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

ESTUDIO DE TAMAÑOS MÍNIMOS DE RINGS Y ORDER GRAPHS PARA JOINS ÓPTIMOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN CIENCIAS DE LA COMPUTACIÓN

JAVIER IGNACIO OLIVA SILVA

PROFESOR GUÍA:
GONZALO NAVARRO BADINO

MIEMBROS DE LA COMISIÓN:
CLAUDIO DOMINGO GUTIÉRREZ GALLARDO
SEBASTIÁN CAMILO FERRADA ALIAGA
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
2026

Resumen

En esta memoria se estudia un problema reciente relacionado con las estructuras denominadas *Rings* y *Order Graphs*, las cuales surgen en el contexto de la optimización temporal y espacial de consultas *join* en bases de datos relacionales. La motivación del trabajo radica en que, aunque los algoritmos de *join* óptimos en el peor caso (*Worst-Case Optimal Joins*, WCO) ofrecen garantías asintóticas favorables en tiempo de ejecución, las estructuras auxiliares necesarias para soportar múltiples órdenes de atributos pueden crecer muy rápidamente con la aridad de la relación, limitando su aplicabilidad práctica. En este contexto, *Rings* y *Order Graphs* aparecen como estructuras compactas que permiten soportar dichos órdenes de atributos con menor uso de espacio.

El trabajo se enfoca principalmente en el desarrollo de algoritmos para la búsqueda de *Rings* y *Order Graphs* de tamaños pequeños para relaciones de distintas dimensiones, con el objetivo de estudiar y aproximar sus tamaños mínimos posibles. El estudio se concentra en el análisis combinatorio de estas estructuras y no en los detalles de implementación de motores de bases de datos. En particular, se consideran tanto algoritmos de búsqueda exhaustiva, orientados a reproducir soluciones óptimas conocidas en dimensiones pequeñas, como algoritmos de búsqueda no exhaustiva, diseñados para explorar dimensiones donde la explosión combinatoria vuelve impracticable la enumeración completa.

Metodológicamente, el trabajo combina diseño teórico de algoritmos, implementación eficiente en `C++` sobre CPU y evaluación experimental. En el caso de conjuntos de *Rings*, el problema se formula como una instancia de *Set Cover*, utilizando representaciones mediante *bitmasks* para acelerar la verificación y la poda. En el caso de *Order Graphs*, debido a la complejidad del caso general, el estudio se concentra en *Cyclic Order Graphs*, para los cuales se desarrollan algoritmos de verificación, búsqueda exhaustiva y búsqueda no exhaustiva. Entre estas últimas se consideran búsqueda local, algoritmos de aproximación, heurísticas voraces y *Simulated Annealing*, entendido como una técnica de optimización combinatoria sobre espacios discretos con múltiples óptimos locales.

Los resultados muestran que la búsqueda exhaustiva permite reproducir soluciones óptimas conocidas en dimensiones pequeñas, validando la correctitud de la formulación y de las implementaciones. Sin embargo, también evidencian una explosión combinatoria severa, que vuelve inviable esta estrategia como enfoque principal para dimensiones mayores. En ese contexto, los métodos no exhaustivos permiten explorar regiones del espacio de búsqueda inaccesibles para la enumeración completa. Entre ellos, *Simulated Annealing* fue la técnica que produjo los mejores resultados empíricos: para *Cyclic Order Graphs* de dimensión $d = 6$ se obtuvo una solución completa de tamaño 51 y una solución incompleta de tamaño 42 a la

que solo le faltan dos restricciones por satisfacer, resultados muy cercanos al estado del arte en un horizonte de una semana de ejecución.

La memoria no logra mejorar las cotas conocidas ni avanzar el estado del arte, pero sí genera evidencia computacional útil sobre el comportamiento del problema, la presencia de óptimos locales y la efectividad relativa de distintas estrategias de búsqueda. Además, se destaca que ciertas cotas inferiores válidas para ciclos no se extienden automáticamente a *Order Graphs* generales con ramificación, lo que refuerza la necesidad de distinguir ambos casos en investigaciones futuras. En conjunto, el trabajo proporciona una base metodológica y experimental para continuar el estudio de estas estructuras, orientando el esfuerzo futuro hacia metaheurísticas más robustas, paralelización y el análisis de *Order Graphs* generales.

A mi familia, por acompañarme en cada etapa de este camino. Gracias por su paciencia y cariño.

Tabla de Contenido

1. Introducción **1**

1.1. Contexto y motivación 1

1.2. Formulación del problema de investigación 2

1.3. Objetivo general 2

1.4. Objetivos específicos 3

1.5. Alcance y limitaciones del estudio 3

1.6. Estructura del documento 4

2. Marco teórico y estado del arte **6**

2.1. Marco teórico y estado del arte 6

2.2. Bases de datos relacionales y operadores de *join* 6

2.3. Modelos de costo y planificación de *joins* 8

2.4. *Rings* y *Order Graphs* para *joins* 9

2.5. Estudio de tamaños mínimos de *Rings* y *Order Graphs* para *joins* óptimos . 10

2.6. Algoritmos y enfoques relevantes de la literatura 12

2.7. Síntesis del estado del arte y brechas de investigación 15

3. Metodología y definición formal del problema **17**

3.1. Formulación del problema 17

3.2. Enfoque metodológico 20

3.3. Métricas y criterios de evaluación 21

4. Diseño e implementación de algoritmos exhaustivos **22**

4.1. Conjuntos de *Rings* CBTW-completos 23

4.2. Conjuntos de *Cyclic Order Graphs* completos 28

4.3. Algoritmos de búsqueda no exhaustiva 31

4.3.1. $O\left(\lg\left(\frac{(d-1)!}{2}\right)\right)$-aproximación para conjuntos de *Rings* CBTW-completos 32

4.3.2. Heurísticas para movimientos locales 32

4.3.3. Búsquedas Locales 35

4.3.4. *Simulated Annealing* 36

5. Implementación y entorno experimental **42**

5.1. Etapas de implementación 42

5.2. Arquitectura de la solución 42

5.3. Entorno de ejecución y protocolo experimental 43

5.3.1. Hardware 43

5.3.2. Decisiones técnicas 43

5.3.3. Experimentación 43

6. Resultados y discusión **45**

6.1. Resultados de búsqueda exhaustiva en dimensiones pequeñas 45

6.2. Resultados de Búsqueda no Exhaustiva 46

6.2.1. Búsqueda Local 46

6.3. Simulated Annealing 48

6.4. Limitaciones de Enfoques Voraces 49

6.5. Discusión general 49

7. Conclusiones y trabajo futuro **50**

7.1. Conclusiones generales 50

7.2. Cumplimiento del objetivo general y de los objetivos específicos 51

7.3. Aportes del trabajo 52

7.4. Limitaciones del trabajo 53
7.5. Trabajo futuro 53
7.5.1. Cotas inferiores para *Order Graphs* generales 54
7.5.2. Construcciones que aprovechen ramificación 54
7.5.3. Mejora de estrategias de búsqueda no exhaustiva 54
7.5.4. Paralelización y uso de hardware especializado 54
7.5.5. Refinamiento de cotas y certificación de soluciones 55
7.5.6. Integración con planificación de consultas 55
7.6. Conclusión final 55

Bibliografía **57**

Índice de Tablas

2.1. Tamaños mínimos de índices de orden para soportar algoritmos wcoj según capacidades del índice (adaptado de [4]). Las cotas superiores marcadas con asterisco no fueron obtenidas por búsqueda exhaustiva, por lo que podrían existir valores menores. Los valores en rojo corresponden a cotas superiores actualizadas. 11

6.1. Cyclic Order Graph completos 46

6.2. Mejores tamaños obtenidos para *Cyclic Order Graphs* según vecindad y dimensión. 46

6.3. Conjunto de *Rings* BCTW-completo encontrado para $d = 6$. 47

Índice de Ilustraciones

3.1. Ejemplo de Ring de dimensión $d = 6$. 18

3.2. Ring $r = (1, 2, 3, 4)$ (d=4): permutaciones generadas y no CBW-completo . . 18

3.3. Ring $d = 4$: ejemplo de CBTW-completitud (caso satisfecho y no satisfecho) 19

3.4. Order graph $d = 4$ completo: nodos como órdenes y aristas como *move-to-front* 20

4.1. Vecindad $\mathcal{N}_1$: intercambio de dos vértices adyacentes en un *Ring*. 33

4.2. Vecindad $\mathcal{N}_2$: traslación de un vértice en un *Ring*. 33

4.3. Vecindad $\mathcal{N}_3$: reverso de una sección contigua en un *Ring*. 34

4.4. Vecindad $\mathcal{N}_{4,3}$: permutación de una subsecuencia contigua de largo l (en el ejemplo, $l = 3$). 34

Índice de Algoritmos

1. Enumeración de los *Rings* canónicos de dimensión d (§ 4.1). 23
2. Cómputo de la máscara de cobertura Mask(r) de un *Ring* (§ 4.1). 25
3. LowerBound usado por la poda del Algoritmo 4. 28
4. Búsqueda exhaustiva de un conjunto de *Rings* CBTW-completo de tamaño mínimo (§ 4.1). 37
5. Verificación de completitud de un *Cyclic Order Graph* en $O(m \cdot d)$ (§ 4.2). . . 38
6. Búsqueda exhaustiva de un *Cyclic Order Graph* completo de tamaño m (§ 4.2), aprovechando las simetrías y, opcionalmente, las Conjeturas 4.1 y 4.2. 39
7. Aproximación *greedy* para CBTW vía *Set Cover* (§ 4.3.1). 40
8. Búsqueda local por mejor vecino (§ 4.3.3), con crecimiento de la estructura cuando se estanca con $E > 0$. 40
9. *Simulated Annealing* con reinicio y crecimiento estructural (§ 4.3.4). 41

Capítulo 1

Introducción

1.1. Contexto y motivación

En el ámbito de las bases de datos, las consultas de tipo *join* son fundamentales para combinar información procedente de múltiples relaciones (tablas). A medida que los volúmenes de datos crecen y las aplicaciones se ejecutan sobre bases de datos masivas, se vuelve crítico contar con algoritmos de *join* que sean eficientes tanto en tiempo de ejecución como en uso de memoria. En este contexto, los algoritmos de *join* óptimos en el peor caso (*worst-case optimal joins*, WCO) han cobrado especial relevancia, pues garantizan tiempos de ejecución asintóticamente óptimos para cualquier consulta posible sobre una instancia dada [4].

El diseño de algoritmos WCO ha estado tradicionalmente orientado a mejorar el tiempo de ejecución, mientras que el espacio requerido por estas técnicas ha recibido menos atención. En particular, varios algoritmos WCO existentes requieren estructuras auxiliares cuyo tamaño crece de manera exponencial [4] con la aridad de la relación. En este contexto, la aridad de una relación corresponde al número de atributos o columnas que esta posee. Este crecimiento limita la aplicabilidad práctica de estas técnicas, incluso para consultas sobre relaciones de dimensión moderada.

En respuesta a este problema, se han propuesto estructuras de datos específicas que permiten implementar algoritmos WCO dadas las capacidades del índice con un uso de espacio más compacto. Entre ellas destacan los *Rings* y los *Order Graphs* [4], introducidos recientemente en el contexto de *joins* sobre bases de datos de grafos. De manera general, estas estructuras codifican en forma compacta órdenes de atributos que permiten ejecutar *joins* WCO utilizando poco espacio adicional, manteniendo al mismo tiempo garantías asintóticas en tiempo de ejecución en el peor caso. Un orden de atributos puede entenderse como una permutación de los atributos de una relación que determina el orden en que estos se consideran durante la evaluación de un *join*. Este concepto es relevante porque algoritmos como *Leapfrog Triejoin* dependen del orden en que se accede y vincula cada atributo, y distintos órdenes pueden afectar tanto la factibilidad de uso de ciertos índices como el rendimiento práctico del algoritmo.

No obstante, al tratarse de un área de investigación reciente, la comprensión que se tiene

sobre sus tamaños mínimos y sobre cómo construir instancias óptimas es todavía limitada, especialmente para aridades mayores. El interés en estas aridades radica en que el número de posibles órdenes de atributos crece factorialmente con la dimensión, lo que vuelve especialmente desafiante la construcción de estructuras compactas y el estudio de su optimalidad.

1.2. Formulación del problema de investigación

El tamaño de un *Ring* o de un *Order Graph* se mide, en términos generales, a partir de la cantidad de elementos estructurales necesarios para representarlo, tales como nodos, aristas o componentes equivalentes según la estructura considerada. Dadas las capacidades de un índice —en este trabajo se consideran índices bidireccionales e índices bidireccionales con *trie switching* [4]— interesa conocer cuál es la estructura de menor tamaño que permite soportar algoritmos WCO sobre una relación de aridad d, o bien obtener estructuras más pequeñas que las actualmente reportadas en el estado del arte.

Para aridades pequeñas ($d \leq 6$) se conocen estructuras óptimas o cotas muy ajustadas, tanto para conjuntos de *Rings* como para *Order Graphs*. En cambio, para aridades mayores ($d \geq 7$) solo se dispone de cotas inferiores y superiores relativamente amplias, y se desconoce si las mejores estructuras conocidas son realmente óptimas o si las cotas disponibles todavía pueden ajustarse.

El problema al que se enfrenta este trabajo puede describirse, de manera informal, del siguiente modo:

> Dada una aridad d, ¿cómo diseñar e implementar algoritmos que permitan explorar el espacio de *Rings* y *Order Graphs* que soportan *joins* WCO, de manera de estudiar y aproximar sus tamaños mínimos posibles, generando nueva evidencia computacional sobre estas estructuras?

Este problema es, en esencia, un problema de optimización combinatoria de gran escala: el número de posibles órdenes de atributos crece factorialmente con d, y el espacio de posibles *Rings* y *Order Graphs* se vuelve rápidamente intratable para una búsqueda exhaustiva directa. En esta memoria, se entiende por búsqueda exhaustiva aquella que recorre sistemáticamente todo el espacio de soluciones factibles dentro de una aridad dada, mientras que la búsqueda no exhaustiva corresponde al uso de heurísticas o metaheurísticas que exploran solo una parte de dicho espacio. Por ello, el desarrollo de algoritmos exhaustivos y no exhaustivos que permitan explorar este espacio de manera sistemática y eficiente constituye el foco central de la memoria.

1.3. Objetivo general

En coherencia con el problema planteado y con los avances ya obtenidos, el objetivo general de esta memoria se formula de la siguiente manera:

Objetivo general: Diseñar, implementar y evaluar algoritmos de búsqueda exhaustiva y no exhaustiva para estudiar los tamaños de *Rings* y *Order Graphs* que permiten algoritmos WCO, con y sin *trie switching*, para distintas aridades, generando evidencia computacional que contribuya a caracterizar sus tamaños mínimos y las limitaciones de las técnicas actuales.

Esta formulación enfatiza el estudio sistemático y la generación de evidencia computacional, sin exigir como objetivo formal la obtención de nuevas cotas teóricas globales, lo que correspondería a un trabajo de mayor alcance.

1.4. Objetivos específicos

A partir del objetivo general, se establecen los siguientes objetivos específicos, que corresponden a las metas efectivamente abordadas en la memoria:

1. Diseñar algoritmos de búsqueda exhaustiva y no exhaustiva para el estudio de tamaños mínimos de conjuntos de *Rings* que permitan algoritmos WCO con y sin *trie switching*, capaces de explorar el espacio de soluciones factibles.
2. Diseñar algoritmos de búsqueda exhaustiva y no exhaustiva para el estudio de tamaños mínimos de *Order Graphs* que soporten *joins* WCO con y sin *trie switching*, permitiendo comparar estas estructuras con los conjuntos de *Rings* desde el punto de vista del espacio requerido.
3. Implementar de forma eficiente los algoritmos anteriores en arquitecturas secuenciales (CPU), de modo que puedan ejecutarse en tiempos razonables para las aridades de interés y produzcan resultados susceptibles de análisis experimental.

La obtención de nuevas cotas teóricas globales para los tamaños óptimos de estas estructuras, aunque motivadora y deseable, se considera fuera del conjunto de objetivos específicos formales de esta memoria.

1.5. Alcance y limitaciones del estudio

El alcance del trabajo se centra en el análisis combinatorio y computacional de los tamaños de *Rings* y *Order Graphs* que permiten algoritmos WCO, en el marco teórico definido por la literatura reciente sobre *joins* óptimos en el peor caso. En particular, la memoria se focaliza en:

- El diseño de algoritmos de búsqueda, exhaustiva y no exhaustiva, orientados a explorar el espacio de posibles estructuras para aridades moderadas.
- La implementación de estos algoritmos en CPU, con énfasis en la eficiencia práctica y en la capacidad de generar resultados en tiempos de ejecución acotados.

- La obtención y análisis de resultados experimentales que permitan comparar diferentes estrategias y caracterizar el comportamiento de los tamaños obtenidos para distintas aridades.

Por otro lado, el trabajo presenta las siguientes limitaciones:

- **Limitación en la aridad y tamaño de las instancias:** debido al crecimiento combinatorio del espacio de búsqueda, las estrategias de búsqueda exhaustiva solo son viables para aridades pequeñas, mientras que para aridades mayores se recurre a métodos no exhaustivos que no garantizan optimalidad.
- **Ausencia de nuevas demostraciones de cotas generales:** la memoria no pretende, como objetivo formal, demostrar nuevas cotas teóricas globales para todas las aridades d, sino aportar evidencia computacional que pueda servir de base a investigaciones teóricas posteriores.
- **Ámbito de aplicación restringido:** el estudio se realiza sobre el modelo abstracto de *Rings* y *Order Graphs* para algoritmos WCO, sin abordar la integración directa de las estructuras obtenidas en sistemas concretos de gestión de bases de datos ni la evaluación sobre cargas de trabajo reales.

Estas decisiones permiten acotar el trabajo, manteniendo al mismo tiempo un aporte relevante dentro de la línea de investigación sobre algoritmos WCO y estructuras de datos asociadas.

