítmico:

- **Codificación de Enteros de Longitud Variable [5, Sec. 2.7]:** Métodos como codificación γ, δ, VByte o Simple9 permiten empaquetar secuencias de enteros usando un número variable de bits según la magnitud del valor.
- **Elementos de Tamaño Variable [5, Sec. 3.2]:** Uso de punteros muestreados o punteros densos para indexar elementos de ancho variable sin pagar el costo de punteros explícitos por elemento.
- **Bitvectors Muy Esparsos [5, Sec. 4.4]:** Estructuras basadas en codificación de Elias-Fano o representaciones agrupadas que soportan `select` en $O(1)$ sobre conjuntos de bits muy esparsos.
- **Navegación sin Materialización y Estructuras k^d:** Representaciones jerárquicas tipo k^d-raster y k^d-tree que permiten almacenar cotas locales (min, max) sobre grillas multinivel, permitiendo realizar consultas de rangos o barridos en orden de filtración sin desplegar la secuencia completa de datos en RAM.

Otro antecedente importante para esta memoria es el trabajo de Barbay et al. [1], que presenta una representación totalmente comprimida de secuencias con `access`, `rank` y `select` en tiempo $O(\lg\lg\sigma)$, y tiempo promedio $O(\lg\mathcal{H}_0(s))$, usando $n\mathcal{H}_0(s) + o(n)(\mathcal{H}_0(s)+1)$ bits. A diferencia de representaciones previas, esta estructura comprime también la redundancia, lo que es importante en secuencias altamente compresibles. Además, derivan la primera representación comprimida de colecciones dinámicas de conjuntos disjuntos, con Union-Find en espacio $(1+\varepsilon)n\mathcal{H}(C) + O(|C|\lg n) + o(n)$ y tiempo $O(r\alpha(n) + (1/\varepsilon)n\lg\lg n)$ para r operaciones. Este último resultado es aplicable al Union-Find sobre el grafo dual en homología persistente cubical.

Compresión de Bitmaps y Listas Invertidas: Complejidad de la Diferencia Simétrica

Un segundo antecedente importante es el estudio experimental comparativo entre compresión de bitmaps y listas invertidas de Wang et al. [8]. Aunque el estudio esté formulado bajo el contexto de bases de datos e información, su observación central es aplicable al problema algebraico de reducción matricial, ya que ambas representaciones resuelven el problema de almacenar y operar sobre un conjunto ordenado de enteros, pero con estrategias opuestas. Los bitmaps comprimidos privilegian operaciones bit a bit sobre palabras, mientras que las listas invertidas privilegian operaciones de fusión sobre deltas.

En el contexto del cálculo de homología persistente, la reducción gaussiana sobre $\mathbb{F}_2$ equivale a sumas de columnas módulo 2, i.e., diferencia simétrica de conjuntos de bits. La complejidad del XOR entre dos vectores de bits A y B varía según de la densidad local de bits de las zonas:

- **Zonas esparsas de bits:** Se reducen a la fusión (*merge*) de elementos no nulos en tiempo $O(a + b)$, proporcional a la suma de los tamaños de los conjuntos.
- **Zonas densas de bits:** Conviene explotar la estructura en runs de 1's consecutivos ($O(\text{runs}(A) + \text{runs}(B))$), o procesar directamente como bitmaps raster mediante instrucciones vectorizadas de tipo XOR bit a bit.

Wang et al. [8] cuantifican experimentalmente este comportamiento para 9 métodos de compresión de bitmaps y 12 de listas invertidas, mostrando que la elección óptima depende de la densidad, la longitud de los runs y la distribución subyacente. Esto sugiere que el uso de representaciones híbridas sobre el *fill-in* de la matriz reducida podría resultar más útil que utilizar una única estructura genérica.

Cabe mencionar que las representaciones de secuencias totalmente comprimidas de Barbay et al. [1] también admiten aplicaciones directas sobre listas invertidas y relaciones binarias, y soportan operaciones de intersección y acceso a listas en tiempo $O(\lg \lg \lambda)$. En el caso de reducción matricial sobre $\mathbb{F}_2$ estas representaciones ofrecen una alternativa complementaria a la codificación adaptativa por zonas si se considera que las columnas se pueden ver como listas de índices de filas no nulas.

