Algoritmos para casamento de Grafos

Hugo Paulino Bonfim Takiuchi

UFPR
2018

Artigo base

- An In-depth Comparison of Subgraph Isomorphism. Algorithms in Graph Databases
- Jinsoo Lee
- Wook-Shin Han
- Romans Kasperovics
- Jeong-Hoon Lee

Sumário

- Introdução
- Algoritmo Genérico
- Algoritmos para casamento de grafos
  - Ullman
  - VF2
  - QuickSI
  - GADDI
  - GraphQL
  - SPath
- Experimentos
- Conclusão

Introdução

- Grafos para modelagem de problemas
  - Redes sociais
  - Moleculas
  - RDF
- Problema importante: Busca de grafos isomorfo em um grafo base

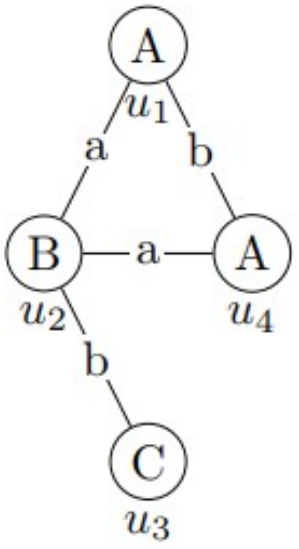
Introdução

- Grafos isomorfos: busca de um padrão de grafo em um grafo base
- Definição:
  - Grafo não-direcionado e rotulado (V, E, L)
  - Grafo busca (q) e Grafo dados (g)
  - Vértice busca (u) e vértice dados (v)
  - Função injetora M: $V \rightarrow V'$

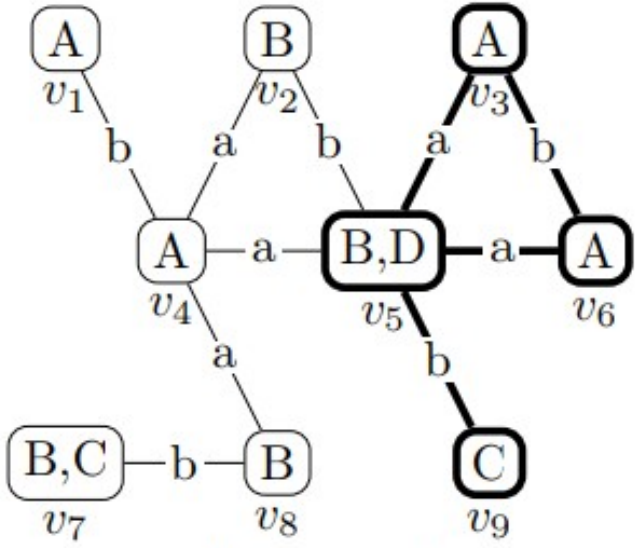
Introdução

- Função injetora M: $V \rightarrow V'$
  - Vértices correspondentes
    - $\forall u \in V$, $L(u) \subseteq L(M(u))$
  - Arestas correspondentes
    - $\forall (ui, uj) \in E$, $(M(ui), M(uj)) \in E'$, então $L(ui, uj) = L(M(ui), M(uj))$.

Introdução



query graph q.



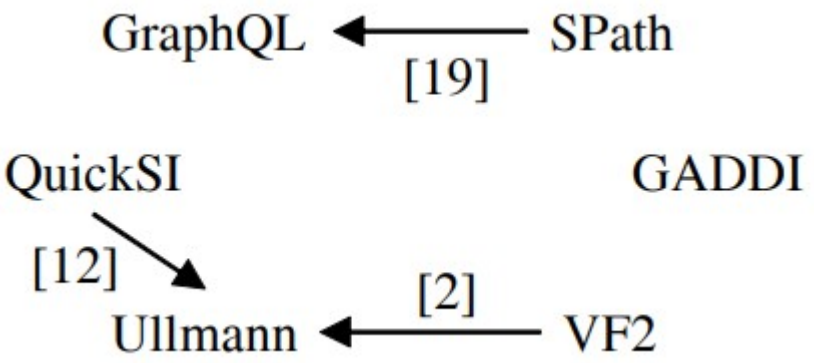
data graph g.

$$M_1 = \{(u_1, v_3), (u_2, v_5), (u_3, v_9), (u_4, v_6)\}$$
$$M_2 = \{(u_1, v_6), (u_2, v_5), (u_3, v_9), (u_4, v_3)\}$$

resulting subgraph isomorphisms.