1.6. Estructura del documento

El resto de la memoria se organiza de la siguiente manera: en el Capítulo 2 se presentan los conceptos fundamentales de bases de datos relacionales, *joins* y algoritmos WCO, junto con la definición formal de *Rings* y *Order Graphs*; además, se revisan los resultados conocidos sobre sus tamaños mínimos y los principales trabajos relacionados en optimización combinatoria y generación de estructuras óptimas. En el Capítulo 3 se detalla el tipo de estudio y el enfoque metodológico adoptado, se formula con precisión el problema de optimización asociado al tamaño de *Rings* y *Order Graphs*, se definen las métricas de evaluación y se expone la estrategia general de diseño de algoritmos, tanto exhaustivos como no exhaustivos. El Capítulo 4 describe en detalle los algoritmos propuestos, junto con sus variantes deterministas, aleatorias y heurísticas, y analiza sus propiedades de correctitud y su complejidad temporal y espacial. El Capítulo 5 presenta la arquitectura de la solución implementada, las decisiones de diseño de software y estructuras de datos, el entorno de ejecución, así como el protocolo experimental, la generación de instancias y las métricas utilizadas para evaluar el desempeño de los algoritmos. En el Capítulo 6 se muestran los resultados obtenidos para distintas aridades, diferenciando entre los casos donde es posible realizar búsqueda exhaustiva y aquellos en que se emplean métodos no exhaustivos; además, se discute la calidad de las soluciones encontradas, los tiempos de ejecución, las limitaciones observadas y su relación con

el estado del arte. Finalmente, el Capítulo 7 sintetiza los principales aportes de la memoria, evalúa el grado de cumplimiento del objetivo general y de los objetivos específicos, y propone líneas de trabajo futuro, tanto en el desarrollo teórico de nuevas cotas como en la mejora de algoritmos y en su eventual integración en sistemas de bases de datos.

Capítulo 2

Marco teórico y estado del arte

2.1. Marco teórico y estado del arte

En este capítulo se revisan los conceptos fundamentales y los principales trabajos relacionados con el estudio de tamaños mínimos de *Rings* y *Order Graphs* para la obtención de estructuras que permitan *joins* óptimos en el peor caso. La revisión se organiza en torno a cuatro ejes: (i) fundamentos de bases de datos relacionales y operadores de *join*, (ii) modelos de costo y planificación, (iii) estructuras específicas basadas en grafos —principalmente *Rings* y *Order Graphs*— para soportar algoritmos de *joins* óptimos en el peor caso, y (iv) trabajos recientes que analizan tamaños mínimos de estas estructuras y su impacto en la planificación. El material se basa exclusivamente en los artículos y documentos proporcionados, complementando, cuando es necesario, con nociones básicas marcadas explícitamente como conocimiento general.

2.2. Bases de datos relacionales y operadores de *join*

Desde una perspectiva de conocimiento general, un esquema relacional se define como un conjunto de relaciones $R_1, R_2, \ldots, R_m$, donde cada relación R_i, para $i \in \{1, 2, \ldots, m\}$, se caracteriza por un conjunto de atributos. Una instancia de base de datos D asigna a cada relación R_i una relación finita $R_i(D)$ sobre sus respectivos atributos. Una consulta de *join* natural puede representarse como una expresión

$$Q = R_1(a_{11}, \ldots, a_{1r_1}) \bowtie \cdots \bowtie R_m(a_{m1}, \ldots, a_{mr_m}),$$

donde Q denota la consulta, D una instancia de base de datos, $Q(D)$ el resultado de evaluar la consulta sobre D, y $|Q(D)|$ la cardinalidad de dicho resultado. En particular, $Q(D)$ corresponde al conjunto de tuplas sobre la unión de atributos de la consulta, tales que su proyección sobre los atributos de cada relación R_i pertenece a $R_i(D)$.[1]

[1]Esta formulación y la notación de *joins* naturales son consistentes con las definiciones empleadas en los trabajos teóricos revisados, pero el nivel de detalle aquí expuesto corresponde a conocimiento general; véase, por ejemplo, [7].

El trabajo de Atserias, Grohe y Marx [7] estudia precisamente este tipo de consultas de *join*, modelándolas y vinculando el tamaño del resultado con parámetros combinatorios como el número de cubrimiento fraccional de aristas. En su formulación, una consulta de *join* Q se asocia a un hipergrafo $H(Q)$ cuyo conjunto de vértices coincide con el conjunto de atributos y cuyas hiperaristas corresponden a los conjuntos de atributos de cada relación. Esto permite caracterizar el tamaño máximo del resultado $Q(D)$ en función del tamaño total $|D|$ de la base de datos y del número de cubrimiento fraccional $\rho^*(Q)$, obteniendo cotas superiores e inferiores ajustadas en el peor caso [7].

Ngo et al. [15] desarrollan esta línea construyendo algoritmos de *join* que alcanzan dichas cotas en términos de complejidad, lo que enmarca formalmente la noción de algoritmos de *join* óptimos en el peor caso (*worst-case optimal joins*, WCOJ). Su trabajo demuestra la existencia de algoritmos para consultas de *join* naturales cuya complejidad temporal está acotada, en el peor caso, por la cota de Atserias, Grohe y Marx, proporcionando una conexión directa entre la teoría de hipergrafos y la implementación de algoritmos de *join*.

El artículo de Atserias, Grohe y Marx [7] aborda el problema de estimar el tamaño del resultado de consultas de *join* y de diseñar planes de ejecución que sean óptimos (hasta factores polinomiales) en el modelo de peor caso. El enfoque consiste en asociar cada consulta a un hipergrafo y utilizar cubrimientos fraccionales para obtener una cota superior

$$|Q(D)| \leq |D|^{\rho^*(Q)},$$

que resulta ser ajustada, ya que también se muestra la existencia de instancias que alcanzan, hasta factores polinómicos, dicho valor. Además, se construyen planes de tipo *join-project* que se evalúan en tiempo $O(|Q|^2 \cdot |D|^{\rho^*(Q)+1})$, y se demuestra que, en el peor caso, estos planes son asintóticamente mejores que cualquier plan compuesto únicamente por *joins* binarios.

Desde una perspectiva crítica, la principal fortaleza del trabajo radica en proporcionar una caracterización combinatoria precisa de la complejidad del *join*, con resultados de optimalidad en términos de clases de consultas y planes de ejecución. El marco de hipergrafos y cubrimientos fraccionales se ha convertido en referencia para la investigación posterior en algoritmos de *joins* óptimos en el peor caso. Sin embargo, el trabajo se centra en modelos teóricos y no aborda de forma detallada cómo estas cotas se traducen en decisiones prácticas de planificación de *joins* dentro de optimizadores reales, ni cómo se representan estructuras apoyadas en órdenes de atributos como los *Rings* u *Order Graphs*. Esta laguna abre espacio para trabajos posteriores que conectan estas cotas con estructuras de datos compactas y estrategias de planificación concretas, como es el caso de la literatura sobre *Rings* y *Order Graphs*.

Ngo et al. [16] presentan un algoritmo para *joins* naturales que es óptimo en el peor caso en términos de la cota de Atserias, Grohe y Marx. El algoritmo está diseñado para procesar *joins* sobre múltiples relaciones, evitando las limitaciones de los planes tradicionales *select-project-join* y proporcionando una prueba constructiva de la cota de cubrimiento fraccional.

Entre sus fortalezas destaca la conexión directa entre teoría y algoritmo: se demuestra que es posible alcanzar en tiempo de ejecución la cota de tamaño del resultado, posicionando a estos algoritmos como referencia para toda la línea de trabajo posterior en WCOJ. Como limitación, el trabajo se centra en consultas conjuntivas completas y no aborda explícitamente

aspectos de implementación en motores de bases de datos ni las restricciones de espacio asociadas a indexar múltiples órdenes de atributos. Asimismo, no se discute la estructura de búsqueda de planes (por ejemplo, árboles de *join* frente a grafos de órdenes), lo que motiva desarrollos posteriores en los que se introducen estructuras como *Rings* y *Order Graphs* para organizar el espacio de planes de *join* sobre colecciones de órdenes de atributos.

2.3. Modelos de costo y planificación de *joins*

Los modelos de costo para la ejecución de consultas de *join* son una pieza clave de los optimizadores de consultas. En la literatura proporcionada, estos modelos pueden agruparse en dos niveles: modelos teóricos de complejidad en el peor caso (como los derivados de la cota de Atserias, Grohe y Marx [7]) y modelos operacionales empleados en motores concretos para seleccionar entre distintas estrategias de *join*.

Atserias, Grohe y Marx (AGM) distinguen explícitamente entre planes basados solo en *joins* binarios (*join plans*) y planes que incluyen proyecciones intermedias (*join-project plans*), mostrando que estos últimos pueden ofrecer mejoras superpolinomiales en el tiempo de ejecución bajo el modelo de peor caso [7]. Asimismo, introducen el número de cubrimiento fraccional $\rho^*(Q)$ como parámetro clave para evaluar la complejidad de un plan *join-project* y proporcionan planes cuya complejidad está acotada por $|D|^{\rho^*(Q)+1}$.

Más adelante, el artículo analiza planes de ejecución haciendo uso de programación lineal para estimar el tamaño de *joins* bajo restricciones sobre los tamaños de las relaciones [7]. De esta forma, la planificación de *joins* se vincula de nuevo con parámetros del hipergrafo subyacente, en particular mediante problemas de optimización que maximizan o minimizan ciertas funciones lineales bajo restricciones derivadas de cubrimientos fraccionales.

En el contexto de algoritmos WCOJ, la planificación se complica por la necesidad de escoger órdenes de atributos adecuados. El trabajo sobre *Rings* destaca que, para algoritmos como *Leapfrog Triejoin* (LTJ) [19], el orden de eliminación de variables tiene un impacto crítico en el rendimiento, incluso si todos los órdenes son óptimos en el peor caso desde un punto de vista asintótico [3]. Este trabajo enfatiza que, en la práctica, es deseable poder seleccionar dinámicamente el orden de atributos durante la fase de planificación de consultas, lo que a su vez requiere disponer de índices que soporten eficientemente múltiples órdenes de atributos.

Freitag et al. estudian la integración de algoritmos WCOJ en un motor relacional de altas prestaciones, analizando variantes como *Hash Trie Join* (HTJ) y su combinación con operadores clásicos [9]. En este contexto, los modelos de costo se basan tanto en estimaciones de cardinalidad —influenciadas por las cotas AGM y la estructura del *join*— como en características físicas del motor, como la capacidad de aprovechar cachés y almacenamiento columnar.

En síntesis, la literatura revisada sugiere que los modelos de costo para WCOJ combinan: (i) parámetros teóricos como $\rho^*(Q)$ extraídos del hipergrafo de la consulta, (ii) estimaciones de cardinalidad adaptadas a datos reales o sintéticos, y (iii) costos físicos de acceso e intersec-

ción de índices multiorden. No obstante, los documentos proporcionados no detallan modelos de costo específicos para *Rings* u *Order Graphs*; más bien, los consideran como estructuras subyacentes cuya eficiencia se refleja indirectamente en las evaluaciones experimentales. Cuando no se dispone de detalles concretos, este capítulo se limita a señalar la existencia de tales modelos sin especular sobre sus fórmulas internas.

2.4. *Rings* y *Order Graphs* para *joins*

El trabajo sobre *Rings* introduce una estructura diseñada para soportar algoritmos WCOJ sobre grafos, con un uso de espacio reducido [3]. La motivación parte de la observación de que, para garantizar la complejidad óptima en el peor caso con algoritmos del tipo LTJ, sería necesario disponer de índices ordenados en todas las permutaciones de los atributos de cada relación, lo cual implica $d!$ índices para una relación de aridad d, algo prohibitivo incluso para aridades moderadas.

Aunque parte importante de la motivación original proviene de *joins* sobre grafos o sobre datos RDF, en esta memoria el problema se abstrae al caso general de relaciones de aridad d, donde d representa el número de atributos de la relación. En ese marco, *Rings* y *Order Graphs* se estudian como estructuras combinatorias capaces de representar, de manera compacta, múltiples órdenes de atributos útiles para WCOJ.

Definición y contexto de los *Rings*

A partir de la literatura revisada, un *Ring* puede describirse, en términos informales, como una representación cíclica de las tuplas de una relación, organizada de tal manera que cada rotación de la secuencia corresponde a un orden de atributos potencialmente útil para un algoritmo de *join* óptimo en el peor caso. En la formulación de Arroyuelo et al. [3], las tuplas de la relación se disponen conceptualmente en una representación cíclica, donde cada rotación corresponde a un posible orden de atributos. Sobre esta representación se construyen estructuras compactas basadas en *wavelet trees*, tries y otros índices sucintos. Mediante operaciones de rango, selección y acceso sobre estas estructuras, es posible simular el acceso a cualquiera de los órdenes de atributos requeridos por LTJ sin materializar explícitamente $d!$ índices. El *Ring* ofrece así una representación comprimida que permite recorrer las tuplas en órdenes de atributos diferentes con un sobrecosto controlado en tiempo y espacio. La pregunta es: ¿cuántos *Rings* se necesitan como mínimo para que se puedan recorrer las tuplas en cualquier orden?

En términos de contribución, el artículo de Arroyuelo et al. [3] demuestra que, en el caso de bases de datos de grafos modeladas mediante relaciones de aridad 3, es posible alcanzar complejidad temporal óptima en el peor caso utilizando estructuras cuyo espacio excede el tamaño de los datos solo en un factor sublogarítmico. Este resultado motiva su extensión al modelo relacional general considerado en esta memoria, donde una relación de aridad d posee d atributos y el problema consiste en determinar cuántos *Rings* u *Order Graphs* se requieren para soportar eficientemente múltiples órdenes de acceso.

Definición y contexto de los *Order Graphs*

Los *Order Graphs* aparecen en la literatura como grafos dirigidos con vértices y aristas etiquetados que representan órdenes de atributos y la transición entre dichos órdenes mediante operaciones elementales. Siguiendo el análisis adoptado en esta memoria, un *Order Graph* de dimensión d puede entenderse como un grafo dirigido $G(V, E)$, donde los vértices representan órdenes (permutaciones) de los atributos $\{1, 2, \ldots, d\}$. Cada arista está etiquetada por un atributo $j \in \{1, 2, \ldots, d\}$, y conecta un orden Π con otro Π' obtenido desplazando el atributo j a la primera posición del orden en Π; de este modo, la etiqueta de la arista corresponde al atributo que se desplaza en el siguiente paso de un algoritmo de *join* basado en orden de atributos.

Un *Order Graph* se dice completo cuando, para todo subconjunto propio de atributos $S \subset \{1, \ldots, d\}$ y para todo atributo $x \notin S$, existe un camino de largo $|S| - 1$, cuyas aristas están etiquetadas por todos los elementos de $S - \{x\}$ sin orden particular, y el camino se puede extender con una arista etiquetada por x, ya sea como última arista del camino o como primera arista del camino. Esta propiedad garantiza que, para cualquier secuencia de vinculaciones de atributos compatible con un algoritmo de tipo *Leapfrog Triejoin* (LTJ), el *Order Graph* contiene un camino que la realiza sin violar los requisitos de prefijo exigidos por dicho esquema de evaluación.

El artículo de Arroyuelo et al. [3] propone una estructura de datos que combina ideas de índices sucintos de texto (como *wavelet trees* y estructuras comprimidas de alto orden) con algoritmos WCOJ para grafos. Construyendo una representación cíclica de las aristas, los autores logran soportar el algoritmo LTJ sobre grafos de gran escala con un espacio adicional muy reducido respecto de la representación clásica de las aristas. La complejidad teórica se mantiene acorde con la cota de $\rho^*(Q)$, y los experimentos muestran que la aproximación es competitiva frente a motores especializados de grafos.

2.5. Estudio de tamaños mínimos de *Rings* y *Order Graphs* para *joins* óptimos

El problema central del trabajo se sitúa en el estudio de estructuras mínimas en tamaño de *Rings* y *Order Graphs* que permitan algoritmos WCOJ.

En [5] se introduce la noción de conjunto d-suficiente de órdenes, esto es, un conjunto de órdenes sobre $\{1, \ldots, d\}$ tal que, para cualquier consulta de *join* sobre atributos $\{1, \ldots, d\}$ y cualquier algoritmo de tipo LTJ, existe al menos un orden en el conjunto que permite alcanzar complejidad óptima en el peor caso. Los autores definen una función (que denotaremos por $ctw(d)$, siguiendo el uso habitual) que mide el tamaño mínimo de un conjunto d-suficiente, y proporcionan cotas superiores e inferiores para esta cantidad bajo distintos supuestos de capacidad de índice (por ejemplo, acceso unidireccional o bidireccional).

En esta línea de investigación se han organizado y sintetizado resultados que, además, se complementan con trabajos posteriores [4] que refinan dichas cotas y aportan construcciones

más eficientes. En particular, además de las cotas inferiores, interesa documentar las mejores cotas superiores conocidas, aun cuando no provengan de búsquedas exhaustivas, porque son las que guían el diseño de *Rings* y *Order Graphs* compactos en escenarios prácticos.

Actualización de cotas superiores reportadas

La Tabla 2.1 resume valores de referencia para el tamaño mínimo de índices de orden (o conjuntos de órdenes) necesarios para soportar algoritmos WCOJ bajo distintas capacidades de índice, en base a [4]. En particular, los valores 72 y 66 (para $d = 6$, CTW y CTWO[4]) y 83 (para $d = 7$, bidireccional CBTWO [4]) se muestran en rojo para destacar que corresponden a valores actualizados.[2]

Tabla 2.1: Tamaños mínimos de índices de orden para soportar algoritmos wcoj según capacidades del índice (adaptado de [4]). Las cotas superiores marcadas con asterisco no fueron obtenidas por búsqueda exhaustiva, por lo que podrían existir valores menores. Los valores en rojo corresponden a cotas superiores actualizadas.

	Unidirectional				Bidirectional		
d	TW	CTW	CTWO	L. Bound	CBTW	CBTWO	L. Bound
2	4	2	2	2	2	2	2
3	12	6	6	6	3	3	3
4	32	16	15	12	8	8	8
5	80	40	33	30	25	20	20
6	192	72*	66*	60	42	42	42
7	448	168*	162*	140	84*	83*	70
8	1024	400*	360*	280	200*	209*	168

Alcance de cotas inferiores: ciclos vs. *Order Graphs* con ramificaciones

Un punto relevante, y corregido en material complementario posterior, es que ciertas cotas inferiores formuladas para ciclos no se extienden automáticamente a *Order Graphs* generales con ramificación (a veces descritos como *hairy cycles*). La intuición basada en δ —según la cual ramificar no produce más *l-paths* distintos que aristas— puede mantenerse, pero falla un supuesto clave: en presencia de ramificación, una misma posición de un atributo A puede cubrir múltiples *l-paths*, de modo que ya no se requiere "tantos *l-paths* distintos por atributo" como en el caso cíclico. En consecuencia, cotas inferiores demostradas para ciclos deben explicitar su dominio de validez cuando se comparan con *Order Graphs* generales.

Este matiz no es solo técnico: se reportó un *Order Graph* para dimensión $d = 6$ con 40 aristas, mientras que la cota inferior derivada bajo el esquema anterior, válida para ciclos, era 42. Esto muestra que, al permitir ramificación, los *Order Graphs* pueden ser estrictamente

[2]Estos valores actualizados fueron informados por el profesor Dr. Gonzalo Navarro como observaciones durante el proceso de revisión de esta memoria.

más pequeños que el menor conjunto de ciclos, por lo que la comparación "*Order Graph* vs. ciclos" requiere distinguir cuidadosamente el tipo de estructura considerada (véase [14]).