A la fecha, no existe en la literatura trabajo que integre de manera sistemática estructuras de datos compactas sobre los tres ejes de estado de un algoritmo de homología persistente cubical. Trabajos previos en el ámbito simplicial (como el uso de wavelet trees sobre símplices maximales propuesto por Kindelan [6], o Simplex Trees por Boissonnat et al. [2]) no se aplican directamente al caso cubical, donde la presencia de la grilla regular permite evitar la materialización de la secuencia mediante codificaciones jerárquicas compactas.

3. Objetivos

Objetivo General

Diseñar, implementar y evaluar un conjunto de estructuras de datos compactas aplicadas sobre los tres ejes de consumo de memoria del cómputo de homología persistente cubical (filtración ordenada, estado de Union-Find y fill-in de la matriz reducida sobre $\mathbb{F}_2$), manteniendo fijo el algoritmo de referencia (**Flash Cubical** [3]) y sus tres componentes algorítmicas (dualidad de Alexander, pruning de aristas duales y lookup tables), con el fin de reducir el peak de consumo de memoria RAM garantizando la obtención de diagramas de persistencia idénticos a los del estado del arte.

Objetivos Específicos

1. **Replicación y Perfilado Baseline:** Replicar el entorno de ejecución de **Flash Cubical** e implementar un sistema de trazado y perfilado de memoria en profundidad (*heaptrack/Valgrind Massif*) sobre volúmenes estándar de la literatura (Bonsai, Head Aneurism, Fuel, Lena, DIV2K), aislando cuantitativamente el consumo peak de RAM asignado a cada uno de los tres ejes de estado.
2. **Diseño e Implementación de Filtración Compacta:** Diseñar e implementar estructuras compactas candidatas para representar el orden de filtración sin materialización explícita, evaluando alternativas basadas en codificación de enteros de longitud variable [5, Sec. 2.7], arreglos de tamaño variable con punteros muestreados [5, Sec. 3.2] e índices jerárquicos k^d-raster con cotas (min, max).
3. **Diseño e Implementación de Union-Find Dual Compacto:** Diseñar e implementar representaciones compactas para el estado conjuntos disjuntos sobre el grafo dual, integrando la técnica de *pruning* de aristas con bitvectors muy esparsos [5, Sec. 4.4] e índices de partición espacial compactos (k^d-trees) para componentes conexas.
4. **Diseño e Implementación de Matriz Reducida Compacta sobre $\mathbb{F}_2$:** Diseñar e implementar representaciones esparsas compactas para la reducción de matriz sobre $\mathbb{F}_2$ (aplicable a la dimensión intermedia H_1 en 3D o H_1, H_2 en 4D), utilizando codificación adaptativa por zonas [8] con bitvectors esparsos [5, Sec. 4.4] y k^d-trees aumentados con el rango máximo de filtración por subárbol para acelerar la identificación de pivotes (`low(j)`).
5. **Evaluación Experimental y Caracterización de Trade-offs:** Realizar una evaluación experimental exhaustiva midiento el consumo de memoria peak (RSS en MB/GB), tiempo de ejecución desglosado por fases, fallos de caché de último nivel (LLC misses) y fallos de página (*page faults*), construyendo un mapa de crossovers frente a bibliotecas estándar (Flash Cubical, CubicalRipser, GUDHI, DIPHA).

Evaluación

La evaluación de la propuesta se guiará por los siguientes criterios medibles:

- **Criterio de Corrección Estricta:** Para cada dataset evaluado, el diagrama de persistencia generado por la solución compacta debe ser 100 % idéntico al producido por la versión original de Flash Cubical.

- **Métrica de Memoria:** Medición del consumo peak de memoria residente (RSS) mediante scripts de captura de sistema y herramientas de perfilado de memoria. La meta es lograr reducciones sustanciales en el consumo peak de RAM respecto al algoritmo base.
- **Métrica de Tiempo y Rendimiento de Hardware:** Medición del tiempo total de ejecución (*wall-clock time*) desglosado en fases y análisis de rendimiento de memoria usando `perf stat` (midiendo fallos de caché LLC y page faults).

4. Solución Propuesta

La solución propuesta consiste en reconstruir el pipeline del algoritmo de referencia Flash Cubical sustituyendo las estructuras de datos genéricas e ineficientes en memoria por un conjunto de estructuras de datos compactas diseñadas a la medida de cada eje de estado.

Arquitectura y Ejes de Sustitución

El flujo del sistema mantendrá intacto el marco algorítmico de Flash Cubical:

Lookup Table $\rightarrow$ Local Edge Pruning $\rightarrow$ H_0 Dual UF $\rightarrow$ H_{d-1} Dual UF $\rightarrow$ Reducción H_1 ($\mathbb{F}_2$)

Sobre esta arquitectura se sustituirán las estructuras de memoria genéricas por representaciones compactas adaptadas a cada eje de estado. Las estructuras que se describen a continuación son *candidatas*: su elección definitiva dependerá del perfilado de referencia y de los resultados experimentales de cada etapa.