Introdução

- Algoritmos Para busca de grafos isomorfos
  - Ullman, VF2, QuickSI, GraphQL, GADDI e Spath



Introdução

- Subgrafo induzido
- Solução parcial (subgrafo isomorfo)
- Conjunto de adjacência
- Vizinhança

Algoritmo Generico

- Backtracking
- Incrementa solução parcial ou abandona o resultado
- Ideia geral: vértice inicial → grafo de busca

Algoritmo Generico

Algorithm 1 GENERICQUERYPROC

Input: query graph q
Input: data graph g
Output: all subgraph isomorphisms of q in g

```
 1: M := ∅;
 2: for each u ∈ V(q) do
 3:    C(u) := FILTERCANDIDATES (q, g, u, ...);
       [[ ∀v ∈ C(u)((v ∈ V(g)) ∧ (L(u) ⊆ L(v))) ]]
 4:    if C(u) = ∅ then
 5:       return;
 6:    end if
 7: end for
 8: SUBGRAPHSEARCH (q, g, M, ...);
```

Subroutine SUBGRAPHSEARCH $(q, g, M, \ldots)$

```
 1: if |M| = |V(q)| then
 2:    report M;
 3: else
 4:    u := NEXTQUERYVERTEX (...);
       [[ u ∈ V(q) ∧ ∀(u', v) ∈ M(u' ≠ u) ]]
 5:    C_R := REFINECANDIDATES (M, u, C(u), ...);
       [[ C_R ⊆ C(u) ]]
 6:    for each v ∈ C_R such that v is not yet matched do
 7:       if ISJOINABLE (q, g, M, u, v, ...) then
 8:          [[ ∀(u', v') ∈ M((u, u') ∈ E(q) ⟹
                    (v, v') ∈ E(g) ∧ L(u, u') = L(v, v')) ]]
 9:          UPDATESTATE (M, u, v, ...);
             [[ (u, v) ∈ M ]]
10:          SUBGRAPHSEARCH (q, g, M, ...);
11:          RESTORESTATE (M, u, v, ...);
             [[ (u, v) ∉ M ]]
12:       end if
13:    end for
14: end if
```

Algoritmo Generico

Input: query graph q
Input: data graph g
Output: all subgraph isomorphisms of q in g

1: $M := \emptyset$;
2: **for each** $u \in \mathrm{V}(q)$ **do**
3: $C(u) :=$ FILTERCANDIDATES $(q, g, u, \ldots)$;
[[$\forall v \in C(u)((v \in \mathrm{V}(g)) \wedge (\mathrm{L}(u) \subseteq \mathrm{L}(v)))$]]
4: **if** $C(u) = \emptyset$ **then**
5: **return**;
6: **end if**
7: **end for**
8: SUBGRAPHSEARCH $(q, g, M, \ldots)$;

Subroutine SUBGRAPHSEARCH $(q, g, M, \ldots)$

Algoritmo Generico

Subroutine SUBGRAPHSEARCH $(q, g, M, \ldots)$

1: **if** $|M| = |V(q)|$ **then**
2: **report** M;
3: **else**
4: $u :=$ NEXTQUERYVERTEX $(\ldots)$;
$[[\ u \in \mathrm{V}(q) \wedge \forall (u', v) \in M (u' \neq u)\ ]]$
5: $C_R :=$ REFINECANDIDATES $(M, u, C(u), \ldots)$;
$[[\ C_R \subseteq C(u)\ ]]$
6: **for each** $v \in C_R$ such that v is not yet matched **do**
7: **if** ISJOINABLE $(q, g, M, u, v, \ldots)$ **then**
8: $[[\ \forall (u', v') \in M ((u, u') \in \mathrm{E}(q) \implies$
$(v, v') \in \mathrm{E}(g) \wedge \mathrm{L}(u, u') = \mathrm{L}(v, v'))\ ]]$
9: UPDATESTATE $(M, u, v, \ldots)$;
$[[\ (u, v) \in M\ ]]$
10: SUBGRAPHSEARCH $(q, g, M, \ldots)$;
11: RESTORESTATE $(M, u, v, \ldots)$;
$[[\ (u, v) \notin M\ ]]$
12: **end if**
13: **end for**
14: **end if**