Síntesis preliminares de la literatura

Como parte del trabajo se elaboraron síntesis de la literatura sobre *Rings* y *Order Graphs*, con énfasis en cotas de tamaños mínimos y en la función $ctw(d)$. Estas síntesis recogen la definición de conjuntos d-suficientes, las principales cotas conocidas para $ctw(d)$ y ejemplos de construcciones existentes, apoyándose en los resultados originales [5] y en resultados complementarios.

En una primera síntesis se ofrece una visión panorámica de la literatura reciente, destacando que, aunque existen cotas generales para $ctw(d)$, la brecha entre cotas superiores e inferiores sigue siendo significativa para muchos valores de d, y poniendo de relieve la ausencia de caracterizaciones exactas. Además, se discuten ejemplos específicos de conjuntos de órdenes construidos a partir de ideas combinatorias. Esta síntesis consolida definiciones, resultados parciales y direcciones futuras, incorporando en su versión final la actualización de cotas superiores destacadas en la Tabla 2.1.

Una segunda síntesis profundiza en la identificación de brechas, organizando trabajos relevantes por tipo de estructura, tipo de resultado sobre tamaños y clase de consultas soportadas. Se destacan preguntas abiertas acerca de la existencia de familias de *Order Graphs* de tamaño casi lineal en d que sean completos para clases amplias de consultas, y se subraya la necesidad de algoritmos constructivos para obtener conjuntos d-suficientes casi mínimos. Asimismo, se incorpora explícitamente la distinción entre cotas válidas para ciclos y su posible no aplicabilidad directa a *Order Graphs* con ramificación, como motivación para estudiar cotas inferiores más robustas en el caso general.

2.6. Algoritmos y enfoques relevantes de la literatura

Además de los resultados teóricos sobre cotas de tamaño y estructuras combinatorias, el estado del arte incluye varios trabajos que integran algoritmos de *join* óptimos en el peor caso (*Worst-Case Optimal Joins*, WCOJ) en motores de bases de datos y sistemas de grafos, y otros que proponen estrategias para seleccionar órdenes de atributos o combinar de forma híbrida *joins* binarios y WCOJ.

Integración de algoritmos de *join* óptimos en el peor caso en motores relacionales y de grafos

Freitag et al. [9] estudian la integración de algoritmos WCOJ en el sistema gestor de bases de datos relacionales Umbra, introduciendo variantes como *Hash Trie Join* (HTJ) y analizando su comportamiento en consultas complejas. Sus resultados muestran que los

WCOJ pueden competir y, en algunos casos, superar a planes tradicionales basados en *joins* binarios, especialmente en consultas densas donde las cotas de AGM son ajustadas. El enfoque combina una planificación basada en el hipergrafo de la consulta con decisiones pragmáticas sobre el uso de índices y estructuras en memoria.

Mhedhbi y Salihoglu, en una serie de trabajos centrados en sistemas gestores de bases de datos de grafos [12, 1], exploran la combinación de WCOJ con estrategias tradicionales de *joins*, proponiendo modelos híbridos que seleccionan dinámicamente el mejor método según la selectividad y el patrón de la consulta. Estos trabajos analizan no solo la complejidad teórica, sino también aspectos de implementación, como la organización de datos en memoria y la compatibilidad con distintos lenguajes de consulta (por ejemplo, SPARQL para RDF).

En términos críticos, estos enfoques son esenciales para validar la relevancia práctica de los WCOJ, demostrando que las ideas provenientes de la teoría de hipergrafos pueden materializarse en sistemas de producción. Sin embargo, en relación con la memoria, su conexión con tamaños mínimos de *Rings* y *Order Graphs* es indirecta: si bien dejan claro que el soporte eficiente de múltiples órdenes de atributos es crucial, no estudian explícitamente la estructura combinatoria de los conjuntos de órdenes ni el tamaño mínimo de las estructuras que los representan.

Optimización de órdenes de atributos

wang et al. [20] abordan el problema de seleccionar órdenes de atributos adecuados para algoritmos WCOJ, es decir, permutaciones de los atributos que determinan el orden en que estos se procesan durante la evaluación del *join*. Empleando técnicas de aprendizaje por refuerzo, ADOPT adapta dicho orden a características de los datos y de las consultas. En este enfoque, el orden de atributos se trata como una política que se actualiza en función del rendimiento observado, equilibrando la exploración de nuevos órdenes con la explotación de órdenes que han demostrado ser eficientes.

Desde el punto de vista de esta memoria de título, ADOPT representa un enfoque complementario: en lugar de reducir el número de órdenes que deben soportarse de forma estática, se centra en seleccionar, dentro de un espacio potencialmente grande de órdenes, aquellos que resultan más prometedores en la práctica. La principal fortaleza es su orientación a datos y su capacidad para adaptarse a patrones de consultas reales; la principal limitación, para los fines de esta memoria, es que no aporta resultados sobre tamaños mínimos de conjuntos de órdenes ni sobre la estructura global del espacio de órdenes.

Free Join y enfoques híbridos

Wang et al. [21] proponen una estrategia que unifica ideas de WCOJ y *joins* tradicionales, permitiendo que el plan de ejecución transite de forma flexible entre etapas que aprovechan la optimalidad en el peor caso y etapas basadas en *joins* binarios cuando las características de la consulta o de los datos así lo aconsejan. El trabajo introduce un marco de planificación en el que se consideran múltiples estructuras de datos e índices, y se enfatiza la importancia

de representar adecuadamente el espacio de planes.

Aunque *Free Join* no utiliza explícitamente *Rings* u *Order Graphs*, su relevancia para la memoria radica en que refuerza la idea de que los planes de *join* deben explorar un espacio complejo de órdenes de atributos y combinaciones de operadores. Las estructuras compactas como *Rings* y *Order Graphs* pueden verse como una posible base para representar dicho espacio de forma más controlada, especialmente si se dispone de resultados sobre tamaños mínimos.

CompactLTJ y variantes

Arroyuelo et al. [2] exploran variantes compactas del algoritmo *Leapfrog Triejoin*, enfocadas en reducir el costo espacial de los índices necesarios para soportar múltiples órdenes de atributos. Se proponen estructuras de datos que reutilizan información entre diferentes órdenes y que se apoyan en técnicas de compresión para limitar el crecimiento del espacio conforme aumenta la aridad. En esencia, estas variantes buscan un compromiso similar al de los *Rings*: conservar la optimalidad en el peor caso sin incurrir en el costo factorial de materializar todos los órdenes posibles.

Críticamente, [2] proporciona evidencia de que es posible reducir el espacio adicional requerido por WCOJ más allá de los índices clásicos, aunque sin llegar necesariamente al nivel de compresión alcanzado por los *Rings* para casos de grafos. Para la memoria, estos trabajos complementan la motivación respecto de por qué es importante estudiar tamaños mínimos de estructuras que representan órdenes de atributos: el espacio puede ser un factor limitante tan determinante como la complejidad temporal.

Berdai y Chiadmi [8] ofrecen una revisión sistemática de los algoritmos WCOJ y de los intentos por incorporarlos en sistemas de datos relacionales. El artículo comienza presentando el trasfondo teórico basado en la cota AGM, para luego revisar los principales algoritmos WCOJ (como NPRR y derivados) y describir las técnicas de indexación sobre tries ordenados que estos requieren. Posteriormente, discute aspectos "más allá de la indexación", tales como la convivencia entre WCOJ y *joins* binarios clásicos, el uso de técnicas de memoización, la escalabilidad en entornos distribuidos y otras propuestas recientes relacionadas con consultas sobre grafos y cargas analíticas.

Como síntesis crítica, este trabajo confirma la relevancia de los WCOJ como alternativa teóricamente sólida frente a los *joins* binarios tradicionales, pero subraya que su adopción en sistemas gestores de bases de datos reales se enfrenta a retos importantes. En particular, destaca el problema del elevado número de índices secundarios necesarios para soportar múltiples órdenes de atributos y las dificultades asociadas a su mantenimiento cuando los datos son mutables. El artículo organiza y discute un amplio conjunto de propuestas algorítmicas y de sistemas, pero no desarrolla de forma específica la teoría de conjuntos d-suficientes ni introduce estructuras como *Rings* u *Order Graphs*; más bien, aborda la cuestión a un nivel de detalle centrado en algoritmos, técnicas de indexación y consideraciones de integración en sistemas, lo que lo convierte en un punto de referencia general sobre el estado del arte en WCOJ, complementario a las contribuciones estructurales que se analizan en esta memoria.

2.7. Síntesis del estado del arte y brechas de investigación

La literatura revisada muestra una evolución clara desde resultados teóricos sobre el tamaño de resultados de *joins* y la complejidad de su evaluación, hacia propuestas de estructuras y algoritmos concretos que buscan hacer viable la adopción de WCOJ en sistemas prácticos. Atserias, Grohe y Marx [7] establecen la conexión entre el hipergrafo de la consulta, el número de cubrimiento fraccional y la complejidad de algoritmos de evaluación, mientras que Ngo et al. [15] demuestran que es posible alcanzar constructivamente estas cotas mediante algoritmos específicos.

Sobre esta base, Arroyuelo et al. [3, 2] y trabajos posteriores sobre variantes compactas de LTJ [2] muestran que uno de los obstáculos para la adopción de algoritmos WCOJ no es solo su complejidad temporal, sino también el costo espacial asociado a la necesidad de múltiples órdenes de atributos. La introducción de *Rings* y de conjuntos d-suficientes formaliza este problema, planteando explícitamente la cuestión de cuántos órdenes son realmente necesarios para garantizar la optimalidad en el peor caso. Los resultados existentes proporcionan cotas y ejemplos, pero no una caracterización completa de los tamaños mínimos. En particular, las mejores cotas superiores conocidas para distintos valores de d dependen de la capacidad de índice (unidireccional o bidireccional) y, en varios casos, no provienen de búsquedas exhaustivas; por ello, conviene reportarlas como referencia del estado del arte, indicando su carácter de cota superior no necesariamente óptima (Tabla 2.1).

Al mismo tiempo, la integración de wcoj en motores reales y enfoques adaptativos como ADOPT y Free Join [9, 12, 1, 21, 20] demuestran que la planificación de joins óptimos en el peor caso debe enfrentarse a un espacio de decisiones complejo, donde confluyen: (i) parámetros combinatorios del hipergrafo de la consulta, (ii) características estadísticas de los datos, (iii) costos físicos de acceso a índices, y (iv) restricciones de espacio. En este contexto, disponer de estructuras compactas (por ejemplo, order graphs o rings cercanos al mínimo) seria una ventaja considerable, pues reduciría el espacio de búsqueda de planes sin comprometer la optimalidad.

Un aspecto refinado en material complementario posterior es la validez del dominio de ciertas cotas inferiores: resultados demostrados para estructuras cíclicas no necesariamente se extienden a *Order Graphs* generales con ramificación. Este punto es relevante porque existen contraejemplos reportados donde un *Order Graph* de dimensión $d = 6$ alcanza 40 aristas, por debajo de una cota inferior 42 derivada bajo supuestos válidos para ciclos. En consecuencia, el estado del arte reciente sugiere que (i) los *Order Graphs* pueden ser estrictamente más pequeños que el menor conjunto de ciclos, y (ii) es necesario distinguir entre "mínimos para ciclos" y "mínimos para *Order Graphs* generales" al formular y evaluar cotas.

De la revisión y análisis de la literatura presentada en este capítulo se identifican brechas de investigación que pueden agruparse en cuatro bloques:

- **Caracterización de tamaños mínimos.** Aunque existen cotas para $ctw(d)$ y para tamaños de *Order Graphs* completos, la brecha entre cotas superiores e inferiores sigue siendo amplia para muchos valores de d. No se dispone de fórmulas cerradas ni de

caracterizaciones completas del crecimiento asintótico exacto, y parte de la evidencia empírica relevante, en particular las cotas superiores, no es exhaustiva.

- **Construcciones explícitas y verificables.** Se requieren familias de conjuntos d-suficientes y *Order Graphs* completos que sean, simultáneamente, (i) casi mínimos, (ii) construibles mediante algoritmos explícitos, y (iii) verificables, o al menos con certificados, para dimensiones moderadas, permitiendo cerrar la brecha entre cotas y valores alcanzados en práctica.
- **Cotas inferiores robustas para *Order Graphs* generales.** Dado que algunas cotas inferiores demostradas para ciclos no se extienden a *Order Graphs* con ramificación, falta un marco de cotas inferiores que capture el caso general. Esto incluye comprender qué propiedades estructurales, como la ramificación o la cobertura de *l-paths*, permiten superar barreras válidas en el caso cíclico.
- **Conexión con optimizadores reales.** Si bien se ha avanzado en la integración de WCOJ en motores de bases de datos y grafos, no existe aún un marco bien desarrollado que incorpore explícitamente estructuras mínimas de *Rings* u *Order Graphs* en el proceso de planificación, ni estudios sistemáticos del impacto de estas estructuras en el rendimiento global, tanto en tiempo como en espacio, frente a otras alternativas.

Estas brechas justifican el *Estudio de tamaños mínimos de rings y order graphs para joins óptimos*. En particular, la memoria de título puede contribuir si logra: (i) refinar cotas conocidas para $ctw(d)$ y/o para el tamaño de *Order Graphs* completos, ya sea teóricamente o mediante evidencia experimental robusta; (ii) proponer algoritmos constructivos para generar colecciones de órdenes de tamaño casi mínimo que sean utilizables en práctica; (iii) estudiar explícitamente la diferencia entre mínimos para ciclos y para *Order Graphs* generales, incluyendo criterios de validez de cotas inferiores; y (iv) explorar, en escenarios controlados, cómo estas estructuras compactas se integran en procesos de planificación de *joins*, evaluando su impacto en la complejidad temporal y espacial.

Capítulo 3

Metodología y definición formal del problema

El trabajo se plantea como un estudio *teórico–computacional* en optimización combinatorial aplicada a estructuras de datos para algoritmos wco. La metodología combina tres componentes principales:

- **Diseño teórico de algoritmos de búsqueda** (exhaustiva y no exhaustiva) sobre el espacio de Rings y Order Graphs completos.
- **Implementación eficiente** de dichos algoritmos en arquitecturas secuenciales (CPU).
- **Evaluación experimental** de los tamaños obtenidos y del desempeño de los algoritmos, comparando con el estado del arte.

3.1. Formulación del problema

El interés de este trabajo se encuentra en la estructura combinatorial de los *Rings* y *Order Graphs*. Por lo tanto, minimalizamos la representación que usamos para ellos para solo trabajar con la estructura combinatorial y dejar en segundo plano la funcionalidad de índice.

Definición 3.1 Un Ring de dimensión d corresponde a un ciclo C_d cuyos vértices están etiquetados con valores en $\{1, 2, \ldots, d\}$ tal que no hay dos nodos que comparten etiqueta.

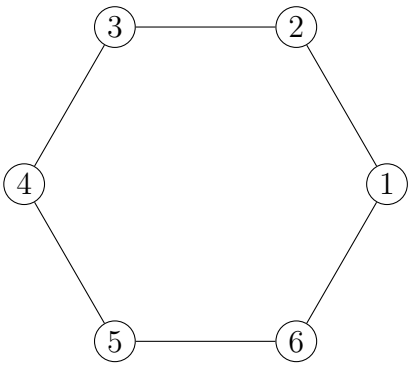

Figura 3.1: Ejemplo de Ring de dimensión $d = 6$.

Un Ring puede representarse mediante una lista de sus vértices en orden que indiquen el orden del ciclo. Como es bidireccional, no importa la dirección en que se especifiquen los vértices. Algunas representaciones de la figura 3,1 son:

$$3, 2, 1, 6, 5, 4$$
$$4, 3, 2, 1, 6, 5$$
$$5, 6, 1, 2, 3, 4$$

Definición 3.2 Un conjunto de Rings R de dimensión d, es CBW-completo si podemos generar cualquier permutación de $[d]$ escogiendo un Ring r de R, un vértice v de r y recorrer r por completo empezando en r hacia cualquier dirección, con la posibilidad de cambiar de dirección en medio de un camino.

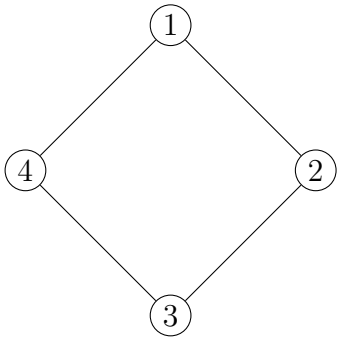

Figura 3.2: Ring $r = (1, 2, 3, 4)$ de dimensión $d = 4$. Al recorrer el ciclo completo desde cualquier vértice y en ambas direcciones, este ring genera las permutaciones $\{(1, 2, 3, 4), (2, 3, 4, 1), (3, 4, 1, 2), (4, 1, 2, 3), (1, 4, 3, 2), (4, 3, 2, 1), (3, 2, 1, 4), (2, 1, 4, 3),$ $(1, 2, 4, 3), \dots\}$, pero no genera, por ejemplo, la permutación $(1, 3, 2, 4)$. Por lo tanto, el ring r por sí solo no es un conjunto CBW-completo.

Definición 3.3 Un conjunto de Rings R es $CBTW$-completo si para todo $S \subseteq [d]$ distinto de vacío, y todo $x \in S$, existe un Ring en R con una sección contigua de exactamente $|S|$ vértices que contiene todos los elementos de S y x se encuentra en un extremo.

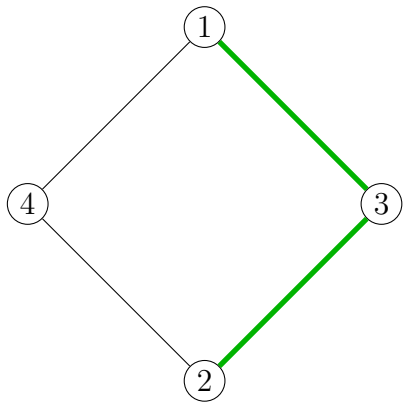

Figura 3.3: Ring de dimensión $d = 4$ ilustrando la definición de CBTW-completitud. Para el subconjunto $S = \{1, 2, 3\}$ y el elemento $x = 1$ (segmento verde), el ring contiene una sección contigua de exactamente $|S|$ vértices que incluye todos los elementos de S, con x en uno de sus extremos. Noten que el orden no importa. En contraste, para el subconjunto $S = \{1, 4, 2\}$ y el elemento $x = 4$, no existe ninguna sección contigua que contenga exactamente los elementos de S, por lo que el ring por sí solo no es CBTW-completo.

Definición 3.4 Un Order Graph de dimensión d es un grafo dirigido etiquetado $G(V, E)$ donde V es un subconjunto de los $d!$ posibles órdenes de $[d]$ y una arista etiquetada $j \in [d]$ de un nodo Π a Π' indica que se puede obtener el orden Π' a partir de Π desplazando el elemento j al inicio.

