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

ESTRUCTURAS DINÁMICAS PARA LA RESOLUCIÓN EFICIENTE DE CONSULTAS EN BASES DE DATOS DE GRAFOS MEDIANTE QDAGS

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

DANTE ALEJANDRO PIAGGIO CUCOCH

MODALIDAD:
Memoria

PROFESOR GUÍA:
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
2026

1. Introducción

Las bases de datos de grafos se han consolidado como una herramienta fundamental en el manejo de información altamente interconectada, siendo ampliamente utilizadas en la industria y la academia. Un ejemplo claro de su impacto son los grafos de conocimiento abiertos (Open Knowledge Graphs) como Wikidata, los cuales reciben millones de consultas diarias [1]. Sin embargo, a medida que el volumen de datos crece, la ejecución de operaciones relacionales complejas sobre estos grafos se vuelve un desafío computacional significativo.

En particular, la realización de operaciones *join* (cruces de datos) puede resultar extremadamente costosa en términos de tiempo de ejecución. Tradicionalmente, resolver estos *joins* de forma eficiente requiere la creación y el mantenimiento de múltiples índices, lo que conlleva un uso intensivo y a menudo prohibitivo de espacio de almacenamiento adicional. Por lo tanto, contar con mecanismos que permitan resolver estas operaciones de manera eficiente y con un bajo consumo de memoria es una necesidad imperante en el estado actual de las bases de datos de grafos.

Para abordar este problema, la literatura ha propuesto algoritmos y estructuras de datos capaces de ejecutar estas operaciones con garantías de optimalidad en el peor de los casos, conocidos como algoritmos *Worst-Case-Optimal* (WCO). Un ejemplo notable de estos algoritmos es el Leapfrog Triejoin [2]. A pesar de sus garantías teóricas de tiempo, estas aproximaciones sufren del problema de requerir una cantidad considerable de espacio extra para operar de manera óptima.

Es en este contexto donde surgen los *Qdags* (Quadtree-based Directed Acyclic Graphs), una estructura altamente compacta que utiliza Quadtrees comprimidos para ejecutar *joins* óptimos sin almacenamiento redundante [3]. No obstante, la naturaleza de los Qdags es actualmente estática. Esta memoria busca integrar dinamismo en los Qdags para permitir la actualización en tiempo real de la base de datos sin reconstruir la estructura, aprovechando técnicas de supernodos y vectores de bits adaptativos [4].

2. Marco Teórico

Para comprender la evolución hacia los Qdags dinámicos, es necesario definir la jerarquía de representación y los mecanismos de consulta:

1. Grillas y Bitmaps: Una relación con d atributos se representa como un conjunto de puntos en una grilla d-dimensional. Esta grilla se particiona recursivamente conformando un Quadtree, el cual luego se serializa en una secuencia de bits (bitmap) compacta que describe la topología del árbol.

2. Dimensión Virtual y Qdags: Un Qdag eleva virtualmente la dimensión de la grilla mapeando las operaciones de recorrido del Qdag a operaciones de descenso a un hijo en el Quadtree correspondiente. Esto permite representar relaciones complejas de manera altamente compacta mediante la compartición de subárboles idénticos.

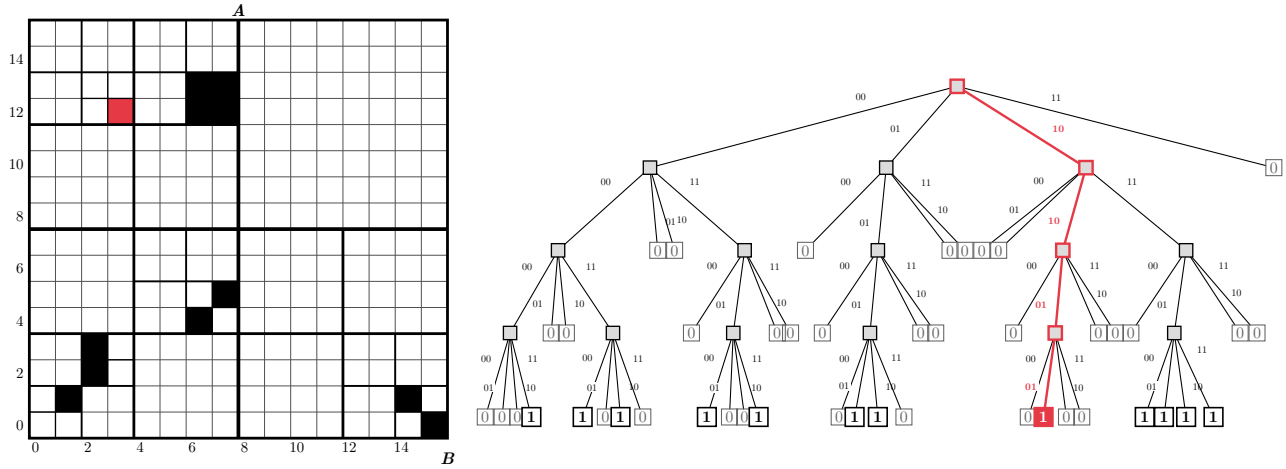
3. Join por Recorrido DFS: La resolución de consultas en los Qdags se basa en un recorrido en profundidad (*Depth-First Search*) simultáneo de las estructuras involucradas. Durante este proceso, se aplica *pruning* (poda) de forma agresiva e inmediata: si en algún punto una de las estructuras indica una región vacía (un bit 0 en el bitmap), se descarta toda esa rama, optimizando drásticamente el espacio de búsqueda.