Ullman

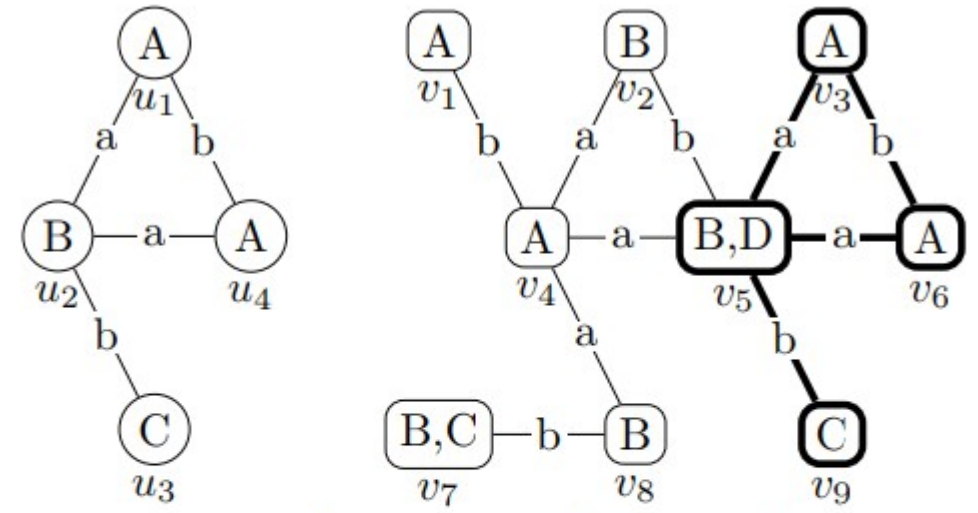
- Filter Predicates: Conjunto de vértices de G que casam com *u*
- Next Query Vertex: ordem FIFO
- Refine Candidates: Todos os vértices de C(*v*) com grau menor igual a *u*
- Is Joinable: procura por todos os vértices adjacentes a *u* e busca no grafo de dados

VF2

- Listas para armazenamento
- Next Query Vertex: escolhe vértice ligados com vértices já comparados
- Refine Candidates: 3 etapas
  - Retira de C(v) os vértices não ligados a vértices já buscados
  - $|Cq \cap adj(u)| > |Cg \cap adj(v)|$
  - $|adj(u) \setminus Cq \setminus Mq| > |adj(v) \setminus Cg \setminus Mg|$

VF2

- M = {(u1, v4)} e u2
- Mq = {u1}
- Mg = {v4}
- C(u2) = {v2, v5, v7, v8}
- Cq = {u2, u4}
- Cg = {v1, v2, v5, v8}



query graph q.

data graph g.

$$M_1 = \{(u_1, v_3), (u_2, v_5), (u_3, v_9), (u_4, v_6)\}$$
$$M_2 = \{(u_1, v_6), (u_2, v_5), (u_3, v_9), (u_4, v_3)\}$$

resulting subgraph isomorphisms.

VF2

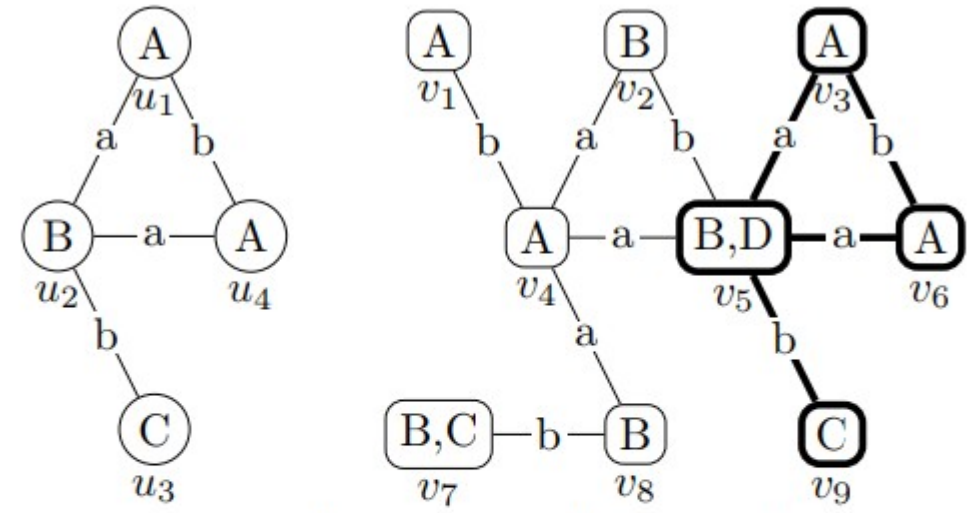
- Etapa 1:
  - Retira v7
- Etapa 2:

  adj(u2) ∩ Cq = {u4}

  adj(v8) ∩ Cg = {}
  - Retira v8
- Etapa 3:

  adj(u2) \ Cq \ Mq = {u3}

  adj(v2) \ Cg \ Mg = {}
  - Retira v2



$$M_1 = \{(u_1, v_3), (u_2, v_5), (u_3, v_9), (u_4, v_6)\}$$
$$M_2 = \{(u_1, v_6), (u_2, v_5), (u_3, v_9), (u_4, v_3)\}$$