Definición 3.5 Un Order Graph G de dimensión d es completo si, para cada subconjunto $S \subseteq [d]$ distinto de vacío, y todo $x \in S$, existe un camino dirigido de largo $|S|$ en G cuyas etiquetas de aristas contienen exactamente los elementos de S tal que la etiqueta x se encuentra al inicio o final del camino.

En [3] se demuestra que los Order Graph completos minimales son aquellos cuyos vértices tienen indegree menor o igual a 1.

Definición 3.6 Un Cyclic Order Graph de dimensión d es un Order Graph de dimensión d con la misma forma de un ciclo de largo finito m, C_m.

Dada la complejidad combinatoria de los *Order Graphs* generales, en este trabajo se consideran únicamente *Cyclic Order Graphs*. Estas estructuras pueden representarse mediante la lista ordenada de etiquetas de sus aristas, ya que las etiquetas de los nodos pueden recuperarse a partir de dicha información.

Además, existía una conjetura [4] según la cual los *Cyclic Order Graphs* completos minimales de dimensión d tendrían la misma cantidad de vértices que los *Order Graphs* completos minimales de dimensión d. Esta conjetura fue refutada por Ruben Becker, Adrián Gómez-Brandón y Nicola Prezza, quienes encontraron un *Order Graph* de 40 arcos para dimensión 6, mientras que la cota inferior conocida para *Cyclic Order Graphs* es 42.

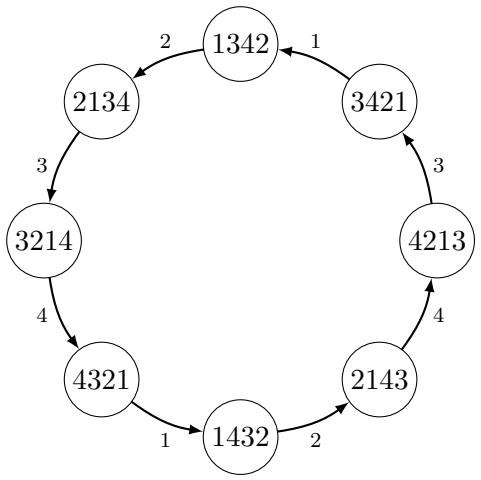

Figura 3.4: Ejemplo de *order graph* de dimensión $d = 4$ completo. Cada nodo representa un orden de [4] y cada arista etiquetada con j indica la operación de mover el elemento j al inicio.

En términos formales, el problema se puede describir así: dado un entero $d \geq 2$,

- Encontrar conjuntos de Rings CBW-completos minimales o pequeños en espacio de dimensión d,
- encontrar conjuntos de Rings $CBTW$-completos minimales o pequeños en espacio de dimensión d, y
- encontrar Cyclic Order Graphs completos minimales de dimensión d.

Ambos problemas son de naturaleza combinatoria: el espacio de órdenes crece factorialmente con d, y el espacio de posibles grafos (y ciclos) crece de forma aún más explosiva. Además, este espacio no puede reducirse ni indexarse fácilmente, ya que las propiedades de interés —como la completitud, la cobertura de secuencias de atributos y el cumplimiento de restricciones de prefijo— dependen de configuraciones globales de la estructura y no de decisiones locales independientes. En consecuencia, no resulta sencillo descartar grandes regiones del espacio de búsqueda mediante criterios simples, por lo que se hace necesario recurrir a estrategias de búsqueda cuidadosamente diseñadas.

3.2. Enfoque metodológico

La metodología considera los siguientes enfoques:

- **Búsqueda exhaustiva para dimensiones pequeñas:** se explora sistemáticamente el espacio de estructuras candidatas (Rings y Cyclic Order Graphs) para valores pequeños de d, sin imponer restricciones fuertes de complejidad, con el objetivo de obtener todas las soluciones óptimas conocidas y desarrollar intuición sobre la estructura de las soluciones.

- **Búsqueda no exhaustiva y heurísticas para dimensiones mayores:** se emplean métodos que no exploran todo el espacio, pero que tienden a encontrar soluciones buenas o cada vez mejores en función del tiempo. En particular, se consideran técnicas inspiradas en *Búsqueda Local* y *Simulated Annealing*, adaptadas al espacio discreto.

Adicionalmente, se establece un criterio práctico de "tiempo razonable": se considera razonable que una ejecución individual de un algoritmo no supere aproximadamente una semana de tiempo real. Si la ejecución proyectada supera ese límite, se corta el cómputo y se considera que la estrategia exhaustiva es inadecuada para esa dimensión.

3.3. Métricas y criterios de evaluación

Para evaluar las soluciones y los algoritmos se consideran, entre otros, los siguientes criterios:

- **Tamaño de la estructura:** número de nodos del Ring o Cyclic Order Graph completo es la medida más importante.
- **Tiempo de ejecución:** tiempo requerido para encontrar soluciones óptimas (o de buena calidad) en cada dimensión d.
- **Calidad relativa de la solución:** comparación con las mejores cotas inferiores y superiores conocidas en la literatura.
- **Escalabilidad:** capacidad del algoritmo para manejar dimensiones crecientes antes de volverse impracticable.

Capítulo 4

Diseño e implementación de algoritmos exhaustivos

Se decidió utilizar `C++` para la implementación de los algoritmos debido a su alta velocidad de ejecución, su madurez como lenguaje de sistemas y la facilidad de integrar optimizaciones de bajo nivel cuando el costo computacional del problema lo exige. Adicionalmente, el uso de `C++` resulta conveniente por familiaridad con `CUDA` y por permitir una implementación directa y eficiente de estructuras de datos, en comparación con `C`. Dado que el sistema en el que se realizarán los experimentos es de 64 bits, se asume un largo de palabra $w = 64$, lo que habilita representar conjuntos y coberturas mediante máscaras de bits, reduciendo operaciones de pertenencia y unión/intersección a operaciones bit a bit de costo constante.

En este capítulo se presentan y analizan los algoritmos utilizados para construir y evaluar estructuras completas bajo distintos modelos de representación. El objetivo transversal es resolver el problema de cobertura asociado a un conjunto de restricciones (subconjuntos y "ventanas" de tamaño acotado) y, a la vez, estudiar el costo computacional de abordarlo tanto de manera exhaustiva como mediante métodos no exhaustivos. Para ello, el capítulo se organiza en torno a tres familias de estructuras: (i) conjuntos de *Rings* completos bajo el criterio **CBW**, (ii) conjuntos de *Rings* completos bajo el criterio **CBTW**, y (iii) *Cyclic Order Graphs* (**COG**) completos. Aunque difieren en su definición formal y en la forma en que inducen cobertura, las tres familias comparten un mismo núcleo algorítmico: representar restricciones y coberturas como conjuntos binarios (*bitmasks*) para acelerar la verificación y habilitar podas efectivas en la búsqueda.

La conexión entre secciones es la siguiente. En primer lugar, las secciones dedicadas a *Rings* (CBW y CBTW) abordan el problema como una selección de estructuras discretas precomputables (los *Rings*) que deben cubrir un universo de restricciones; esto permite formular la tarea como una instancia de *Set Cover*, donde cada *Ring* induce un subconjunto cubierto del universo. En particular, CBW y CBTW se diferencian en la relación de satisfacción empleada para definir qué significa "cubrir" una restricción, por lo que sus algoritmos comparten la misma arquitectura (generación, verificación y búsqueda con poda), cambiando únicamente el predicado de cobertura. En contraste, los COG se tratan como una única estructura global, donde la completitud se verifica recorriendo caminos ("ventanas") dentro

del grafo y comprobando cobertura sin poder separar el problema en "piezas" independientes con la misma facilidad. Finalmente, la última sección del capítulo introduce enfoques no exhaustivos (aproximación, búsquedas locales y heurísticas iterativas), que reutilizan la misma noción de cobertura pero sacrifican optimalidad garantizada para escalar a tamaños donde la enumeración exhaustiva deja de ser viable.

4.1. Conjuntos de *Rings* CBTW-completos

Al trabajar con *Rings* se considera implícitamente la noción de *permutaciones cíclicas bidireccionales* de los elementos de $\{1, 2, \ldots, d\}$. Dos permutaciones se consideran equivalentes si una puede obtenerse de la otra mediante una rotación o mediante la inversión del orden de recorrido. Un resultado conocido de combinatoria establece que, para $d > 2$, el número de permutaciones cíclicas bidireccionales distintas de $\{1, 2, \ldots, d\}$ es

$$\frac{(d-1)!}{2}.$$

Cada una de estas clases de equivalencia puede representarse de manera canónica fijando la rotación que deja al menor valor en la primera posición y, entre las dos orientaciones posibles (orden directo o reverso), escogiendo aquella en que el segundo valor sea el menor posible.

Para generar todas estas representaciones canónicas, basta con permutar los elementos de $\{2, 3, \ldots, d\}$, anteponer el valor 1, y luego normalizar cada permutación según el criterio anterior. La complejidad temporal y espacial de este procedimiento es $O((d-1)!)$.

Algoritmo 1 Enumeración de los *Rings* canónicos de dimensión d (§ 4.1).

Require: Dimensión $d \geq 2$.
Ensure: Lista L de los $(d-1)!/2$ *Rings* canónicos distintos (cada clase de equivalencia bajo rotación y reverso queda representada exactamente una vez).

```
L ← ⟨ ⟩
T ← [ 2, 3, ..., d ]                                   ▷ Cola a permutar
for cada permutación T de [2, 3, ..., d] do
    r ← [ 1 ] · T                                      ▷ Antepone 1
    if r[1] > r[d − 1] then          ▷ Orientación canónica: vecino menor primero
        r ← [ 1, T[d − 1], T[d − 2], ..., T[1] ]
    end if
    if r ∉ L then añadir r al final de L
end for
return L
```

Definición 4.1. Conjunto de *Rings* CBTW-completo de dimensión d. Es un conjunto de *Rings* R tal que, para todo subconjunto no vacío $S \subseteq \{1, 2, \ldots, d\}$ y todo elemento $x \in S$, existe un *Ring* $r \in R$ que contiene una sección contigua de largo $|S|$ formada exactamente por los elementos de S, de modo que x ocupa uno de los extremos de dicha sección.

En ese caso, se dice que el *Ring* r satisface la restricción (S, x). Se define el *conjunto de validación* como

$$Uts := \{(S, x) \mid S \subseteq [d],\ S \neq \emptyset,\ x \in S\},$$

y el *conjunto de validación* de un *Ring* r como

$$U_r := \{(S, x) \mid S \subseteq [d],\ S \neq \emptyset,\ x \in S,\ r \text{ satisface la restricción } (S, x)\}.$$

Representación y verificación mediante *bitmasks*

Para acelerar la verificación de completitud y habilitar podas en la búsqueda, se representa el universo de restricciones Uts como un arreglo indexado

$$Uts = \{u_0, u_1, \ldots, u_{|Uts|-1}\}, \qquad u_k = (S_k, x_k),$$

y se precomputa una función de indexación

$$idx : Uts \to \{0, \ldots, |Uts| - 1\}$$

que permite ubicar rápidamente cada restricción en una posición de bit.

Predicado de satisfacción

Sea $r = (r_0, r_1, \ldots, r_{d-1})$ un *Ring* de dimensión d (permutación cíclica bidireccional) y sea $(S, x) \in Uts$, con $|S| = \ell$. Decimos que r satisface la restricción (S, x) si existe un índice $t \in \{0, \ldots, d-1\}$ tal que la subsecuencia contigua

$$(r_t, r_{t+1}, \ldots, r_{t+\ell-1}) \quad \text{(índices módulo } d\text{)}$$

contiene exactamente los elementos de S, y además x coincide con uno de los extremos de dicha subsecuencia, es decir,

$$x \in \{r_t, r_{t+\ell-1}\}.$$

Notar que, por bidireccionalidad del *Ring*, no es necesario distinguir entre recorrer en sentido directo o inverso: considerar ambos extremos captura ambas orientaciones.

Máscara de cobertura $\mathrm{Mask}(r)$

Se define $\mathrm{Mask}(r)$ como un *bitmask* de largo $|Uts|$ tal que

$$\mathrm{Mask}(r)[k] = 1 \iff r \text{ satisface } u_k = (S_k, x_k).$$

Para construir $\mathrm{Mask}(r)$ se enumeran todas las subsecuencias contiguas del *Ring*: para cada inicio t y largo $\ell \in \{1, \ldots, d\}$ se obtiene el conjunto $S = \{r_t, \ldots, r_{t+\ell-1}\}$ y se activan en la máscara las restricciones (S, r_t) y $(S, r_{t+\ell-1})$. Implementacionalmente, el conjunto S puede representarse con un *bitmask* de d bits (uno por etiqueta de $[d]$), y la activación se realiza mediante $k = idx(S, x)$.

Costo de verificación

Dado que existen d elecciones posibles para el inicio de una subsecuencia y hasta d longitudes posibles, construir Mask(r) para un *Ring* r requiere $O(d^2)$ operaciones sobre representaciones de conjuntos de tamaño d. En la práctica, cuando d es pequeño, este costo se reduce a operaciones bit a bit. Posteriormente, al trabajar con *bitmasks* de largo $|Uts|$, las operaciones `OR`/`AND` y el cómputo de `popcount` se ejecutan en $O(|Uts|/w)$ palabras de tamaño w.

Algoritmo 2 Cómputo de la máscara de cobertura Mask(r) de un *Ring* (§ 4.1).

Require: *Ring* $r = (r_0, r_1, \ldots, r_{d-1})$; función de indexación $idx(\cdot, \cdot)$; tamaño $|Uts|$.
Ensure: Mask(r), *bitmask* de largo $|Uts|$ con un 1 en cada (S, x) que r satisface (Definición 4.1).

$M \leftarrow \mathbf{0}^{|Uts|}$
for $t \leftarrow 0$ **to** $d - 1$ **do** ▷ Posición inicial
  $S \leftarrow \mathbf{0}^d$ ▷ Conjunto vacío
  for $\ell \leftarrow 1$ **to** d **do** ▷ Largo de la sección contigua
    $j \leftarrow (t + \ell - 1) \bmod d$
    $S \leftarrow S \,\mathrm{OR}\, \mathrm{BIT}(r_j)$ ▷ Insertar etiqueta
    $M[\, idx(S, r_t)\,] \leftarrow 1$
    $M[\, idx(S, r_j)\,] \leftarrow 1$ ▷ Activar ambos extremos
  end for
end for
return M

El problema que se desea resolver consiste en encontrar conjuntos minimales de *Rings* R tales que la unión de los conjuntos U_r, para $r \in R$, coincida con el universo U. Esta formulación corresponde a una instancia de *Set Cover*, con la diferencia de que aquí no solo interesa obtener una solución minimal, sino también estudiar exhaustivamente las soluciones minimales.

El número total de *Rings* candidatos que deben considerarse coincide con el número de permutaciones cíclicas bidireccionales distintas sobre $\{1, 2, \ldots, d\}$, es decir,

$$\frac{(d-1)!}{2}.$$

Para $d = 5$, esto corresponde a 12 *Rings* posibles.

Por otra parte, el tamaño del universo de restricciones es

$$|U| = \sum_{i=1}^{d} \binom{d}{i} \cdot i = d \cdot 2^{d-1}.$$

Además, para cada *Ring* r, el tamaño de su conjunto de validación satisface

$$|U_r| = d \cdot (1 + (d-2) \cdot 2) + d = 2d(d-1).$$

Esta expresión se obtiene mediante el siguiente argumento combinatorio. Primero, se puede escoger el vértice inicial de una sección contigua de d formas. Luego se distinguen los siguientes casos:

- **Sección de largo** 1**.** En este caso, S contiene únicamente a x, y dicho elemento coincide con el vértice escogido. Esto aporta una única posibilidad para (S, x).
- **Sección de largo** $2 \leq |S| \leq d-1$**.** Para cada uno de los $d-2$ largos posibles ($\ell = 2, \ldots, d-1$), la sección queda determinada por la elección de su otro extremo, y el elemento x puede ocupar cualquiera de los dos extremos, lo que aporta $(d-2) \cdot 2$ posibilidades.

El caso $|S| = d$ debe tratarse por separado, pues en ese caso no se eligen extremos intermedios: S coincide con el conjunto total de vértices, y x puede ser cualquiera de los d elementos. De este modo se obtiene la expresión anterior para $|U_r|$.

Para $d = 5$, se tiene $|U| = 80$ y cada *Ring* puede cubrir, a lo más, 40 validaciones. Aunque la intersección entre los conjuntos U_r de distintos *Rings* no tiene por qué ser vacía, esta estimación sugiere que una solución minimal podría requerir pocos *Rings*, lo que vuelve razonable el uso de un algoritmo de *backtracking* con poda. Para mantener el conjunto de validaciones cubiertas se utiliza el mismo *bitmask* de largo $|Uts|$ descrito antes, por lo que las operaciones relevantes toman tiempo

$$O\left(\frac{d \cdot 2^{d-1}}{w}\right),$$

que para $d = 5$ es aproximadamente 2 operaciones sobre palabras.

Sabiendo que 5 *Rings* son suficientes para $d = 5$, la recursión puede detenerse si se intenta utilizar un sexto *Ring*. Además, como toda solución contiene al menos un *Ring*, puede suponerse sin pérdida de generalidad que el *Ring* asociado a la permutación identidad forma parte de la solución. En efecto, cualquier solución que no contenga dicho *Ring* puede transformarse en otra equivalente aplicando la permutación inversa a uno de sus *Rings* para llevarlo a la identidad, y aplicando luego esa misma permutación al resto de los *Rings* de la solución.

Por lo tanto, para $d = 5$, el *backtracking* genera $\binom{11}{4}$ conjuntos de *Rings*, sobre los cuales la validación debe aplicarse a lo más 4 veces. Un estimado del número de operaciones simples es entonces

$$\binom{11}{4} \cdot 4 \cdot 2 = 2640,$$

lo que resulta extremadamente rápido.

Para dimensión variable d, la complejidad temporal queda dada por

$$O\left(\binom{\frac{(d-1)!}{2} - 1}{s(d) - 1} \cdot \frac{d \cdot 2^{d-1}}{w} \cdot s(d)\right),$$

donde $s(d)$ denota el número de *Rings* en una solución minimal. Esta expresión supone que dicho valor es conocido. Si no lo es, el algoritmo puede extenderse partiendo de la hipótesis $s(d) = 2$ e incrementándola una vez que la búsqueda termina sin éxito, obteniendo

$$O\left(\sum_{i=1}^{s(d)} \binom{\frac{(d-1)!}{2} - 1}{i-1} \cdot \frac{d \cdot 2^{d-1}}{w} \cdot i\right).$$

Para $d = 6$, se conoce que $s(6) = 7$, lo que da lugar a aproximadamente $9 \cdot 10^8$ operaciones simples, una cantidad todavía abordable en la práctica.