- **Eje 1: Filtración ordenada compacta.** En lugar de materializar en RAM la secuencia explícita de las $2^d \cdot n^d$ celdas del complejo, se explorarán representaciones compactas que exploten la regularidad de la grilla y la dependencia de los valores de celda respecto del máximo de sus vértices incidentes. Entre las alternativas candidatas se consideran:
  - **Representación de secuencias totalmente comprimida [1].** Almacenar la secuencia de celdas ordenadas por filtración usando la estructura de Barbay et al. permitiría navegar el orden de filtración sin materializar la secuencia completa.
  - **Índices jerárquicos k^d-raster.** Un k^d-raster sobre la grilla original almacena cotas $(\text{mín}, \text{máx})$ de intensidad por bloque espacial. Permitiría responder consultas de conteo por rango y emular el ordenamiento por *buckets* navegando el árbol jerárquico al vuelo, evitando materializar la secuencia ordenada completa.
  - **Codificación de diferencias y punteros muestreados [5, Sec. 2.7 y 3.2].** Los deltas de valores de filtración se codificarían con códigos de ancho variable (por ejemplo, γ, δ, VByte o Simple9) acompañados de punteros muestreados que garanticen acceso sublineal sobre la secuencia comprimida.
- **Eje 2: Union-Find dual compacto.** Para calcular H_0 y H_{d-1} mediante Union-Find sobre el grafo dual, la versión original mantiene un arreglo explícito de punteros de padre y rangos. Se explorarán dos líneas candidatas:
  - **Representación comprimida de coleccciones dinámicas de conjuntos disjuntos [1].** Aplicar el resultado de Barbay et al. sobre el grafo dual almacenando la asignación de cada vértice dual a su componente conexa como una secuencia totalmente comprimida.
  - **Integración del pruning con bitvectors esparsos [5, Sec. 4.4].** Tras el *local edge pruning* de Flash Cubical, el subconjunto de aristas duales sobrevivientes podría representarse como un bitvector muy esparso con soporte para `select` en $O(1)$, lo que permitiría agrupar componentes conexas y reducir el número de conjuntos a gestionar.

  - **Particiones mediante k^d-trees.** Dado que las componentes conexas en volúmenes cúbicos tienden a ocupar regiones espacialmente coherentes, se evaluará representar cada componente mediante un k^d-tree compacto, traduciendo las uniones en fusiones de subárboles. La meta es almacenar las fronteras de los conjuntos en espacio sublineal respecto al volumen.

- **Eje 3: Matriz reducida compacta sobre $\mathbb{F}_2$.** Para resolver la dimensión intermedia H_1 en volúmenes 3D, la matriz de borde reducida almacena columnas esparsas sobre $\mathbb{F}_2$. Se estudiará la combinación de las siguientes estructuras candidatas:
  - **Bitvector Elias-Fano muy esparso para la región fija.** Almacenar la parte estática de la matriz reducida (columnas en su estado inicial) en espacio cercano a su entropía.
  - **Diccionario compacto de pivotes.** Permite identificar el pivote inferior `low(j)` de cada columna sin recorrer la estructura completa.
  - **Bitvector muy esparso dinámico para el XOR.** Representa las modificaciones producidas por las sumas de columnas durante la reducción gaussiana.

  Si el perfilado experimental lo justifica, se evaluarán variantes alternativas como k^d-trees aumentados, bitvectors H/L [5, Sec. 4.4], representaciones de listas invertidas totalmente comprimidas [1] o codificación adaptativa por zonas [8].

Aislamiento de Variables y Metodología

Para garantizar la validez científica del trabajo, las sustituciones de estructuras compactas se realizarán de forma incremental y modular (sustituyendo un eje a la vez) dentro de la base de código C++ de Flash Cubical. De este modo, se aislará de forma precisa el porcentaje de ahorro de RAM y la variación del tiempo atribuible a cada estructura de datos específica.

