Evaluating Navigational Queries Over Graph Databases

IIC3432 - Tópicos Avanzados en Bases de Datos

Gabriel Aguirre - Benjamín Farías - Juan Hernández - Benjamín Lepe

Introducción

RDF es el *framework* estándar para representar la información web semántica, el que posee una estructura de grafo subyacente:

```
inTods:StonebrakerWKH76   dc:creator   M.Stonebraker
inSigmod:PavloPRADMS09    dc:creator   M.Stonebraker
```

Su lenguaje oficial de consultas, **SPARQL**, incluye la capacidad de realizar consultas usando *regular expressions*, mediante *property paths*. Este tipo de consultas entregan una forma intuitiva de encontrar patrones de interés entre grandes volúmenes de datos de la web:

```
M.Stonebraker  (^dc:creator/dc:creator)*  ?x
```

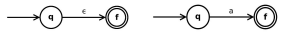
Autómata

1. **Property Paths (Regex)**

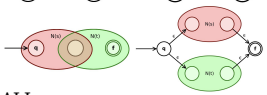
$$
\begin{aligned}
&[\![a]\!]_G = \{(s,o) \mid (s,a,o) \in G\},\\
&[\![a^-]\!]_G = \{(s,o) \mid (o,s) \in [\![a]\!]_G\},\\
&[\![e_1 \cdot e_2]\!]_G = [\![e_1]\!]_G \circ [\![e_2]\!]_G,\\
&[\![e_1 + e_2]\!]_G = [\![e_1]\!]_G \cup [\![e_2]\!]_G,\\
&[\![e^*]\!]_G = \bigcup_{i \geq 1} [\![e^i]\!]_G \cup \{(a,a) \mid a \text{ is a term in } G\},\\
&[\![e?]\!]_G = [\![e]\!]_G \cup \{(a,a) \mid a \text{ is a term in } G\}.
\end{aligned}
$$

2. **Regex a Autómata**

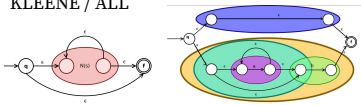
ATOM



AND / OR



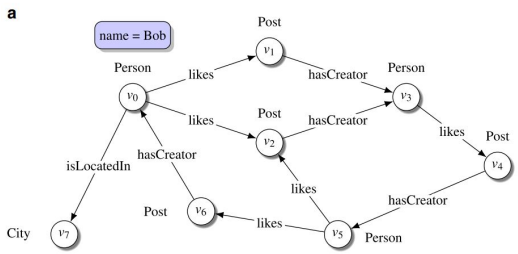
KLEENE / ALL



Ejecución, Algoritmos y Semántica

La idea principal es que a partir de una *property path expression,* debemos obtener un **autómata equivalente**, el que nos servirá para evaluar la expresión en un ***web graph.***

El algoritmo consiste en ir explorando **estados** (v, q), en donde v corresponde a un nodo del grafo y q es un estado del autómata. De esta forma, se recorre el producto cruz entre el grafo y autómata, a partir de un par inicial (v_0, q_0), y continuando iterativamente con los nodos del vecindario que cumplan con **dos condiciones:** que exista una arista e directa desde v_0 y que haya una función de transición desde q_0 leyendo e. Siguiendo este proceso **iterativo de búsqueda**, se explora un conjunto de estados (v_i, q_i), agregando al set de **soluciones** aquellos caminos que lleven a un estado cuyo q_i pertenece a los **estados finales** del autómata.



Data graph.

Results of RPQ evaluation.

La exploración del **espacio de búsqueda** al ejecutar una consulta puede basarse en cualquier **algoritmo** conocido que actúe sobre grafos. Por ejemplo:

BFS: Explora a lo ancho, entrega caminos más cortos si la medida de distancia es la cantidad de aristas.

DFS: Explora en profundidad, puede ser más rápido al fijar un límite de resultados, pero no es óptimo.

Dijkstra: Permite encontrar caminos más cortos cuando la métrica de distancia es un valor numérico real.

A*: Encuentra caminos óptimos de forma eficiente (mediante heurísticas), pero usa demasiada memoria.

Este tipo de consultas soporta distintas **semánticas**, relativas al objetivo de la consulta en específico. Dado un nodo **inicial** *NI* y un nodo **final** *NF*, se distinguen 3 casos:

(NI=X, NF=Y): Chequear si existe un camino entre los nodos X e Y que cumpla con la consulta.

(NI=X, NF=?) o (NI=?, NF=Y): Enumerar todos los caminos que salen de X (o llegan a Y) y que cumplan con la consulta.

(NI=?, NF=?): Enumerar todos los caminos que cumplan con la consulta, para cualquier par de nodos inicial y final.

Experimentos

Estudiaremos cómo se comportan estas consultas en el contexto del motor *MillenniumDB*, comparando su rendimiento con otros motores típicos de bases de datos de grafos. El enfoque está en la evaluación de *regular path queries* (***RPQs***), mediante los algoritmos de **enumeración BFS y DFS**.

Resultados

Resultados y Rendimiento (en segundos):

Engine	Supported	Error	Timeouts	Average	Median
MillDB BFS	1683	0	0	1.1	0.095
MillDB DFS	1683	0	0	1.1	0.072
Blazegraph	1683	2	44	27.6	0.396
Jena	1683	14	46	22.8	0.207
Virtuoso	1683	55	4	5.8	0.325
Neo4J	1622	0	42	23.3	0.328

Se observa clara superioridad de *MillenniumDB*. Entre ambos algoritmos, **DFS** es ligeramente superior, pero esto se debe a que no se imprimen los caminos en estos experimentos, de otra forma **BFS** suele ser la mejor opción (y además asegura caminos más cortos).

Conclusiones

El motor de *MillenniumDB* presenta resultados favorables en bases de datos aplicables a problemas reales (***Wikidata***), por lo que tiene valor tanto en **investigación** como en la **práctica**. El trabajo futuro corresponde a mejorar las capacidades de este motor, extendiendo sus funcionalidades y haciéndolo más flexible.