Para $d = 7$, en cambio, el valor exacto de $s(7)$ no es conocido, aunque se sabe que $10 \leq s(7) \leq 12$. Si se adopta una estimación optimista y se asume $s(7) = 10$, el costo en operaciones simples asciende a aproximadamente $2 \cdot 10^{19}$, lo que vuelve inviable una búsqueda exhaustiva directa. Considerando que una máquina potente puede ejecutar del orden de 10^{10} operaciones por segundo, serían necesarios alrededor de 10^4 procesos de ese nivel para reducir el tiempo de ejecución a una escala comparable con algunos días.

Ruptura de simetrías: fijar el *Ring* identidad

El espacio de búsqueda contiene simetrías por renombramiento de etiquetas. En efecto, sea σ una permutación de $[d]$ y sea $\sigma(r)$ el *Ring* obtenido aplicando σ a cada etiqueta de r. La propiedad de CBTW-completitud es invariante bajo renombramientos: un conjunto R es CBTW-completo si y solo si $\sigma(R)$ lo es, pues Uts contiene todas las restricciones (S, x) y σ induce una biyección sobre ellas. Por lo tanto, al buscar soluciones de tamaño mínimo, se puede asumir sin pérdida de generalidad que una solución contiene un *Ring* canónico, por ejemplo el *Ring* identidad

$$r_0 = (1, 2, \ldots, d)$$

(en la representación normalizada que fija 1 en la primera posición). Esta decisión elimina soluciones equivalentes por simetría y reduce significativamente el número de combinaciones que se exploran.

Cota inferior usada en la poda (*LowerBound*)

Sea rem el conjunto (*bitmask*) de restricciones aún no cubiertas y sea i el índice a partir del cual se permite seleccionar nuevos *Rings* (en el orden lexicográfico). Definimos

$$B(rem, i) := \max_{j \geq i} |rem \cap U_{r_j}| = \max_{j \geq i} \text{popcount}\big(\text{AND}(rem, \text{Mask}(r_j))\big).$$

Si se pueden agregar a lo más m *Rings* adicionales, entonces cualquier extensión de la solución actual cubre como máximo $m \cdot B(rem, i)$ restricciones nuevas. En consecuencia, el número mínimo de *Rings* requeridos para cubrir rem está acotado inferiormente por

$$\text{LowerBound}(rem, i) := \left\lceil \frac{|rem|}{B(rem, i)} \right\rceil,$$

con la convención $\text{LowerBound}(rem, i) = +\infty$ si $B(rem, i) = 0$. La poda del Algoritmo 4 es admisible porque, si $\text{LowerBound}(rem, i) > m$, entonces incluso en el mejor caso (no alcanzable necesariamente) no existe forma de cubrir lo restante usando solo m *Rings*.

Algoritmo 3 LowerBound usado por la poda del Algoritmo 4.

Require: Bitmask de restricciones pendientes rem; índice i del primer *Ring* permitido; máscaras $\text{Mask}(r_i), \ldots, \text{Mask}(r_{K-1})$.

Ensure: $\lceil |rem|/B \rceil$, con $B = \max_{j \geq i} \text{popcount}\big(rem \,\text{AND}\, \text{Mask}(r_j)\big)$, o $+\infty$ cuando $B = 0$.

```
B ← 0
for j ← i to K − 1 do
    c ← popcount(rem AND Mask(r_j))
    if c > B then B ← c
end for
if B = 0 then return +∞
return ⌈|rem|/B⌉
```

Nota de complejidad (verificación y operaciones bit a bit)

Una vez precomputadas las máscaras $\text{Mask}(r)$ para todos los *Rings* candidatos, cada operación de actualización

$$covered \leftarrow \text{OR}(covered, \text{Mask}(r))$$

y cada cómputo de `popcount` se ejecuta en $O(|Uts|/w)$ palabras de tamaño w. Por lo tanto, el costo dominante se traslada desde la verificación de restricciones individuales a operaciones sobre palabras, lo que explica el buen desempeño en dimensiones pequeñas y la efectividad de podas basadas en cobertura.

4.2. Conjuntos de *Cyclic Order Graphs* completos

Los *Cyclic Order Graphs* pueden representarse mediante las etiquetas de sus aristas, en orden. A diferencia de los *Rings*, esta es una única estructura de gran tamaño, impidiendo o complejizando la división en estructuras independientes. El paradigma divide y vencerás no se aplica con tanta facilidad como en el caso de los *Rings*.

Se empieza con un algoritmo de verificación de completitud. Dada una dimensión d y una secuencia de etiquetas de largo m, se debe verificar si es un *Cyclic Order Graph* completo. Para ello se debe verificar que para todo $S \subseteq [d]$ no vacío, y todo $x \in S$, existe un camino dirigido de largo $|S|$ en G cuyas etiquetas de aristas contienen exactamente los elementos de S tal que la etiqueta de x se encuentra al inicio o al final del camino.

Nuevamente se utiliza el conjunto

$$Uts := \{(S, x) \mid S \subseteq [d],\ S \neq \emptyset,\ x \in S\}$$

como universo de restricciones (idéntico al de § 4.1).

Para verificar que todas las restricciones se satisfacen, se puede, por fuerza bruta, recorrer todos los caminos de largo menor o igual a d. Luego, para cada uno de esos caminos, se construye S con las etiquetas de las aristas y, luego, x puede ser cualquiera de los extremos del camino.

La secuencia de etiquetas de aristas es de largo m y cada posición puede ser el inicio de un nuevo camino de largo entre 1 y d. Hay $m \cdot d$ caminos posibles y para cada uno hay dos elecciones posibles de x, por lo que la verificación se ejecuta en $O(m \cdot d)$. Esto se puede mejorar: el número de restricciones distintas con $|S| = l$ es $\binom{d}{l} \cdot l$. Con esto, la verificación puede hacerse primero para los largos donde $\binom{d}{l} \cdot l$ se maximiza; si no se verifican todas, se puede detener inmediatamente, de otra forma se sigue con el siguiente largo que maximiza el número de restricciones.

Dado esto, se definen los conjuntos

$$U_l := \{(S, x) \mid S \subseteq [d],\ S \neq \emptyset,\ x \in S,\ |S| = l\}$$

y en la verificación de restricciones para largos de camino igual a l, se empieza con el conjunto vacío y se insertan los (S, x) que se vayan encontrando. Una vez se acaban los caminos de largo l, se verifica si el conjunto generado es igual a U_l. Por supuesto, se utilizan *bitsets* para la implementación.

Hay un problema: la complejidad ya no es $O(m \cdot d)$ pues se asumía que los largos de caminos se recorrían en orden; al computar S se podía utilizar el S anterior y agregarle el nuevo elemento. Como los largos ya no se recorren en orden, eso ya no es posible. Es necesario poder computar el conjunto generado por las etiquetas en un rango $[l, r]$ sin necesariamente tener el rango anterior. Afortunadamente, la complejidad $O(m \cdot d)$ se puede mantener. Primero, se computa el conjunto generado por el camino de largo l para la primera posición; esto toma $O(l)$ y computa el conjunto del rango $[1, l]$. Para computar el conjunto del rango $[2, l + 1]$ se puede incluir la etiqueta $(l + 1)$-ésima y eliminar la primera con complejidad $O(1)$; esto se puede repetir hasta la última posición. Por lo que la complejidad queda

$$O\left(\sum_{\ell=1}^{d} \ell + m \cdot d\right) = O(d^2 + m \cdot d),$$

que, como $d \leq m$ (de otro modo no podría existir un camino de largo d sin repeticiones), se simplifica a $O(m \cdot d)$.

Un primer algoritmo de búsqueda exhaustiva puede ser generar todas las listas posibles de etiquetas de largo m, con m empezando en d y aumentando de a uno mientras no se encuentren soluciones.

Se sabe que el *Cyclic Order Graph* minimalista para $d = 5$ tiene tamaño 20 en nodos o aristas. Esto quiere decir que son $m = 20$ etiquetas, donde cualquiera de ellas puede ser cualquier valor de $[d]$. El número de secuencias posibles es $5^{20} \sim 10^{14}$. No es para nada viable, pues no funciona para $d = 5$ sin un gran poder de paralelización.

Varias de las secuencias que se recorren son isomorfas. Dada una secuencia, siempre se puede encontrar una secuencia isomorfa que empieza con la etiqueta 1. La demostración es

que si empieza con $v \neq 1$, entonces se puede tomar la secuencia, intercambiar todo v por 1 y todo 1 por v y la secuencia queda con la misma estructura pero con el 1 al inicio.

Por lo tanto no es necesario probar todas las etiquetas para la primera posición, pero no es suficiente: el espacio se redujo a $5^{19} \sim 2 \cdot 10^{13}$. Esto puede reducirse más. Como existen restricciones de largo d, debe existir un camino de largo d en el que no se repite ninguna etiqueta de arista. Por lo tanto, se pueden escoger las primeras d etiquetas como $1, 2, \ldots, d$ pues todo *Cyclic Order Graph* completo minimal puede convertirse en uno que empieza con $1, 2, \ldots, d$ mediante la aplicación de un mapeo. El espacio se redujo a $5^{15} \sim 3 \cdot 10^{10}$. Y claro, se tiene que agregar a esto la verificación, por lo que aún no es viable.

Para continuar se harán las siguientes dos conjeturas obtenidas en base a las soluciones observadas en [4] con respecto a soluciones minimales.

Conjetura 4.1. Existe un *Cyclic Order Graph* completo minimal cuyas etiquetas aparecen el mismo número de veces.

Conjetura 4.2. Existe un *Cyclic Order Graph* completo minimal cuyas etiquetas no se repiten a una distancia menor o igual a $\lfloor d/2 \rfloor$.

Para $d = 5$, el tamaño de un *Cyclic Order Graph* completo minimal es $m = 20$, por lo que cada etiqueta se repite 5 veces. Como las primeras 5 ya están colocadas, debemos escoger dónde colocar las otras 15, 3 de cada valor en $[d]$. Esto es simplemente

$$\frac{15!}{(3!)^5} \sim 2 \cdot 10^8.$$

Y al agregarle la verificación queda un algoritmo que toma aproximadamente 10^{10} operaciones, y sin aprovechar la segunda conjetura.

Para $d = 5$ es posible una búsqueda exhaustiva con este algoritmo.

Para $d = 6$, el tamaño del *Cyclic Order Graph* completo minimal es $m = 42$. Se tienen las primeras 6 etiquetas escogidas; falta colocar las 6 restantes de cada elemento en $[d]$. Sería

$$\frac{36!}{(6!)^6} \sim 3 \cdot 10^{24},$$

completamente inviable. ¿Qué pasa si se aprovecha la segunda conjetura? Las etiquetas no se pueden repetir a una distancia de 3, por lo que en cada posición hay a lo más 3 candidatos; con este análisis queda $3^{36} \sim 2 \cdot 10^{17}$, que aún es imposible.

Se puede intentar computar el número de secuencias mediante inclusión exclusión pero lamentablemente la restricción sobre el número de repeticiones impide obtener una fórmula cerrada para la intersección de eventos, por lo que computarlo (sin más análisis) tomaría al menos $2^{36} \cdot 36$ operaciones.

El número de secuencias válidas a verificar se puede computar mediante Programación Dinámica. Se utiliza memoización y recurrencias.

Se define la recurrencia

$$f_{l,v_1,v_2,v_3,v_4,v_5,v_6,p_1,p_2,p_3}$$

como el número de formas de repartir v_i etiquetas con valor i en las primeras l posiciones, teniendo como las tres últimas etiquetas p_1, p_2 y p_3. Evidentemente no todas las dimensiones son independientes; se puede obtener el valor de l como la suma de los v_i. La respuesta sería

$$\sum_{x \neq y \neq z} f_{36,6,6,6,6,6,6,x,y,z}$$

(se sigue sin considerar que x, y, z tienen las restricciones de las primeras 6 etiquetas que ya están escogidas).

La transición es

$$f_{l,v_1,v_2,v_3,v_4,v_5,v_6,p_1,p_2,p_3} = \sum_{i=1,\ i \neq p_3}^{d} f_{l-1,v_1-\mathbf{1}_{1=p_3},v_2-\mathbf{1}_{2=p_3},v_3-\mathbf{1}_{3=p_3},v_4-\mathbf{1}_{4=p_3},v_5-\mathbf{1}_{5=p_3},v_6-\mathbf{1}_{6=p_3},i,p_1,p_2}.$$

Con caso base

$$f_{0,0,0,0,0,0,0,4,5,6} = 1$$

y

$$f_{l,v_1,v_2,v_3,v_4,v_5,v_6,p_1,p_2,p_3} = 0 \qquad \forall l \leq 0.$$

Eliminando las dimensiones innecesarias, el número de estados posibles para v_i es 7 pues cada etiqueta puede estar entre 0 y 6 veces, y para p_1, p_2, p_3 es 6 pues pueden ser entre 1 y 6. Queda con $7^6 \cdot 3^6$ estados. Suficientemente pequeño para poder ejecutarlo.

Se obtiene un valor de $2 \cdot 10^{15}$. No es viable la búsqueda exhaustiva pues a todas estas secuencias se les tiene que verificar después y no es claro que otros supuestos se pueden tomar que logren disminuir esta cantidad de forma significativa.

4.3. Algoritmos de búsqueda no exhaustiva

La segunda línea de diseño aborda el caso de aridades mayores, donde la búsqueda exhaustiva completa es impracticable. En este contexto se consideran:

- **Algoritmos de Aproximación:** Se diseña un algoritmo que si bien no entrega necesariamente un óptimo, sí entrega una solución con una garantía sobre qué tan lejos está de un óptimo.
- **Técnicas inspiradas en Búsquedas Locales y *Simulated Annealing*:** se diseña una función objetivo que capture el tamaño de la estructura y penalice violaciones de completitud, y se aplican movimientos aleatorios controlados para escapar de óptimos locales.
- **Estrategias voraces (*greedy*):** se exploran construcciones incrementales de *Rings* u *Order Graphs* añadiendo nodos/arcos de manera secuencial según algún criterio local de "ganancia" en completitud versus costo en tamaño.

Si bien varias de estas ideas pueden no igualar las mejores soluciones conocidas en la literatura, su análisis contribuye a entender qué tipos de enfoques son prometedores y cuáles no, y proporciona una base para futuras mejoras.

4.3.1. $O\left(\lg\left(\frac{(d-1)!}{2}\right)\right)$-aproximación para conjuntos de *Rings* CBTW-completos

Como se mostró en la sección de búsqueda exhaustiva, el problema de determinar conjuntos de *Rings* CBTW-completos minimales puede formularse como una instancia del problema de *Set Cover*.

La transformación es sencilla. Dada la dimensión d, el conjunto de restricciones que deben satisfacer los conjuntos de *Rings* CBTW-completos Uts y los conjuntos de las restricciones que cada *Ring* r satisface Uts_r, el problema se reduce a escoger el número de conjuntos de restricciones satisfechas de *Rings* que cubra el universo Uts. Este problema tiene dos algoritmos de aproximación que no son difíciles de implementar.

En la reducción se tienen $\frac{(d-1)!}{2}$ conjuntos (*Rings*) y $|Uts| = d \cdot 2^{d-1}$ elementos en el espacio. Esta aproximación se basa en simplemente agregar el conjunto que maximiza el número de elementos nuevos de forma repetitiva hasta completar el universo.

Cada vez que se quiere agregar un conjunto (*Ring*), se debe iterar sobre todos los conjuntos (*Rings*) que no se han agregado a la solución, evaluar el número de elementos nuevos que agrega y escoger el *Ring* que maximiza ese valor.

Tiene una complejidad de

$$O\left(\left(\frac{(d-1)!}{2}\right)^2 \cdot \frac{d \cdot 2^{d-1}}{w}\right)$$

pues no se pueden agregar más *Rings* que el total, y en cada una de las veces que se agregan *Rings* se deben recorrer todos los *Rings* no elegidos hasta el momento y evaluar el número de elementos nuevos. Para $d = 7$ este valor es aproximadamente $9 \cdot 10^5$ y para $d = 8$ es 10^8. Está claro que el límite para este algoritmo es $d = 8$. El factor de aproximación es bastante malo; para $d = 8$ es 3,4 y para $d = 7$ es 2,5, es decir, puede ser incluso peor que el doble de un minimal y eso sin considerar que es una aproximación asintótica.

Se puede utilizar el mismo algoritmo para CBW-completos.

4.3.2. Heurísticas para movimientos locales

El problema no exhibe un patrón evidente ni una construcción inductiva clara: los patrones que se observan en los pocos ejemplos minimales conocidos no se extienden a la siguiente dimensión, donde el tamaño minimal aún no está establecido. Avanzar en esa dirección requiere disponer de más ejemplos y de ejemplos de mayor tamaño.

Debido a ello se desarrollan heurísticas para modificar de forma local distintas estructuras en lugar de definir una heurística para encontrar una solución.

Modificaciones locales de la estructura permiten recorrer el espacio de búsqueda según un criterio definido. En esta sección solo se mostrarán las diferentes definiciones de vecindad de una estructura para utilizarlas en futuros algoritmos.

Las vecindades desarrolladas para *Rings*

- **Intercambio de posiciones adyacentes, N_1.** Es decir, dos ciclos son vecinos si se puede obtener uno a partir de intercambiar dos vértices adyacentes del otro.

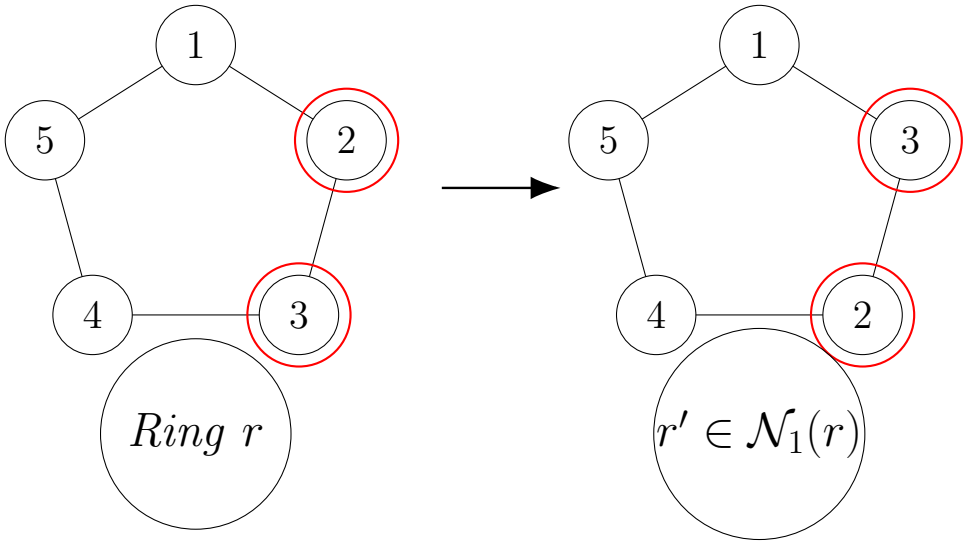

Figura 4.1: Vecindad $\mathcal{N}_1$: intercambio de dos vértices adyacentes en un *Ring*.