5. Plan de Trabajo (Preliminar)

El plan de trabajo preliminar comprende un periodo de desarrollo de un semestre académico, estructurado en seis etapas secuenciales. La Tabla 1 resume las actividades de cada etapa:

Etapa	Actividades y entregables
1. Replicación y baseline	Compilación e instalación del código fuente oficial de Flash Cubical. Configuración del entorno de evaluación con datasets estándar de la literatura (Bonsai, Head Aneurism, Fuel, Lena, DIV2K). Ejecución de pruebas de corrección y establecimiento de la baseline de métricas (RSS pico y tiempo de ejecución por fase).
2. Perfilado y especificación de operaciones	Perfilado detallado de memoria con herramientas como `heaptrack` y Valgrind Massif sobre Flash Cubical. Cuantificación de la memoria asignada a cada uno de los tres ejes de estado. Extracción del perfil de consultas emitidas por el algoritmo sobre cada eje (`access`, `rank`, `select`, consultas por rango, fusiones de conjuntos).
3. Eje 1: Filtración ordenada compacta	Implementación y evaluación de las estructuras candidatas para la filtración (codificación de enteros de longitud variable, arreglos de tamaño variable e índice k^d-raster). Integración en el módulo de barrido de Flash Cubical. Validación de la igualdad de diagramas de persistencia y medición del ahorro de memoria atribuible al Eje 1.
4. Eje 2: Union-Find dual compacto	Implementación y evaluación de bitvectors muy esparsos y componentes representadas mediante k^d-trees sobre el grafo dual, aprovechando el pruning de aristas. Integración con la dualidad de Alexander en Flash Cubical. Validación de los diagramas de H_0 y H_{d-1} y medición del ahorro de RAM atribuible al Eje 2.
5. Eje 3: Matriz reducida compacta	Implementación y evaluación de una representación compacta para las columnas de la matriz reducida sobre $\mathbb{F}_2$, basada preliminarmente en un bitvector Elias-Fano muy esparso para la región fija de la matriz, un diccionario compacto para identificar pivotes y un bitvector muy esparso dinámico para las operaciones de XOR. Integración con el algoritmo de reducción matricial sobre $\mathbb{F}_2$. Validación final de la exactitud completa del diagrama de persistencia.
6. Evaluación experimental y redacción	Pruebas comparativas integradas frente a Flash Cubical original, CubicalRipser, GUDHI y DIPHA. Medición de rendimiento de hardware con `perf stat` (fallos de caché LLC y *page faults*), identificando puntos de cruce (*crossovers*). Análisis crítico de resultados, elaboración de gráficos comparativos y redacción del documento final de memoria de título.

Tabla 1: Plan de trabajo preliminar por etapas.

Referencias

[1] Barbay, Jérémy, Francisco Claude, Travis Gagie, Gonzalo Navarro y Yakov Nekrich: *Efficient Fully-Compressed Sequence Representations*. Algorithmica, 69(1):232–268, 2014. https://doi.org/10.1007/s00453-012-9726-3.

[2] Boissonnat, Jean Daniel, Karthik C. S. y Sébastien Tavenas: *Building Efficient and Compact Data Structures for Simplicial Complexes*. Algorithmica, 79(2):530–567, 2016, ISSN 1432-0541. http://dx.doi.org/10.1007/s00453-016-0207-y.

[3] Breton, Titouan Le, Karol Szustakowski y Marie Piraud: *Fast Cubical Persistent Homology on 2D and 3D Images via Union-Find, Pruning, and Lookup Tables*. CoRR, abs/2606.04801, 2026. https://doi.org/10.48550/arXiv.2606.04801.

[4] Kaji, Shizuo, Takeki Sudo y Kazushi Ahara: *Cubical Ripser: Software for computing persistent homology of image and volume data*. CoRR, abs/2005.12692, 2020. https://arxiv.org/abs/2005.12692.

[5] Navarro, Gonzalo: *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016, ISBN 978-1-10-715238-0. http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/compact-data-structures-practical-approach?format=HB.

[6] Nunez, Rolando Kindelan: *Topological Data Analysis for Classification of Noisy and High-Dimensional Datasets*. Tesis de Doctorado, Universidad de Chile, Santiago, Chile, 2025.

[7] Wagner, Hubert, Chao Chen y Erald Vuçini: *Efficient Computation of Persistent Homology for Cubical Data*. 2012. https://api.semanticscholar.org/CorpusID:16946295.

[8] Wang, Jianguo, Chunbin Lin, Yannis Papakonstantinou y Steven Swanson: *An Experimental Study of Bitmap Compression vs. Inverted List Compression*. En Salihoglu, Semih, Wenchao Zhou, Rada Chirkova, Jun Yang y Dan Suciu (editores): *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, páginas 993–1008. ACM, 2017. https://doi.org/10.1145/3035918.3064007.