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

Extensión de algoritmo de regular path queries con matrices de adyacencia

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

Cristóbal Álvarez

PROFESOR GUÍA:
Gonzalo Navarro

SANTIAGO DE CHILE
2023

1. Introducción

"Las regular path queries (consulta de rutas regulares o RPQ como se denominarán en adelante), que son básicamente expresiones regulares que se comparan con las etiquetas (o labels) de rutas en grafos etiquetados, son el núcleo de lenguajes de consulta de bases de datos de grafos como SPARQL". Extracto traducido al español [2]. Es decir, son búsquedas de caminos que cumplen ciertas condiciones dentro de un grafo[5].

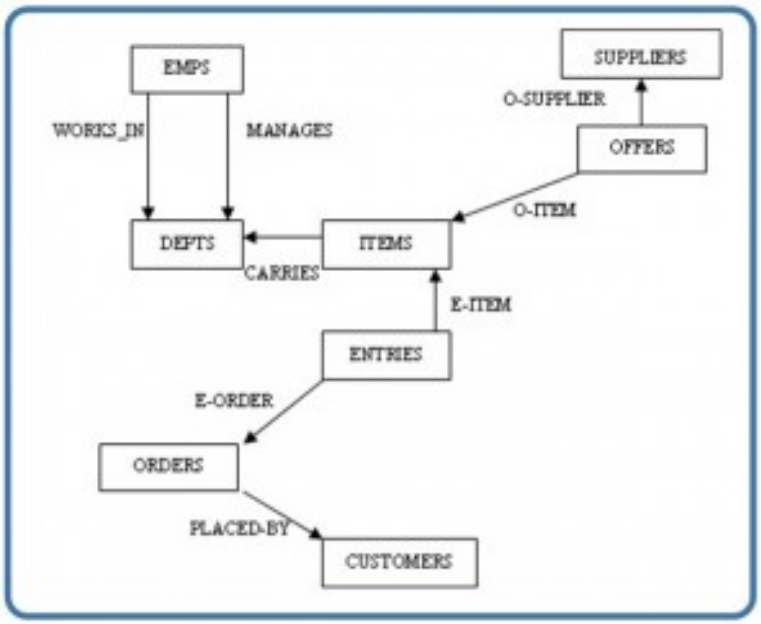

Figura 1: Ejemplo de un grafo etiquetado.

Ya existen varios lenguajes para resolver estas consultas como SPARQL, pero un objeto que no es muy usado debido a la poca fiabilidad que se le tiene en términos de tiempo, son las matrices de adyacencia, estas son, como su nombre indica, matrices que determinan si hay una conexión entre dos nodos, es decir, si son adyacentes, colocando un 1 en la posición (i,j) si hay una arista entre i y j, o un 0 si no.

En general, se evita trabajar con matrices, debido a que, por su forma, comúnmente les toma un tiempo de orden 'n' al cuadrado o mayor, con 'n' el tamaño de las filas y columnas de la matriz. Sin embargo, esto ha sido estudiado [2] y aunque el orden de tiempo pueda mantenerse similar, con las nuevas técnicas se puede disminuir este tiempo, a que no sea una función dependiente del tamaño de la matriz, sino, de 'm', con 'm' la cantidad de 1s en esta, así, para casos en que la matriz está cercano a lo vacío, le tomará mucho menos tiempo que una matriz llena, permitiéndole ser competitiva contra otros métodos. Además, la matriz de adyacencia cumple con varias propiedades que puede facilitar la ejecución de las consultas más complejas, es fácilmente operable entre una matriz y otra, pues, pueden sumarse entre ellas, multiplicarse, existe la matriz identidad, entre otras operaciones.

De hecho, ya fue presentado una técnica, o algoritmo, que usa matrices de adyacencia [2], y que en sus resultados ha probado que su eficiencia es cercana a la de los mejores algoritmos, tanto en espacio de uso, como en tiempo de ejecución. Tal como se mencionaba anteriormente,

en consultas simples que toman menor tiempo, a la técnica le toma mayor tiempo, pero no en gran escala, mientras que, para consultas complejas, se compara con los mejores algoritmos que ocupan más espacio que este.

Una operación interesante cuando se analizan las RPQs, es la de negar las etiquetas (o labels en inglés), esto quiere decir, que dentro de un grafo si hay aristas con una etiqueta en específica y se tiene una RPQ con esa etiqueta negada, la consulta ignorará todas esas aristas con esa etiqueta. Es complicado entonces, representar esta negación dentro de una matriz de adyacencia, pero, ya se aplica en lenguajes como SPARQL [2]

En esta tesis, se busca complementar la idea de usar matrices de adyacencia para ejecutar las RPQs, añadiéndole la facultad de negar las etiquetas, manteniendo su capacidad.