- **Traslación de vértice, N_2.** Es decir, dos ciclos son vecinos si se puede obtener uno a partir de eliminar un vértice y volver a colocarlo en otra posición.

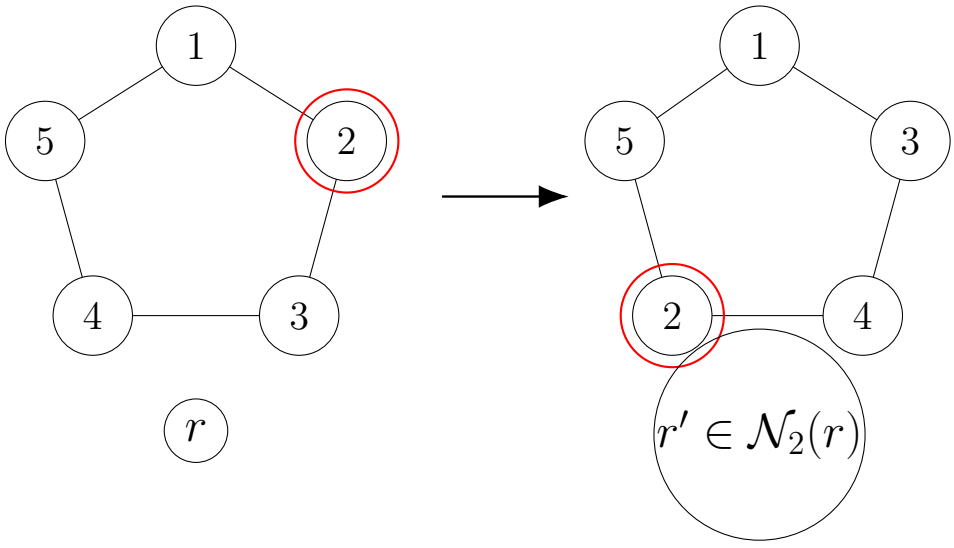

Figura 4.2: Vecindad $\mathcal{N}_2$: traslación de un vértice en un *Ring*.

- **Reverso de sección,** N_3**.** Es decir, dos ciclos son vecinos si se puede obtener uno a partir de seleccionar una sección contigua del otro y darle la vuelta o colocarla en reverso.

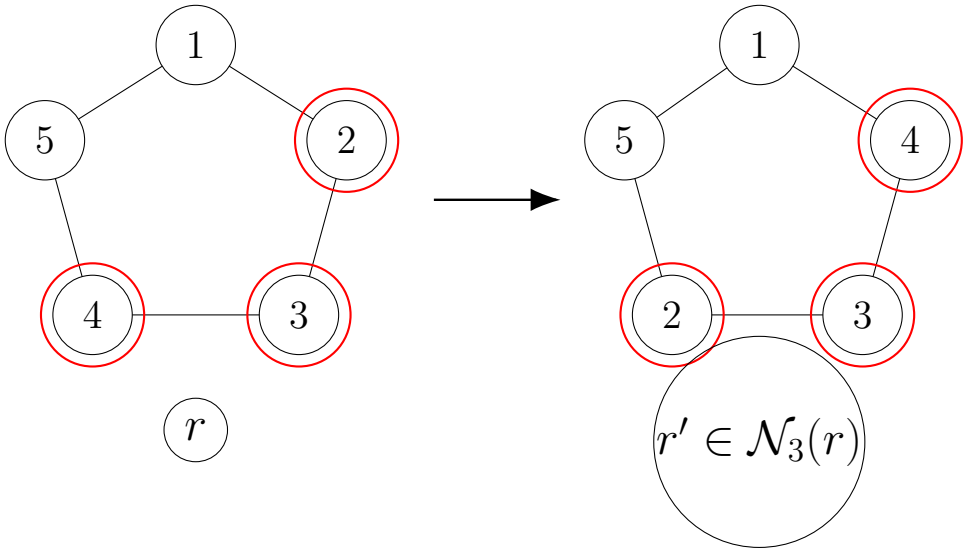

Figura 4.3: Vecindad $\mathcal{N}_3$: reverso de una sección contigua en un *Ring*.

- **Permutación de subsecuencia de largo** l**,** $N_{4,l}$**.** Es decir, dos ciclos son vecinos si se puede obtener uno a partir de seleccionar una subsecuencia de largo l del otro y permutarla.

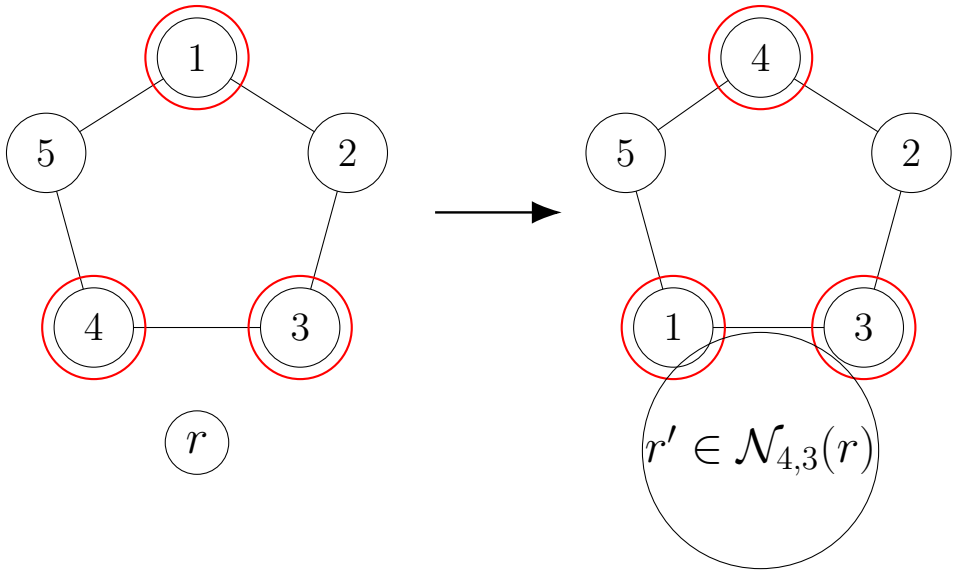

Figura 4.4: Vecindad $\mathcal{N}_{4,3}$: permutación de una subsecuencia contigua de largo l (en el ejemplo, $l = 3$).

Las vecindades desarrolladas para Conjuntos de *Rings*

- **Intercambio de posiciones adyacentes en *Ring* interno,** $\mathcal{CN}_1$**.** Es decir, dos conjuntos de *Rings* son vecinos si uno se puede obtener a partir de cambiar un *Ring* r

interno por un vecino en $\mathcal{N}_1(r)$ del otro.

- **Traslación de vértice en *Ring* interno, $\mathcal{CN}_2$.** Es decir, dos conjuntos de *Rings* son vecinos si uno se puede obtener a partir de cambiar un *Ring* r interno por un vecino en $\mathcal{N}_2(r)$ del otro.
- **Reverso de sección en *Ring* interno, $\mathcal{CN}_3$.** Es decir, dos conjuntos de *Rings* son vecinos si uno se puede obtener a partir de cambiar un *Ring* r interno por un vecino en $\mathcal{N}_3(r)$ del otro.
- **Permutación de subsecuencia de largo l en *Ring* interno, $\mathcal{CN}_{4,l}$.** Es decir, dos conjuntos de *Rings* son vecinos si uno se puede obtener a partir de cambiar un *Ring* r interno por un vecino en $\mathcal{N}_{4,l}(r)$ del otro.
- **Intercambio de *Ring*, $\mathcal{CN}_5$.** Es decir, dos conjuntos de *Rings* son vecinos si uno se puede obtener a partir de cambiar un *Ring* r interno por cualquier otro *Ring*.

Las vecindades desarrolladas para *Cyclic Order Graphs*

- **Intercambio de etiquetas adyacentes, $\mathcal{CG}_1$.** Es decir, dos *Cyclic Order Graphs* son vecinos si se puede obtener uno a partir de intercambiar dos etiquetas de aristas adyacentes del otro.
- **Traslación de etiquetas, $\mathcal{GC}_2$.** Es decir, dos *Cyclic Order Graphs* son vecinos si se puede obtener uno a partir de eliminar un vértice y volver a colocarlo en otra posición.
- **Reverso de sección, $\mathcal{N}_3$.** Es decir, dos *Cyclic Order Graphs* son vecinos si se puede obtener uno a partir de seleccionar una sección contigua del otro y darle la vuelta o colocarla en reverso.
- **Permutación de subsecuencia de largo l, $\mathcal{N}_{4,l}$.** Es decir, dos *Cyclic Order Graphs* son vecinos si se puede obtener uno a partir de seleccionar una subsecuencia de largo l del otro y permutarla.

4.3.3. Búsquedas Locales

Se implementa un algoritmo por tipo de vecindad y por tipo de estructura (Conjuntos de *Rings* CBW-completos, conjuntos de *Rings* CBTW-completos y *Cyclic Order Graphs* completos).

Se define $E(G)$, donde G es uno de los tipos de estructuras trabajados, como el nivel de energía de la estructura G y su valor es igual al número de restricciones que la estructura no satisface. El objetivo es obtener una estructura con energía igual a 0.

La búsqueda local busca, sobre toda la vecindad utilizada, la estructura que tenga menos energía y, de tener menos energía que la estructura actual, cambiarla. Y seguir así hasta que no tenga vecinos con menos energía. Si llega a no tener vecinos con menos energía y su energía no es 0, entonces aumenta el número de *Rings* en el caso de conjuntos de *Rings* o aumenta un nodo al ciclo en el caso de *Cyclic Order Graphs*.

Cada iteración recorre toda la vecindad y evalúa la energía de cada vecino. Aunque esto puede parecer caro, en la práctica no lo es: la energía está acotada por $|Uts|$, que para las dimensiones aquí estudiadas es del orden de unos pocos miles, y el número de iteraciones antes de quedar atrapado en un óptimo local no supera 200 para $d \leq 8$.

4.3.4. *Simulated Annealing*

Simulated Annealing es una técnica de optimización combinatorial muy potente para problemas combinatoriales con varios óptimos locales. Lo que hace *Simulated Annealing* es parecido a lo que hace la búsqueda local, pero permitiendo un escape una vez se llega a un óptimo local. Además, no recorre toda la vecindad de la estructura actual, sino que escoge una de manera aleatoria y decide moverse con cierta probabilidad.

Se utiliza la misma noción de energía en la búsqueda local, pero de otra manera. Se define una función que decide si cambiar la estructura actual o no:

$$P(E, E_n, T) = \left[U[\cdot, \infty] \leq \exp\left(-\frac{E_n - E}{T} \right) \right].$$

En este caso E es la energía de la estructura actual, E_n la energía de la estructura nueva y T es un parámetro de temperatura que cambia a medida que aumentan las iteraciones, se "enfría". El propósito es permitir escapar de un óptimo local y disminuir la probabilidad de escape a medida que aumentan las iteraciones y la temperatura disminuye.

Por cada iteración la temperatura cambia con

$$T = \alpha \cdot T$$

donde α es el parámetro de enfriamiento menor a 1. Una vez que la temperatura se hace menor a 1, se termina y se entrega la estructura con la menor energía que se haya visto en el total de iteraciones.

Una vez que termina, se ejecuta de nuevo con la temperatura reiniciada y la estructura actual la mejor que se haya visto. Esto se hace las veces que se quiera, es decir, se puede tener ejecutando hasta que uno decida terminar la ejecución y observar la mejor estructura vista hasta el momento.

En este caso se decide que si la mejor energía vista hasta el momento no disminuye en 500 ejecuciones de *Simulated Annealing*, se aumenta la cantidad de vértices.

Algoritmo 4 Búsqueda exhaustiva de un conjunto de *Rings* CBTW-completo de tamaño mínimo (§ 4.1).

Require: Dimensión d; lista de *Rings* canónicos $r_0, r_1, \ldots, r_{K-1}$ en orden lexicográfico, con r_0 el *Ring* identidad $(1, 2, \ldots, d)$; máscaras precomputadas $\mathrm{Mask}(r_j)$; universo Uts; cota superior $m_{\text{máx}}$ del tamaño buscado.

Ensure: Un conjunto $R^\star \subseteq \{r_0, \ldots, r_{K-1}\}$ CBTW-completo de tamaño mínimo, con $|R^\star| \leq m_{\text{máx}}$, o $\perp$ si no existe.

1: $mejor \leftarrow \perp$
2: $mejorTam \leftarrow m_{\text{máx}} + 1$
3: $R \leftarrow \{\, r_0 \,\}$ ▷ Ruptura de simetrías: fijar el *Ring* identidad
4: $rem \leftarrow Uts \,\mathrm{AND}\, \overline{\mathrm{Mask}(r_0)}$
5: $\textsc{Search}(R, rem, 1)$
6: **return** $mejor$

7: **procedure** $\textsc{Search}(R, rem, i)$
8: **if** $rem = \mathbf{0}$ **then** ▷ Se cubrió todo Uts
9: **if** $|R| < mejorTam$ **then**
10: $mejor \leftarrow R$
11: $mejorTam \leftarrow |R|$
12: **end if**
13: **return**
14: **end if**
15: $cap \leftarrow \min\big(m_{\text{máx}},\ mejorTam - 1\big)$ ▷ Tope dinámico
16: **if** $|R| \geq cap$ **then return**
17: **if** $|R| + \textsc{LowerBound}(rem, i) > cap$ **then return** ▷ Poda admisible
18: **for** $j \leftarrow i$ **to** $K - 1$ **do**
19: $cover \leftarrow \mathrm{popcount}\big(rem \,\mathrm{AND}\, \mathrm{Mask}(r_j)\big)$
20: **if** $cover = 0$ **then continue** ▷ r_j no aporta cobertura nueva
21: $R \leftarrow R \cup \{r_j\}$
22: $rem' \leftarrow rem \,\mathrm{AND}\, \overline{\mathrm{Mask}(r_j)}$
23: $\textsc{Search}(R, rem', j + 1)$
24: $R \leftarrow R \setminus \{r_j\}$
25: **end for**
26: **end procedure**

Algoritmo 5 Verificación de completitud de un *Cyclic Order Graph* en $O(m \cdot d)$ (§ 4.2).

Require: Dimensión d; secuencia cíclica de etiquetas $e = (e_0, e_1, \ldots, e_{m-1})$ con $e_i \in [d]$; función de indexación $idx(\cdot, \cdot)$ del universo U.

Ensure: **verdadero** si e describe un COG completo, **falso** en otro caso.

cubierto $\leftarrow \mathbf{0}^{|U|}$
for $t \leftarrow 0$ **to** $m - 1$ **do** ▷ Inicio del camino
    $S \leftarrow \mathbf{0}^d$; *rep* $\leftarrow$ **falso**
    for $\ell \leftarrow 1$ **to** d **do** ▷ Largo del camino (ventana cíclica)
        $j \leftarrow (t + \ell - 1)$ mód m
        $b \leftarrow \text{BIT}(e_j)$
        if S AND $b \neq 0$ **then** *rep* $\leftarrow$ **verdadero**
        $S \leftarrow S$ OR b
        if $\neg$*rep* **and** popcount$(S) = \ell$ **then** ▷ Etiquetas distintas: el multiconjunto $= S$
            cubierto$[\, idx(S, e_t)\,] \leftarrow 1$
            cubierto$[\, idx(S, e_j)\,] \leftarrow 1$
        end if
    end for
end for
for cada $(S, x) \in U$ **do**
    if *cubierto*$[\, idx(S, x)\,] = 0$ **then return falso**
end for
return verdadero

Algoritmo 6 Búsqueda exhaustiva de un *Cyclic Order Graph* completo de tamaño m (§ 4.2), aprovechando las simetrías y, opcionalmente, las Conjeturas 4.1 y 4.2.

Require: Dimensión d; tamaño $m \geq d$; *flags* booleanos C1 (cuenta uniforme por etiqueta) y C2 (sin repetición a distancia $\leq \lfloor d/2 \rfloor$).

Ensure: Una secuencia $e \in [d]^m$ que describe un COG completo, o $\perp$ si la búsqueda agota el espacio filtrado.

if C1 **and** m mód $d \neq 0$ **then**
    return $\perp$
end if
prefijo $\leftarrow [\,1, 2, \ldots, d\,]$ ▷ Simetría: las primeras d etiquetas son $1, 2, \ldots, d$
cuota$[i] \leftarrow m/d$ para todo $i \in [d]$ **si** C1
Restar 1 de *cuota*$[i]$ por cada i ya colocado en *prefijo*
dist $\leftarrow \lfloor d/2 \rfloor$ **si** C2, sino 0
EXTEND(*prefijo*, *cuota*, *dist*)
return *resultado* ▷ Variable global puesta por EXTEND la primera vez que halla solución

procedure EXTEND(e, *cuota*, *dist*)
    if $|e| = m$ **then**
        if C2 **and not** CYCLICMINDIST(e, *dist*) **then return**
        if COG-COMPLETO(e, d) **then**
            resultado $\leftarrow e$; **return**
        end if
        return
    end if
    for $\ell \leftarrow 1$ **to** d **do** ▷ Probar cada etiqueta candidata
        if C1 **and** *cuota*$[\ell] = 0$ **then continue**
        if *dist* > 0 **then**
            for $k \leftarrow 1$ **to** mín(*dist*, $|e|$) **do**
                if $e[\,|e| - k\,] = \ell$ **then continue outer**
            end for
        end if
        $e \leftarrow e \cdot [\,\ell\,]$; *cuota*$[\ell] \leftarrow$ *cuota*$[\ell] - 1$
        EXTEND(e, *cuota*, *dist*)
        if *resultado* $\neq \perp$ **then return**
        $e \leftarrow e$ sin su última etiqueta; *cuota*$[\ell] \leftarrow$ *cuota*$[\ell] + 1$
    end for
end procedure

Algoritmo 7 Aproximación *greedy* para CBTW vía *Set Cover* (§ 4.3.1).

Require: Universo UT (bitmask); lista de *Rings* candidatos $r_0, \ldots, r_{K-1}$ con sus máscaras precomputadas $\text{Mask}(r_j)$.

Ensure: Subconjunto $R \subseteq \{r_0, \ldots, r_{K-1}\}$ que cubre UT (cuando es posible), construido elemento a elemento.

$rem \leftarrow UT$; $R \leftarrow \langle\,\rangle$
while $rem \neq \mathbf{0}$ **do**
  $j^\star \leftarrow \perp$; $best \leftarrow 0$
  for $j \leftarrow 0$ **to** $K-1$ **con** $r_j \notin R$ **do**
    $c \leftarrow \text{popcount}\big(rem \,\text{AND}\, \text{Mask}(r_j)\big)$
    if $c > best$ **then**
      $best \leftarrow c$; $j^\star \leftarrow j$
    end if
  end for
  if $j^\star = \perp$ **then return** R (cobertura imposible con el catálogo dado)
  $R \leftarrow R \cdot [\, r_{j^\star} \,]$
  $rem \leftarrow rem \,\text{AND}\, \overline{\text{Mask}(r_{j^\star})}$
end while
return R

Algoritmo 8 Búsqueda local por mejor vecino (§ 4.3.3), con crecimiento de la estructura cuando se estanca con $E > 0$.