4. Dinamismo por Supernodos: Siguiendo la técnica propuesta en [5], el dinamismo se puede lograr utilizando un árbol de supernodos, donde cada nodo contiene y administra una parte conexa del Quadtree. Esto permite modificar el bitmap de manera local al agregar o quitar aristas, evitando la invalidación y reconstrucción total de la estructura de datos.

5. Adaptive Bitvectors: Esta técnica permite modificar la estructura del bitmap en tiempo adaptivo a la tasa de actualizaciones respecto a las consultas. Dado que en bases de datos de grafos las consultas suelen ser órdenes de magnitud más frecuentes que las actualizaciones, esta tasa de respuesta resulta ideal para mantener el rendimiento del sistema [4].

A modo de ejemplo práctico sobre la serialización y recorrido espacial, consideremos la relación bidimensional $R(A, B) = \{(0, 15), (1, 1), (1, 14), (2, 2), (3, 2), (4, 6), (5, 7), (12, 3), (12, 6), (12, 7), (13, 6), (13, 7)\}$.

La Figura (a) ilustra la representación espacial de esta relación sobre una grilla de $2^4 \times 2^4$ elementos, donde las celdas en negro corresponden a las tuplas existentes en R y las líneas demarcan la partición jerárquica dictada por el Quadtree subyacente.



(a) Representación espacial.

(b) Estructura del Quadtree.

Figura 1: Representación de la relación $R(A, B)$ y su correspondiente topología jerárquica.

La figura (b) muestra la estructura topológica del Quadtree generado. Los nodos en gris oscuro representan nodos internos, mientras que cada terminal en 1 ubicado en el último nivel indica la presencia de una tupla.

Si se concatena la secuencia de decisiones (etiquetas de las aristas) en el camino descendente desde la raíz hasta una hoja, se obtiene una cadena binaria que codifica las coordenadas de la tupla original utilizando la curva de Z-order (Código Morton). Por ejemplo, el camino resaltado en la figura (b) forma el bit-string 10100101. En esta representación, la primera dimensión (A) está denotada por los bits en posiciones impares, mientras que la segunda dimensión (B) ocupa los bits en posiciones pares. Al decodificar esta cadena, se recuperan las coordenadas $(1100, 0011) = (12, 3)$, lo cual coincide exactamente con la celda resaltada en rojo en la figura (a).

3. Situación Actual

Los Qdags estáticos han demostrado ser eficientes en espacio y óptimos en tiempo para consultas complejas [3]. Sin embargo, su limitación crítica es su incapacidad para manejar inserciones o borrados sin una reconstrucción total.

Actualmente, existen implementaciones parciales que abordan este problema de forma aislada. La tesis de [6] implementó una versión dinámica basada en supernodos, pero dicha estructura no ha sido evaluada ni integrada en el proceso de resolución de consultas *join*. Por otro lado, los *Adaptive Dynamic Bitvectors* [4] cuentan con implementaciones eficientes en C++ que permiten modificar el bitmap directamente, pero su integración en el pipeline de Qdags permanece inexplorada. No se ha definido aún cómo se traducirá la inserción o borrado de una arista en términos de las operaciones lógicas necesarias para mantener la integridad de los Qdags dinámicos durante un proceso de intersección.

4. Objetivos

Objetivo General

Integrar estructuras de datos dinámicas en la arquitectura de los Qdags para permitir la inserción y eliminación eficiente de aristas en bases de datos de grafos, evaluando su rendimiento algorítmico frente a la versión estática tradicional y frente a las operaciones de join.

Objetivos Específicos

1. Establecer un entorno de experimentación temprana integrando las implementaciones existentes de Qdags estáticos y la base dinámica de supernodos.
2. Diseñar e implementar la integración de *Adaptive Dynamic Bitvectors* en el pipeline de resolución de consultas de los Qdags basándose en la implementación ya existente.
3. Construir una interfaz de programación (*API*) unificada que exponga de manera transparente las operaciones de consulta y actualización para las distintas implementaciones.
4. Diseñar y ejecutar un marco de evaluación experimental para comparar exhaustivamente las versiones dinámicas frente a la estática en términos de tiempo de ejecución, uso de memoria y latencia de actualización.
5. Liberar una implementación pública, documentada y eficiente de los mejores Qdags dinámicos resultantes del trabajo realizado.