2. Situación Actual

El tema con respecto a trazar eficientemente RPQs se podría decir que es popular, debido a eso, ya existen lenguajes como SPARQL que tiene implementaciones para trabajar con RPQs, mientras otros han formado nuevos algoritmos, como los de unión multidireccional óptimos en el peor de los casos [3], o la técnica de Ring [1].

Pero ninguno ha usado matrices de adyacencia para generar resultados, pues, como se mencionó anteriormente, al ser matrices, no se esperaban buenos resultados de tiempo, pero, el algoritmo reciente que usa matrices de adyacencia [2], con técnicas novedosas como la multiplicación de Shoor [6],ha probado dar excelentes resultados con respecto al orden de tiempo para consultas, tanto simples, como complejas.

Antes de continuar con los resultados, hay que repasar ciertos conceptos importantes. La matriz de adyacencia, como se mencionó antes, corresponde a una matriz n x n, tal que la posición (i,j) es 1 si hay una arista que une a 'i', 'j' y 0 si es que no. Un árbol k^2, es un árbol tal que cada nodo tiene 2^k hijos, para uso de este algoritmo, $k = 2$. La representación LOUDS de un árbol, es una lista con los valores del árbol, tomando en orden de arriba a abajo, de izquierda a derecha. La relación de los tres se ejemplifica en la siguiente imagen:

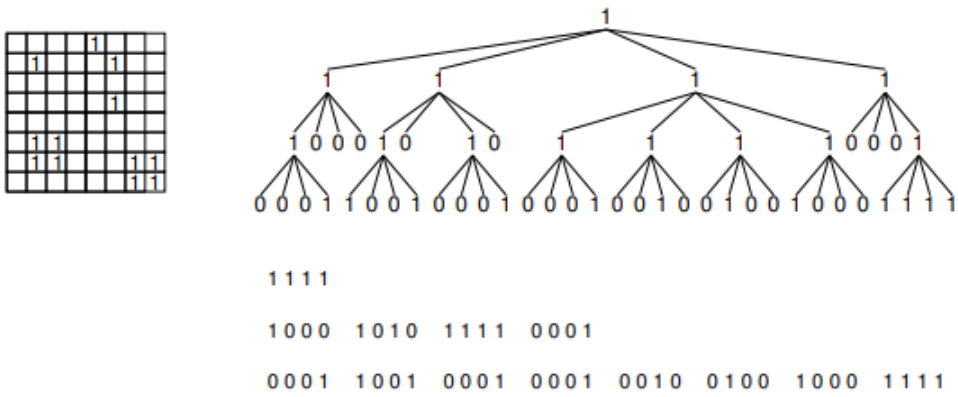

Figura 2: 'Una matriz booleana (arriba, izquierda), su correspondiente representación k^2 tree(arriba, derecha) y la correspondiente representación vectorial del árbol (abajo)'. Traducción al español de figura 2 [2]. La última representación, al ser unida en una lista, es la representación LOUDS.

La técnica de matrices de adyacencia, fue comparada con otros algoritmos, como Ring, Virtuoso, entre otros. Los resultados fueron los siguientes.

Table 2 Index space (in bytes per triple), indexing time (in hours), and some statistics on the query times (in seconds). Row "Timeouts" counts queries that take over 60 seconds or are rejected by the planner as too costly. 2RPQs with some constant node are indicated by c, and without by ¬c.

	k^2-tree	k^2-tree-p	Baseline	Ring	Ring$_{AB}$	Jena	Virtuoso	Blazegraph
Index space (bpt)	4.33	4.33	16.45	16.41	27.93	95.83	60.07	90.79
Indexing time (hs)	0.3	0.3	5.5	7.5	8.3	37.4	3.0	39.4
Average (sec)	3.25	3.47	1.39	1.19	0.41	4.51	2.08	3.23
Median (sec)	0.35	0.40	0.005	0.09	0.03	0.21	0.13	0.13
Timeouts	39	44	14	9	1	84	1	41
Average c (sec)	2.84	3.10	1.19	0.65	0.25	3.62	1.79	3.24
Median c (sec)	0.35	0.40	0.005	0.08	0.03	0.19	0.11	0.13
Timeouts c	30	33	12	2	0	58	1	39
Average ¬c (sec)	11.92	11.21	5.45	12.43	3.66	22.83	8.17	2.98
Median ¬c (sec)	0.87	0.79	0.01	2.09	0.93	1.57	3.89	0.14
Timeouts ¬c	9	11	2	7	1	26	0	6

Figura 3: 'Espacio del índice (en bytes por triple), tiempo de indexación (en horas) y algunas estadísticas sobre los tiempos de consulta (en segundos). Fila "Tiempos de espera cuenta las consultas que tardan más de 60 segundos o que son rechazadas por el planificador por ser demasiado costosas. Las 2RPQ con algún nodo constante se indican con c, y sin, con ¬c'. Traducción al español de tabla de resultados [2].