Require: Estructura inicial G_0; función de energía $E(\cdot)$; operador de vecindad $N(\cdot)$; operador de crecimiento $\textsc{Grow}(\cdot)$ (un *Ring* más, o un nodo más en el ciclo); cota de iteraciones $T_{\text{máx}}$.

Ensure: La mejor estructura observada $G^\star$ y su energía $E(G^\star)$.

$G \leftarrow G_0$; $G^\star \leftarrow G_0$; $e \leftarrow E(G_0)$
for $t \leftarrow 1$ **to** $T_{\text{máx}}$ **do**
  if $e = 0$ **then return** $(G, 0)$ ▷ Energía nula: solución completa
  $G' \leftarrow \arg\min_{H \in N(G)} E(H)$
  $e' \leftarrow E(G')$
  if $e' < e$ **then**
    $G \leftarrow G'$; $e \leftarrow e'$
    if $e < E(G^\star)$ **then** $G^\star \leftarrow G$
  else
    $G \leftarrow \textsc{Grow}(G)$ ▷ Estancamiento: agregar un *Ring* / un nodo
    $e \leftarrow E(G)$
    if $e < E(G^\star)$ **then** $G^\star \leftarrow G$
  end if
end for
return $(G^\star, E(G^\star))$

Algoritmo 9 *Simulated Annealing* con reinicio y crecimiento estructural (§ 4.3.4).

Require: Estructura inicial G_0; energía $E(\cdot)$; muestreador de vecino $\textsc{Sample}(\cdot)$; temperatura inicial T_0; coeficiente de enfriamiento $\alpha \in (0,1)$; temperatura final $T_{\text{mín}}$; cota de reinicios sin mejora K (típicamente 500); operador $\textsc{Grow}(\cdot)$.

Ensure: La mejor estructura observada $G^\star$ y su energía.

$G \leftarrow G_0$; $\quad G^\star \leftarrow G_0$; $\quad E^\star \leftarrow E(G_0)$; $\quad \mathit{stale} \leftarrow 0$
repeat
  if $E^\star = 0$ **then return** $(G^\star, 0)$
  $T \leftarrow T_0$
  while $T \geq T_{\text{mín}}$ **do**
    $G' \leftarrow \textsc{Sample}(G)$
    $\Delta \leftarrow E(G') - E(G)$
    if $\Delta \leq 0$ **or** $U[0,1] \leq \exp(-\Delta/T)$ **then**
      $G \leftarrow G'$
      if $E(G) < E^\star$ **then**
        $G^\star \leftarrow G$; $\quad E^\star \leftarrow E(G)$; $\quad \mathit{stale} \leftarrow 0$
      end if
    end if
    $T \leftarrow \alpha \cdot T$
  end while
  $G \leftarrow G^\star$ ▷ Reinicio desde la mejor solución vista
  $\mathit{stale} \leftarrow \mathit{stale} + 1$
  if $\mathit{stale} \geq K$ **then**
    $G \leftarrow \textsc{Grow}(G^\star)$; $\quad \mathit{stale} \leftarrow 0$
    if $E(G) < E^\star$ **then** $G^\star \leftarrow G$; $\quad E^\star \leftarrow E(G)$
  end if
until condición externa de paro
return $(G^\star, E^\star)$

Capítulo 5

Implementación y entorno experimental

5.1. Etapas de implementación

La implementación de los algoritmos se realizó en dos etapas principales.

En la primera fase, se desarrollaron algoritmos secuenciales de búsqueda exhaustiva para *Rings* y *Order Graphs*, sin imponer restricciones fuertes de complejidad, con el objetivo de determinar estructuras completas de tamaño mínimo para dimensiones pequeñas y verificar la correctitud de la formulación. En este contexto, la optimalidad se refiere al tamaño de la estructura bajo las condiciones de completitud definidas para una dimensión d dada, y no al desempeño respecto de una consulta específica. Estos algoritmos permitieron, en particular, obtener todas las soluciones óptimas conocidas para ciertos valores de d, con tiempos de ejecución inferiores a cinco minutos por instancia, lo que facilitó el estudio estructural de las soluciones.

En la segunda fase se desarrollaron e implementaron algoritmos de búsqueda no exhaustiva, debido al inmenso crecimiento de los espacios de búsqueda, un crecimiento tan rápido que la paralelización no fue suficiente para continuar utilizando algoritmos exhaustivos. Varias de estas técnicas, si bien pueden no encontrar buenas soluciones, tienen además otro objetivo: aprender más del problema, identificando qué tipos de estrategias producen mejores resultados y cuáles no muestran avances relevantes.

5.2. Arquitectura de la solución

La arquitectura de la solución se basa en:

- Un módulo central de representación de estructuras (*Rings* y *Order Graphs*), que define las estructuras de datos y operaciones básicas sobre ellas siguiendo el paradigma

orientado a objetos.

- Un módulo para *managers* (`RingManager` y `OrderGraphManager`) donde se implementan los distintos algoritmos de generación de soluciones.
- Un módulo de ejecución experimental, encargado de orquestar las ejecuciones, registrar tiempos, tamaños y resultados, y generar salidas en formatos adecuados para el análisis posterior.
- Una página web simple para visualizar las estructuras.

5.3. Entorno de ejecución y protocolo experimental

5.3.1. Hardware

- Sistema operativo `Windows 11` de 64 bits.
- Notebook `HP`.
- 8 GB de RAM y 500 GB de memoria SSD libre.
- Procesador de 2000 MHz, 8 *cores*.
- Tarjeta gráfica `Nvidia`, la cual dejó de funcionar durante el desarrollo del trabajo.

5.3.2. Decisiones técnicas

Todos los algoritmos de búsqueda fueron implementados en `C++17` utilizando el compilador `gcc 13.2.0`. La razón de ello es que `C++` es uno de los lenguajes más rápidos y accesibles, además de ofrecer facilidades para utilizar programación orientada a objetos. Asimismo, resulta conveniente por su compatibilidad con `CUDA` en caso de utilizar la GPU.

Junto a ello, se utilizó `JavaScript` desde el navegador `Google Chrome` para el desarrollo de la herramienta visual destinada a observar las estructuras.

5.3.3. Experimentación

Los experimentos realizados en esta memoria se centran en un problema de naturaleza combinatoria y estructural. En particular, no se evalúa el desempeño de algoritmos de *join* sobre instancias concretas de datos ni sobre consultas específicas, sino la capacidad de construir estructuras completas o casi completas de tamaño mínimo para una dimensión d dada. Por ello, el parámetro principal de experimentación es la dimensión del problema, mientras que los datos y atributos concretos no intervienen de manera explícita.

Algoritmos de búsqueda exhaustiva

Se ejecutaron los algoritmos de búsqueda exhaustiva para aquellas dimensiones en las que se esperaba un funcionamiento correcto y tiempos de ejecución acotados. Asimismo, se realizaron ejecuciones sobre una dimensión superior al límite teórico estimado para tiempos de ejecución menores a una semana, con el fin de observar el comportamiento del algoritmo fuera de su rango práctico esperado.

Algoritmos de búsqueda no exhaustiva

Se realizaron experimentos para todas las dimensiones consideradas, fijando un límite de ejecución de una semana y utilizando dos instancias por configuración, con el objetivo de evaluar la calidad de las soluciones obtenidas en un horizonte temporal más extenso. Una vez transcurrida una semana, la ejecución del algoritmo se detenía de forma forzada.

Adicionalmente, se realizaron 10 ejecuciones para cada dimensión con un límite temporal de un día, con el fin de comparar el comportamiento de los algoritmos no exhaustivos bajo una restricción de tiempo más exigente.

Capítulo 6

Resultados y discusión

6.1. Resultados de búsqueda exhaustiva en dimensiones pequeñas

Los algoritmos exhaustivos desarrollados son capaces de encontrar todas las soluciones óptimas en ciertos valores de d, con tiempos de ejecución inferiores a diez minutos por instancia. Esto permite:

- Verificar que se iguala el estado del arte en tamaños mínimos de Rings y Order Graphs completos para dimensiones pequeñas.
- Validar empíricamente la correctitud de las implementaciones y de las condiciones de completitud utilizadas.
- Disponer de un conjunto de soluciones de referencia que sirve como base para analizar estructuras generadas por algoritmos heurísticos.

La Tabla 6.1 presenta ejemplos de *Cyclic Order Graphs* completos para distintas dimensiones d. En cada columna se muestra una secuencia de etiquetas de aristas que representa un *Cyclic Order Graph* completo de la dimensión correspondiente. Para $d = 3$, la Tabla 6.1 muestra la única estructura mínima encontrada. Para $d = 4$ y $d = 5$, se listan distintas soluciones completas minimales obtenidas experimentalmente. En cambio, para $d = 6$, se indica que la búsqueda exhaustiva no fue terminada, por lo que no se reportan estructuras completas minimales en esa dimensión.

La lectura de cada secuencia debe entenderse de manera cíclica: las etiquetas aparecen en el orden en que se recorren las aristas del ciclo, y la completitud se verifica a partir de las subsecuencias contiguas inducidas por ese recorrido. De este modo, la Tabla 6.1 resume de forma compacta las soluciones encontradas para dimensiones pequeñas y permite apreciar el rápido crecimiento de la complejidad estructural del problema a medida que aumenta d.

Tabla 6.1: Cyclic Order Graph completos

$d = 3$	$d = 4$	$d = 5$	$d = 6$
1 2 3	1 2 3 4	1 2 3 4 5 1 3 4 1 2 5 3 2 4 1 5 2 4 5 3	*(no termi-*
	1 2 4 3	1 2 3 4 5 1 4 2 3 5 2 1 3 5 4 2 5 1 3 4	*nado)*
	1 2 3 4	1 2 3 4 5 1 4 2 3 5 2 1 4 3 1 5 2 4 5 3	
	1 3 2 4	1 2 3 4 5 1 4 2 3 5 4 2 5 1 3 4 1 2 5 3	
	1 2 3 4	1 2 3 4 5 1 4 2 5 1 3 4 1 2 5 3 2 4 5 3	
	2 1 3 4	1 2 3 4 5 2 3 5 1 4 2 1 3 5 4 1 3 4 2 5	
	1 2 3 4	1 2 3 4 5 3 1 2 4 1 5 2 4 3 1 4 5 2 3 5	
	2 1 4 3	1 2 3 4 5 3 1 2 4 1 5 3 2 5 4 1 3 4 2 5	
		1 2 3 4 5 3 1 2 4 3 1 4 5 2 3 5 1 4 2 5	
		1 2 3 4 5 3 1 4 5 2 3 5 1 4 2 1 3 4 2 5	

6.2. Resultados de Búsqueda no Exhaustiva

6.2.1. Búsqueda Local

Cyclic Order Graphs

Los resultados obtenidos para las distintas vecindades sobre *Cyclic Order Graphs* se presentan a continuación. En esta subsección se compara, para cada tipo de movimiento, el mejor tamaño de solución alcanzado en las dimensiones estudiadas y se identifica si dicho valor coincide o no con el tamaño minimal conocido. En conjunto, estos resultados sugieren la presencia de óptimos locales, en particular para $d = 6$, donde distintas vecindades convergen a soluciones no minimales de tamaño similar.

La Tabla 6.2 resume los mejores resultados obtenidos para cada vecindad.

Tabla 6.2: Mejores tamaños obtenidos para *Cyclic Order Graphs* según vecindad y dimensión.

Vecindad	$d = 4$	$d = 5$	$d = 6$
$\mathcal{CG}_1$	minimal	25 (no minimal)	54 (no minimal)
$\mathcal{CG}_2$	minimal	25 (no minimal)	53 (no minimal)
$\mathcal{CG}_3$	minimal	minimal	53 (no minimal)
$\mathcal{CG}_1 \cup \mathcal{CG}_2 \cup \mathcal{CG}_3$	minimal	minimal	53 (no minimal)

Para la vecindad $\mathcal{CG}_1$, el algoritmo encuentra una solución minimal para $d = 4$. En cambio, para $d = 5$ obtiene una solución de tamaño 25, que no es minimal, y para $d = 6$ una solución de tamaño 54, que tampoco alcanza el mínimo conocido.

Para la vecindad $\mathcal{CG}_2$, el algoritmo encuentra una solución minimal para $d = 4$. Para $d = 5$ obtiene nuevamente una solución de tamaño 25 no minimal, mientras que para $d = 6$

alcanza una solución de tamaño 53, también no minimal.

Para la vecindad $\mathcal{CG}_3$, el algoritmo encuentra soluciones minimales para $d = 4$ y $d = 5$. Sin embargo, para $d = 6$ converge a una solución de tamaño 53, la cual no es minimal y coincide con la mejor solución encontrada mediante la vecindad $\mathcal{CG}_2$.

Finalmente, al combinar las vecindades $\mathcal{CG}_1$, $\mathcal{CG}_2$ y $\mathcal{CG}_3$, se obtienen soluciones minimales para $d = 4$ y $d = 5$, mientras que para $d = 6$ se vuelve a alcanzar la misma solución de tamaño 53. Este comportamiento sugiere que, para dicha dimensión, las estrategias consideradas tienden a estabilizarse en soluciones localmente óptimas sin alcanzar el mínimo global.

A continuación se presentan, a modo de referencia, las mejores soluciones obtenidas para $d = 6$ en las vecindades $\mathcal{CG}_1$ y $\mathcal{CG}_2$.

Solución obtenida con la vecindad $\mathcal{CG}_1$ para $d = 6$ (tamaño 54).

```
1 2 3 4 5 6 2 3 1 4 5 6 1 4 2 5 3 6 1 2 3 6 4 1 5 2 3 4 6 2
5 1 3 5 4 2 6 1 3 4 6 5 1 2 4 3 6 5 1 3 2 4 5 6
```

Solución obtenida con la vecindad $\mathcal{CG}_2$ para $d = 6$ (tamaño 53).

```
1 2 3 4 5 6 1 2 4 5 3 1 2 6 4 3 5 6 1 2 5 3 6 4 2 5 6 4 1 2
3 4 1 5 6 2 3 4 6 1 3 4 5 1 3 6 2 1 5 4 2 3 5
```

Conjuntos de *Rings* BCTW-completos

Utilizando el conjunto completo de vecindades, se logran obtener soluciones para $d \leq 6$. En particular, para $d = 6$ se encontró el conjunto de *Rings* BCTW-completo que se muestra en la Tabla 6.3. Cada fila corresponde a un *Ring*, representado por la permutación cíclica bidireccional de sus etiquetas en forma canónica. En conjunto, estas ocho permutaciones constituyen una solución completa para la dimensión 6 bajo el criterio BCTW.

Tabla 6.3: Conjunto de *Rings* BCTW-completo encontrado para $d = 6$.

Rings de la solución
1 2 3 4 5 6
6 3 1 4 5 2
1 4 2 3 5 6
1 2 4 6 3 5
2 3 1 5 4 6
1 2 5 3 4 6
2 1 3 4 5 6
6 3 2 4 5 1

Para $d = 7$, en cambio, el algoritmo no logra encontrar soluciones, incluso al permitir conjuntos con un mayor número de *Rings*. Este comportamiento sugiere que, para dicha dimensión, la búsqueda podría quedar atrapada sistemáticamente en óptimos locales.

6.3. Simulated Annealing

Cyclic Order Graphs

Mediante *Simulated Annealing* se obtuvo, para $d = 6$, un *Cyclic Order Graph* incompleto de tamaño 42 al que solo le faltan dos restricciones por satisfacer. La solución encontrada se representa mediante la secuencia de etiquetas de aristas que define el ciclo:

```
1, 2, 3, 4, 5, 6, 2, 4, 1, 5, 3, 6, 2, 1, 5, 3, 4, 1, 2, 5, 6, 1,
3, 4, 6, 5, 1, 3, 2, 5, 4, 1, 6, 3, 2, 4, 6, 3, 5, 2, 4, 6
```

Asimismo, se obtuvo un *Cyclic Order Graph* completo de tamaño 51, representado por la siguiente secuencia de etiquetas de aristas:

```
1, 2, 3, 4, 5, 6, 2, 1, 4, 3, 5, 6, 2, 1, 6, 4, 2, 5, 1, 6, 3, 2,
4, 1, 5, 3, 6, 2, 4, 3, 6, 1, 4, 5, 3, 1, 3, 5, 2, 6, 1, 3, 2, 5,
4, 2, 1, 3, 4, 6, 5
```

Estos resultados muestran que, para $d = 6$, las soluciones obtenidas mediante *Simulated Annealing* mejoran progresivamente al aumentar el tiempo de ejecución. En particular, al restringir el algoritmo para que no aumentara el número de nodos, se logró obtener en una semana una solución muy cercana al mínimo conocido para esa dimensión.

Desde un punto de vista empírico, *Simulated Annealing* fue la técnica que produjo las mejores soluciones entre los algoritmos evaluados en esta memoria, tanto por la calidad de las estructuras encontradas como por su capacidad de aproximarse al mínimo conocido en tiempos de ejecución razonables.

Como referencia en dimensiones mayores, para $d = 7$ se reporta que el ciclo más pequeño encontrado mediante *gradient descent*[1] tiene tamaño 83, siendo uno menos que el menor conjunto de *rings* hallado y, a la vez, mayor que la cota inferior de 70. También se conjetura que existen ciclos más pequeños cercanos a dicha cota, pero la búsqueda exhaustiva realizada en ese trabajo muestra que no existe un ciclo de tamaño 70 para $d = 7$ [5].[2]

[1]En este contexto, *gradient descent* se refiere a un procedimiento iterativo de mejora local que modifica progresivamente la solución actual buscando reducir una función objetivo, por ejemplo el tamaño de la estructura o el número de restricciones no satisfechas.

[2]Esta información fue proporcionada por el Profesor Dr. Gonzalo Navarro después del proceso de revisión de este manuscrito.

6.4. Limitaciones de Enfoques Voraces

Se exploraron algoritmos voraces para la construcción de Rings, con el objetivo de producir soluciones razonablemente buenas para valores mayores de d. Sin embargo, las estructuras generadas no lograron competir con las mejores conocidas en la literatura, lo que llevó a descartar varias variantes de este enfoque. Este resultado negativo es igualmente informativo, en cuanto sugiere que la estructura del espacio de soluciones es lo suficientemente compleja como para no ser capturada adecuadamente por criterios puramente locales. Además, la presencia de óptimos locales es muy grande, la necesidad de implementar un escape más complejo que el implementado se vuelve necesario.