Evaluación

El éxito del trabajo se medirá a través de un análisis empírico y comparativo. Se utilizarán *datasets* estándar de grafos (o simulaciones representativas de cargas de trabajo como Wikidata) para evaluar tres métricas fundamentales: la eficiencia en espacio (memoria RAM consumida por la estructura), la eficiencia en tiempo para consultas de solo lectura (comparada contra la versión estática), y el throughput (operaciones por segundo) para las cargas de trabajo que involucren inserciones y borrados. La evaluación determinará el trade-off entre el costo extra de mantener la dinamicidad de la estructura y la ganancia al evitar la reconstrucción total del índice.

5. Solución Propuesta

La solución propuesta consiste en una implementación a bajo nivel en C++, orientada a la eficiencia computacional. El uso de C++ es fundamental para aprovechar el código base existente de Qdags estáticos, el dinamismo del arbol de supernodos y la interfaz de los Adaptive Dynamic Bitvectors, optimizando el tiempo de desarrollo y el control de recursos.

El trabajo se dividirá en dos grandes partes técnicas:

1. Qdags basados en supernodos: Se adaptarán los algoritmos de navegación e intersección DFS para que operen sobre la estructura de supernodos existente, permitiendo modificaciones locales en el bitmap sin invalidar subárboles compartidos.
2. Qdags basados en Adaptive Dynamic Bitvectors: Se integrará una capa de almacenamiento basada en vectores adaptativos, traduciendo las operaciones lógicas de descenso en el Quadtree a operaciones de *rank* y *select* dinámicas.

Finalmente, se desarrollará una interfaz de línea de comandos (CLI) y una biblioteca de funciones que abstraiga la complejidad interna de las estructuras. Esto permitirá al usuario instanciar el grafo, ejecutar comandos de inserción (insert(u, v)), borrado (delete(u, v)) y consultas complejas de manera estandarizada, ya sea mediante el paso de argumentos por consola o mediante llamadas directas a la API de la biblioteca. Este diseño garantizará que la lógica de evaluación sea agnóstica a la variante de Qdag seleccionada internamente, facilitando las pruebas de rendimiento comparativas.

6. Plan de Trabajo (Preliminar)

El desarrollo de esta memoria está planificado para ejecutarse a lo largo de 21 semanas. Considerando que restan aproximadamente 8 semanas del semestre académico en curso, este plan contempla iniciar la fase de estudio, familiarización con el código e implementación base durante este período, extendiendo el desarrollo principal, la evaluación y la redacción del informe hacia el semestre siguiente. Para asegurar la factibilidad técnica y contar con holgura ante imprevistos, se ha asignado tiempo adicional a cada hito respecto a una planificación estándar (una semana más en cada punto):

1. Semanas 1-3: Familiarización y entorno base. Estudio profundo del código existente en C++ (Qdags y tesis de David) y ejecución de pruebas de concepto para asegurar su correcto funcionamiento.
2. Semanas 4-7: Integración de Supernodos. Adaptación del algoritmo de *join* para soportar la estructura de supernodos dinámicos.
3. Semanas 8-11: Integración de Adaptive Bitvectors. Implementación de la variante basada en vectores adaptativos en el pipeline de Qdags.
4. Semanas 12-14: Desarrollo de la Biblioteca y CLI. Construcción de la interfaz de línea de comandos y la biblioteca de funciones que unifique las operaciones. Preparación de los scripts y conjuntos de datos (grafos) para los experimentos.
5. Semanas 15-17: Evaluación Experimental, Optimización e Inicio de Redacción. Ejecución exhaustiva de las pruebas de rendimiento (tiempo/espacio) y análisis de cuellos de botella. Se inicia la redacción de los capítulos técnicos que describen las implementaciones previas para adelantar el informe final.
6. Semanas 18-21: Redacción del Informe Final. Sistematización de los resultados obtenidos, redacción del documento de memoria, conclusiones y preparación de la defensa y publicación del código fuente documentado.

Referencias

[1] G. Lederrey, D. Causse, y L. Pintscher, «Wikidata Query Service: Where are we? Where is it going?». [En línea]. Disponible en: https://www.wikidata.org/wiki/Event:Data_Reuse_Days_2025

[2] T. L. Veldhuizen, «Triejoin: A simple, worst-case optimal join algorithm». [En línea]. Disponible en: https://doi.org/10.48550/arXiv.1210.0481

[3] D. Arroyuelo, G. Navarro, J. L. Reutter, y J. Rojas-Ledesma, «Optimal Joins using Compressed Quadtrees», *ACM Transactions on Database Systems*, vol. 37, n.º 4, ago. 2018, doi: 10.1145/1122445.1122456.

[4] G. Navarro, «Practical Adaptive Dynamic Bitvectors», *Software Practice and Experience*, vol. 55, n.º 9, pp. 1539-1559, 2025.

[5] D. Arroyuelo, G. de Bernardo, T. Gagie, y G. Navarro, «Faster Dynamic Compressed d-ary Relations», en *Proc. 26th International Symposium on String Processing and Information Retrieval (SPIRE)*, en LNCS 11811. 2019, pp. 419-433.

[6] D. Véliz, «Estructuras compactas dinámicas más eficientes para bases de datos de grafos con atributos», Memoria para optar al título de Ingeniero Civil en Computación, Santiago, Chile, 2021.