Con Baseline y los k2, siendo las formas que usan el algoritmo de matrices, se puede analizar que con buenos resultados en general, se destacan para el caso en que no hay nodos

constantes en relación a tiempo y espacio, es decir, se luce en las situaciones más complejas, pues, aunque no tuvieron los mejores tiempos, tienen un espacio de uso menor que los que lo superaron. Mientras que, en el caso de consultas con nodos constantes, tuvieron peores resultados que Ring, que es otro algoritmo de tamaño similar, pero, aun así los tiempos que dieron fueron mejores que la media.

Como ya se mencionó anteriormente, una funcionalidad que carece el algoritmo de matrices de adyacencia, es que no considera la opción de tener etiquetas negadas. Entonces, se busca extender los límites del algoritmo, añadiendo la capacidad de negar etiquetas, aumentando las posibilidades en la realización de RPQs, y, al mismo tiempo, buscando mantener la eficiencia de tiempo y espacio de ejecución, que es de gran importancia.

3. Objetivos

Objetivo General

Añadir nueva capacidad de negación de etiquetas al algoritmo de RPQs de matrices adyacentes [2], editando las operaciones ya existentes para que sean capaces de manejar la nueva facultad y evaluar el desempeño de la técnica con estos cambios.

Objetivos Específicos

1. Diseñar una representación compacta para etiquetas negadas.
2. Adaptar operación de suma a la nueva representación.
3. Adaptar operación de multiplicación a la nueva representación.
4. Adaptar operación de transposición a la nueva representación.
5. Adaptar operación de transitividad cerrada a la nueva representación.
6. Adaptar operación de transitividad refleja a la nueva representación.
7. Adaptar operación de restricción de columnas y filas a la nueva representación.
8. Implementar los algoritmos y evaluarlos en grafos y consultas reales

Evaluación

Para evaluar los puntos a realizar en los objetivos, se harán múltiples pruebas, en primer lugar, las realizadas para el algoritmo sin extensión [2], para asegurar de que no falle para los casos en que no haya etiquetas negadas, y segundo se medirá con casos con etiquetas negadas y casos borde que se puedan identificar en el algoritmo.

Junto a esto, se evaluará usando los mismos parámetros que el caso sin cambios, es decir, se medirá la velocidad y tamaño de la ejecución de las consultas, analizando el tiempo y espacio que ocupa mientras se realiza la búsqueda.

4. Solución Propuesta

Para realizar este tema, considerando que se busca extender la funcionalidad, se trabajará desde el algoritmo de matrices adyacente [2], lo que significa, que primero se concentrará

en entender completamente el código ya existente, pues, no si no se comprende los cambios que se realizan se puede perder eficiencia, tanto de tiempo de ejecución del algoritmo, como tiempo de realización del tema. Siendo que el código está escrito en C, este será el lenguaje predominante a usar durante el trabajo.

Una vez entendido el código, se empezará a modificar, lo primero que habría que hacer, es ajustar la formación de las estructuras y sus reglas para que acepten la idea de una etiqueta negada.

Considerando lo anterior, la idea que se planea usar, para insertar la facultad de negación de etiquetas en el algoritmo, es 'modificar' el árbol para aparentar que una submatriz está llena de 1s. Para esto, en la representación LOUDS, se estima colocar el hijo como existente, pero sus 4 hijos vacíos, es decir, que todos sean 0s, lo cual no puede suceder actualmente debido a que, si un padre tiene todos sus hijos vacíos, entonces el padre es quien tiene valor 0 en la representación LOUDS, en este caso sería (1...0000). Esto permite que la representación siga siendo navegable y cumpla sus propiedades actuales. Entonces, la idea es recorrer el árbol de cada etiqueta negada, reemplazando los nodos vacíos por nodos padres llenos de nodos vacíos.