6.5. Discusión general

Es un problema muy difícil que crece demasiado rápido.

En conjunto, los resultados sugieren que es posible igualar y reproducir el estado del arte en dimensiones pequeñas mediante algoritmos exhaustivos bien diseñados. Lamentablemente el espacio crece tan rápido que avanzar una sola dimensión se vuelve un desafío mucho más grande, tanto que imagino que la búsqueda exhaustiva no avanzará en dimensiones por años.

El estado del arte sobre dimensiones más grandes es mucho más difícil de igualar. Se requiere experiencia, potencia de procesador y recursos para dejar el algoritmo ejecutándose por mucho más tiempo. Los óptimos locales son demasiados y hay que saber escapar de ellos. El problema no pareciera tener patrones o una dirección fija para llegar a una solución.

Esto también se ve en $d = 7$: aun cuando existe una cota inferior de 70, los mejores resultados empíricos reportados para ciclos quedan bastante por encima (por ejemplo, tamaño 83 con *gradient descent*), y además la búsqueda exhaustiva descarta que exista un ciclo de tamaño 70. En la práctica, esto refuerza la idea de que la brecha entre cotas teóricas y lo alcanzable con búsquedas se vuelve muy difícil de cerrar a medida que aumenta la dimensión [5].

Alcance de cotas inferiores: ciclos vs. *order graphs* generales. Una actualización posterior a los resultados base de la literatura muestra que la cota inferior formulada para ciclos (Teorema 7.9 en la referencia correspondiente) no se extiende en general a *order graphs* con ramificación ("hairy cycles"). En particular, se reporta un *order graph* de dimensión $d = 6$ con 40 aristas, mientras que la cota inferior derivada para el caso cíclico entregaba 42. Esto indica que los *order graphs* pueden ser estrictamente más pequeños que el menor conjunto de ciclos y, por lo tanto, las comparaciones basadas en cotas demostradas para ciclos deben interpretarse con dicha restricción (véase [14]).

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones generales

Esta memoria abordó el problema del estudio de tamaños mínimos de *Rings* y *Order Graphs* para soportar algoritmos de *join* óptimos en el peor caso, con énfasis en el diseño, implementación y evaluación de algoritmos de búsqueda exhaustiva y no exhaustiva. El problema fue tratado como uno de naturaleza combinatoria y estructural, donde el parámetro central es la aridad d de la relación, y donde el objetivo no consistía en evaluar consultas particulares sobre bases de datos concretas, sino en estudiar estructuras abstractas capaces de soportar múltiples órdenes de atributos con el menor tamaño posible.

El principal aprendizaje de esta memoria es que el problema presenta una explosión combinatoria extremadamente severa, tanto en el caso de conjuntos de *Rings* como en el de *Cyclic Order Graphs*. Los resultados muestran que la búsqueda exhaustiva sí es útil para dimensiones pequeñas, porque permite validar implementaciones, reproducir soluciones óptimas conocidas y estudiar con precisión la estructura de las soluciones. Sin embargo, también muestran que dicha estrategia deja rápidamente de ser viable como enfoque principal a medida que aumenta la dimensión. En este sentido, una conclusión importante del trabajo es que no parece razonable seguir invirtiendo esfuerzo en diseñar algoritmos de búsqueda exhaustiva con el objetivo de escalar a dimensiones mucho mayores, salvo como herramienta de validación o de generación de instancias de referencia.

Un segundo aprendizaje relevante es que los métodos no exhaustivos, aunque no garantizan optimalidad, sí permiten explorar regiones del espacio de búsqueda inaccesibles para la enumeración completa. En particular, dentro de los algoritmos implementados y evaluados en esta memoria, *Simulated Annealing* fue la técnica que produjo las mejores soluciones, especialmente en el caso de *Cyclic Order Graphs* para $d = 6$, donde logró encontrar estructuras muy cercanas al mínimo conocido. En contraste, los enfoques voraces y varias de las búsquedas locales simples tendieron a estabilizarse en óptimos locales, lo que sugiere que los mecanismos explícitos de escape son esenciales para este problema.

Un tercer resultado importante es que, aunque no se mejoraron las cotas conocidas ni se avanzó el estado del arte, sí se generó evidencia computacional útil sobre el comportamiento

del problema y sobre la calidad relativa de distintas estrategias de búsqueda. Esta evidencia permite orientar mejor futuras investigaciones. En particular, el trabajo muestra que ciertas familias de movimientos locales no son suficientes por sí solas para alcanzar buenas soluciones en dimensiones mayores, mientras que estrategias probabilísticas con enfriamiento controlado muestran un potencial considerablemente más alto.

Finalmente, un punto especialmente relevante que se detectó durante el proceso de revisión es que algunas cotas inferiores reportadas en la literatura deben interpretarse con cuidado. En particular, ciertas cotas demostradas para ciclos no se extienden automáticamente al caso de *Order Graphs* generales con ramificación. El hecho de que se haya reportado un *Order Graph* de dimensión $d = 6$ con 40 aristas, por debajo de la cota inferior 42 derivada para ciclos, muestra que los *Order Graphs* generales pueden ser estrictamente más pequeños que el menor conjunto de ciclos. Este punto no solo refina la interpretación del estado del arte, sino que también abre una línea de investigación teórica y computacional especialmente importante para trabajos futuros.

7.2. Cumplimiento del objetivo general y de los objetivos específicos

El objetivo general de esta memoria fue diseñar, implementar y evaluar algoritmos de búsqueda exhaustiva y no exhaustiva para estudiar los tamaños de *Rings* y *Order Graphs* que permiten algoritmos WCO, con y sin *trie switching*, para distintas aridades, generando evidencia computacional que contribuya a caracterizar sus tamaños mínimos y las limitaciones de las técnicas actuales.

A la luz de los resultados obtenidos, puede afirmarse que este objetivo general fue cumplido de manera satisfactoria. En efecto, se diseñaron e implementaron algoritmos para distintos tipos de estructuras, se evaluó su comportamiento en múltiples dimensiones y se obtuvieron resultados concretos tanto en el caso exhaustivo como en el no exhaustivo. Asimismo, se logró caracterizar empíricamente el rango práctico de aplicabilidad de los distintos enfoques y contrastar su desempeño frente a referencias del estado del arte.

Respecto de los objetivos específicos, se observa lo siguiente:

1. **Diseñar algoritmos de búsqueda exhaustiva y no exhaustiva para el estudio de tamaños mínimos de conjuntos de *Rings*.**

   Este objetivo fue cumplido. Se desarrollaron algoritmos exhaustivos para conjuntos de *Rings* bajo criterios de completitud CBW y CBTW, junto con estrategias no exhaustivas basadas en aproximación, búsqueda local y metaheurísticas. En dimensiones pequeñas, los métodos exhaustivos permitieron recuperar soluciones óptimas conocidas. En dimensiones mayores, los métodos no exhaustivos permitieron explorar el espacio de soluciones y comparar distintas estrategias, aunque sin alcanzar mejoras sobre el estado del arte.

2. **Diseñar algoritmos de búsqueda exhaustiva y no exhaustiva para el estudio**

de tamaños mínimos de *Order Graphs*.

Este objetivo fue cumplido en el alcance efectivamente asumido por la memoria. Dada la complejidad del problema general, el trabajo se focalizó en *Cyclic Order Graphs*, para los cuales se desarrollaron algoritmos de verificación, búsqueda exhaustiva y búsqueda no exhaustiva. Esta decisión metodológica permitió mantener el problema en un rango abordable sin perder su interés combinatorio central. En ese marco, se obtuvieron resultados significativos, incluyendo soluciones muy cercanas al mínimo conocido para $d = 6$ mediante *Simulated Annealing*. En cambio, el caso de *Order Graphs* generales quedó fuera del alcance efectivo del trabajo.

3. **Implementar de forma eficiente los algoritmos anteriores en arquitecturas secuenciales (CPU).**

   Este objetivo fue cumplido. La implementación en `C++` permitió desarrollar versiones eficientes de los algoritmos, apoyadas en representaciones compactas mediante *bitmasks*, operaciones bit a bit y criterios de poda. Ello hizo posible ejecutar búsquedas exhaustivas en dimensiones pequeñas y realizar campañas experimentales extensas para métodos no exhaustivos dentro de límites temporales razonables.

En síntesis, la memoria cumple con los objetivos declarados en la medida en que logra estudiar el problema, construir herramientas computacionales para explorarlo y producir evidencia experimental relevante. Lo que no se logró fue mejorar cotas conocidas ni superar el estado del arte, pero ello tampoco invalida el cumplimiento del propósito central del trabajo, que era precisamente estudiar el problema y producir información útil para avanzar en su comprensión.

7.3. Aportes del trabajo

Los principales aportes de esta memoria pueden resumirse en los siguientes puntos:

- Se formuló el problema de búsqueda de estructuras mínimas para *Rings* y *Order Graphs* desde una perspectiva combinatoria explícita, separándolo de la evaluación de consultas concretas sobre datos específicos.
- Se diseñaron e implementaron algoritmos exhaustivos para conjuntos de *Rings* y *Cyclic Order Graphs*, junto con criterios de verificación y poda que permiten estudiar de forma precisa las dimensiones pequeñas.
- Se desarrollaron e implementaron múltiples estrategias no exhaustivas, incluyendo aproximación voraz, búsqueda local y *Simulated Annealing*, lo que permitió comparar su comportamiento relativo y evaluar cuáles enfoques resultan más prometedores.
- Se recuperaron soluciones óptimas conocidas en dimensiones pequeñas, validando empíricamente tanto la correctitud de las implementaciones como la formulación del problema.
- Se obtuvo evidencia computacional concreta sobre la presencia de óptimos locales y sobre la dificultad de escapar de ellos mediante vecindades simples, especialmente en dimensiones mayores.

- Se identificó a *Simulated Annealing* como la técnica de mejor desempeño empírico entre las probadas, lo que constituye una guía clara para trabajos posteriores.
- Se incorporó una observación importante sobre la interpretación de cotas inferiores en la literatura: las cotas válidas para ciclos no deben asumirse automáticamente como válidas para *Order Graphs* generales con ramificación.

Estos aportes son principalmente metodológicos, algorítmicos y experimentales. No constituyen una mejora directa del estado del arte, pero sí generan una base más clara para continuar investigando el problema con mejores herramientas y con una comprensión más precisa de sus dificultades reales.

7.4. Limitaciones del trabajo

A pesar de los resultados obtenidos, esta memoria presenta limitaciones que deben ser reconocidas explícitamente.

En primer lugar, el trabajo se restringe al estudio combinatorio y estructural del problema. En consecuencia, no se evalúa la integración efectiva de las estructuras encontradas en sistemas gestores de bases de datos, ni su impacto concreto en tiempos de ejecución de consultas reales. Los resultados deben interpretarse, por tanto, como evidencia sobre la estructura del problema y no como una validación directa de desempeño en entornos operacionales.

En segundo lugar, el análisis de *Order Graphs* se acota a *Cyclic Order Graphs*. Esta decisión fue metodológicamente razonable dada la complejidad del caso general, pero limita el alcance de las conclusiones respecto de *Order Graphs* con ramificación.

En tercer lugar, los métodos no exhaustivos implementados, aunque útiles para explorar el espacio de soluciones, no garantizan optimalidad. Por ello, las mejores soluciones encontradas para dimensiones mayores deben interpretarse como cotas superiores empíricas y no como mínimos comprobados.

En cuarto lugar, la evaluación experimental se realizó en un entorno computacional acotado y con recursos limitados. Esto no invalida los resultados, pero sí sugiere que ciertas estrategias, especialmente las paralelas o las de ejecución prolongada, podrían beneficiarse de una infraestructura más robusta.

7.5. Trabajo futuro

A partir de los resultados obtenidos y de las limitaciones identificadas, se abren varias líneas de trabajo futuro.

7.5.1. Cotas inferiores para *Order Graphs* generales

Una línea especialmente importante consiste en caracterizar cotas inferiores que sean válidas en el caso general de *Order Graphs* con ramificación, y no solo para ciclos. La evidencia discutida en esta memoria sugiere que esta distinción no es meramente técnica, sino estructural. Avanzar en esta dirección permitiría clarificar qué parte de las barreras conocidas es propia del caso cíclico y qué parte se mantiene en el caso general.

7.5.2. Construcciones que aprovechen ramificación

Otra línea natural consiste en estudiar construcciones sistemáticas que aprovechen explícitamente la ramificación para obtener *Order Graphs* más pequeños. Esto incluye tanto el diseño de nuevos esquemas constructivos como el desarrollo de mecanismos que permitan verificar completitud y comparar de manera justa estas estructuras con los mejores resultados conocidos para ciclos y conjuntos de *Rings*.

7.5.3. Mejora de estrategias de búsqueda no exhaustiva

Los resultados obtenidos indican que el trabajo futuro debería concentrarse principalmente en métodos no exhaustivos. En particular, sería valioso diseñar estrategias guiadas por patrones estructurales observados en soluciones buenas o casi mínimas, en lugar de insistir en variantes de enumeración exhaustiva. En esta línea, parece especialmente prometedor desarrollar:

- vecindades más informativas,
- mecanismos de escape de óptimos locales más robustos,
- heurísticas guiadas por propiedades estructurales de soluciones parciales,
- reinicios adaptativos, y
- combinaciones híbridas entre búsqueda local y metaheurísticas.

7.5.4. Paralelización y uso de hardware especializado

Dada la explosión combinatoria observada, cualquier intento serio de avanzar en dimensiones mayores debería apoyarse en paralelización y en una mejor utilización del hardware disponible. En particular, el uso de múltiples núcleos, `GPU` u otras arquitecturas paralelas podría ampliar significativamente el rango de dimensiones abordables, tanto para búsqueda exhaustiva acotada como para metaheurísticas de largo aliento.

7.5.5. Refinamiento de cotas y certificación de soluciones

También sería deseable refinar cotas superiores e inferiores mediante una combinación de análisis teórico y evidencia experimental más extensa. En paralelo, sería útil desarrollar métodos que acompañen las soluciones encontradas con mecanismos de verificación más sistemáticos o incluso con certificados de completitud para dimensiones moderadas.

7.5.6. Integración con planificación de consultas

Finalmente, una línea de trabajo de proyección aplicada consiste en estudiar cómo las estructuras analizadas en esta memoria podrían incorporarse en procesos de planificación de consultas dentro de sistemas gestores de bases de datos. Esto supone conectar el estudio combinatorio de *Rings* y *Order Graphs* con modelos de costo, selección de órdenes de atributos y decisiones de indexación en entornos reales.

7.6. Conclusión final

En conclusión, esta memoria muestra que el estudio de tamaños mínimos de *Rings* y *Order Graphs* constituye un problema combinatorio difícil, pero científicamente relevante. Los resultados obtenidos confirman que las dimensiones pequeñas pueden abordarse mediante búsqueda exhaustiva, pero también muestran que esta estrategia deja de ser práctica rápidamente y que el avance hacia dimensiones mayores depende de métodos no exhaustivos mejor diseñados.

Aunque no se logró mejorar el estado del arte, sí se aprendió de forma concreta qué estrategias son útiles, cuáles resultan insuficientes y qué aspectos del problema requieren una reformulación conceptual más cuidadosa. En particular, la memoria permite concluir que el esfuerzo futuro debe dirigirse menos a extender la búsqueda exhaustiva y más a entender la estructura de las soluciones, diseñar heurísticas guiadas por ese entendimiento y desarrollar herramientas para el caso general de *Order Graphs* con ramificación.

Desde esa perspectiva, el trabajo cumple con su propósito: estudiar el problema de manera rigurosa, generar evidencia computacional útil y dejar una base más sólida para investigaciones posteriores.

Bibliografía

[1] Milani Mhedhbi and. Optimizing one-time andcontinuous subgraphqueries using worst-case optimaljoins. *ACM Transactions on Database Systems (TODS)*, 46:1–45, 2021.

[2] Diego Arroyuelo, Daniela Campos, Adrián Gómez-Brandón, Gonzalo Navarro, Carlos Rojas, and Domagoj Vrgoc. Compact ltj: Space & time efficient leapfrog triejoin on graph databases. *The VLDB Journal*, pages 34–67, 2025.

[3] Diego Arroyuelo et al. The ring: Worst-case optimal joins in graph databases using (almost) no extra space. *ACM Transactions on Database Systems*, 49:1–45, 2024.

[4] Diego Arroyuelo, Adrián Gómez-Brandón, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, and Adrián Soto. The ring: Worst-case optimal joins in graph databases using (almost) no extra space. *ACM Transactions on Database Systems*, 49(2):5:1–5:45, 2024.

[5] Diego Arroyuelo, Adrián Gómez-Brandón, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, and Adrián Soto. The ring: Worst-case optimal joins in graph databases using (almost) no extra space. *ACM Transactions on Database Systems*, 49(2):5:1–5:45, 2024.

[6] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[7] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. 2017. Versión arXiv consultada en 2025.

[8] Ayoub Berdai and Dalila Chiadmi. Attempts in worst-case optimal joins on relational data systems: A literature survey. In *2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, pages 01–08, 2023.

[9] Michael Freitag et al. Adopting worst-case optimal joins in a relational database system. In *Proceedings of the VLDB Endowment*, volume 13, pages 1891–1904, 2020.

[10] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 841–850. SIAM, 2003.

[11] Ioannis Karatzas. and Steven E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, Berlin, 2nd edition, 2000.

[12] Milani Mhedhbi and Semih Salihoglu. Optimizing subgraph queries by combining binary and worst-case optimal joins. In *Proceedings of the VLDB Endowmen*, volume 12, pages 1692–1704, 2019.

[13] Gonzalo Navarro. Wavelet trees for all. In *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching (CPM 2012)*, volume 7354 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2012.

[14] Gonzalo Navarro. Fix to The Ring: Worst-case optimal joins in graph databases using (almost) no extra space, n.d. Página web personal.

[15] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2012)*, pages 37–48, Scottsdale, AZ, USA, 2012. ACM.

[16] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM*, 65(3):16:1–16:40, 2018.

[17] Philip Protter. *Stochastic Integration and Differential Equations*. Springer, 1990.

[18] Daniel Revuz and Marc Yor. *Continuous martingales and Brownian motion*. Number 293 in Grundlehren der mathematischen Wissenschaften. Springer, Berlin [u.a.], 3. ed edition, 1999.

[19] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of the 17th International Conference on Database Theory (ICDT 2014)*, pages 96–106, Athens, Greece, 2014. OpenProceedings.org.

[20] Junxiong Wang, Immanuel Trummer, Ahmet Kara, and Dan Olteanu. Adopt: Adaptively optimizing attribute orders for worst-case optimal join algorithms. volume 16, pages 2805–2817, 2023.

[21] Yisu Remy Wang, Max Willsey, and Dan Suciu. Free join: Unifying worst-case optimal and traditional joins. In *Proceedings of the ACM on Management of Data*, volume 1, pages 1–23, 2023.