Una vez hecho lo anterior, habrá que modificar todas las operaciones para que consideren la existencia de estas nuevas submatrices y representaciones, tal que, se pueda ejecutar la operación correctamente, y, además, poder simplificar la operación en caso de que haya una regla matemática en ella, para mantener el tiempo de ejecución al mínimo. Por ejemplo, al aparecer estas submatrices llenas, cuando se sumen con otra matriz, el resultado es una matriz llena, ya que la suma es un 'o lógico' para cada misma coordenada. Modificaciones de este tipo, se tendrán que realizar para cada operación, algunas se esperan sean simples, pero otras, como la multiplicación, se cree serán mucho más complejas.

Una idea que se tiene para mejorar el rendimiento a la hora de usar etiquetas negadas, es que, estas etiquetas formarán nuevas submatrices con pocos 0s, entonces con las submatrices llenas, se desea intentar seguir un procedimiento similar al que se realiza para submatrices vacías y que fue implementado para la suma de matrices y su transitividad [2, 4], pero no para otras operaciones. Esto trata, cuando hay una submatriz vacía en la suma, sobre crear una copia de la otra matriz, y envés de ir nodo por nodo del árbol k^2, se realiza una copia por nivel del árbol. Traduciendo un fragmento para ejemplificar [2]: 'En esta copia se trabaja O(1) tiempo por palabra computada copiada, que en el modelo RAM transdichotómico de computación almacena O(log v) nodos de los k^2 del árbol. Por ejemplo, copiando por completo las 'a' hojas del árbol k^2 (y O(a log v) nodos) tarda O(a + log v), no O(a log v) (el segundo término aditivo corresponde a la sobrecarga de tiempo O(1) en cada nivel).'

Una vez logrado esto, se planea ejecutar el mismo plan de prueba, con casos de etiquetas negadas, que la versión actual, si es posible, en caso de que los recursos que se tengan no sean suficientes, se realizaría a una menor escala. En caso de que los resultados de tiempo y espacio empeoren en gran medida, habrá que revisar los cambios hechos y buscar nuevas ideas para implementar más eficientemente la nueva capacidad, pues, ya fue mencionado que una prioridad en el trabajo actual del algoritmo, es su buena relación tiempo-espacio.

5. Plan de Trabajo (Preliminar)

1. Estudiar código del algoritmo ya existente.
2. Ajustar algoritmo para ser capaz de representar etiquetas negadas.
3. Adaptar las operaciones de matrices para que acepten la nueva representación
4. Evaluar el algoritmo con los cambios hechos.
5. Analizar resultados.
6. Según lo analizado en el punto anterior, repetir proceso para tareas que se hayan realizado ineficientemente.
7. Escribir memoria con todo el trabajo realizado.

Referencias

[1] Arroyuelo, Diego, Adrián Gómez-Brandón, Aidan Hogan, Gonzalo Navarro y Javiel Rojas-Ledesma: *Optimizing RPQs over a compact graph representation.* The VLDB Journal, 33:349–374, 2024. https://doi.org/10.1007/s00778-023-00811-2.

[2] Arroyuelo, Diego, Adrián Gómez-Brandón y Gonzalo Navarro: *Evaluating Regular Path Queries on Compressed Adjacency Matrices*, 2024. https://arxiv.org/abs/2307.14930.

[3] Hogan, Aidan, Cristian Riveros, Carlos Rojas y Adrián Soto: *A Worst-Case Optimal Join Algorithm for SPARQL.* En Ghidini, Chiara, Olaf Hartig, Maria Maleshkova, Vojtěch Svátek, Isabel Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois y Fabien Gandon (editores): *The Semantic Web – ISWC 2019*, páginas 258–275, Cham, 2019. Springer International Publishing, ISBN 978-3-030-30793-6.

[4] Quijada-Fuentes, Carlos, Miguel R. Penabad, Susana Ladra y Gilberto Gutiérrez: *Set operations over compressed binary relations.* Information Systems, 80:76–90, 2019, ISSN 0306-4379. https://www.sciencedirect.com/science/article/pii/S0306437917302612.

[5] Sakr, Sherif, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang Yun Wu, Nikolay Yakovets, Da Yan y Eiko Yoneki: *The future is big graphs: a community view on graph processing systems.* Communications of the ACM, 64(9):62–71, Agosto 2021, ISSN 1557-7317. http://dx.doi.org/10.1145/3434642.

[6] Schoor, Amir: *Fast algorithm for sparse matrix multiplication.* Information Processing Letters, 15(2):87–89, 1982, ISSN 0020-0190. https://www.sciencedirect.com/science/article/pii/0020019082901144.