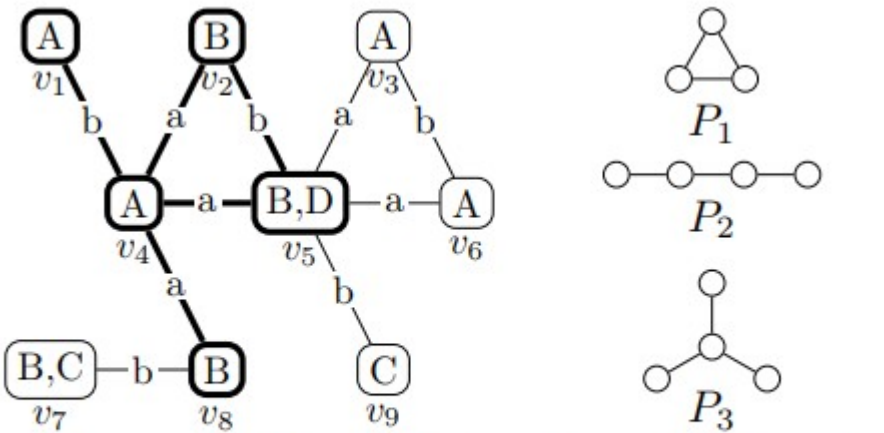
resulting subgraph isomorphisms.

QuickSI

- Utiliza listas e árvores B+ para armazenamento
- Next Query Vertex:
  - escolhe com base no menos acessado
  - Árvore de mínimo
- Refine Candidates:
  - Pula o primeiro vértice
  - Verifica se o pai do vértice de busca tem correspondente no grafo de dados
- Is Joinable:
  - Vértices adjacentes e já comparados

GADDI

- Listas para armazenamento
- Distância NDS (ΔNDS(v1, v2, P)):
  - Quantidade de casamentos do subgrafo P em Nk(v1)∩Nk(v2)



$$\Delta_{\mathrm{NDS}}(v_1, v_4, P_1) = 3,\ \Delta_{\mathrm{NDS}}(v_1, v_4, P_2) = 8,$$
$$\Delta_{\mathrm{NDS}}(v_1, v_4, P_3) = 24$$

GADDI

- Utilização de ΔNDS(v1, v2, P):
  - Pegar 10 subgrafos induzidos mais frequentes
  - Criar tabela
    - Linhas = subgrafos induzidos
    - Colunas = subgrafos P
  - Selecionas 3 colunas com maior variação nos valores

GADDI

- Next Query Vertex:
  - Utiliza uma busca em profundidade
- Refine Candidates:
  - Para todos u’ $\in$ Nk(u), se não existir um v’ $\in$ Nk(v) que satisfaz:
    - L(u’) $\subseteq$ L(v’)
    - ΔNDS(u’, u, Pi) ≤ ΔNDS(v’, v, Pi)
    - Menor caminho entre u e u’ >= menor caminho entre v e v’
  - Retira v de C(u)

GADDI

- Update State, Restore State:
  - mesma condições que Refine Candidates, porém invertidas (v’ para u’)
  - Retira/adiciona v’

GraphQL

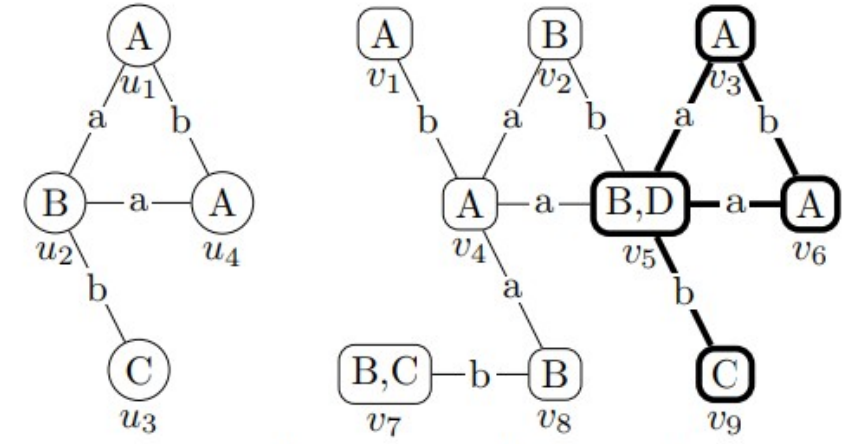
- Listas para armazenamento
- Filter Candidates: 2 etapas
  - Assinatura de vizinhança GraphQL
    - **sigGraphQL(u)** não contido em **sigGraphQL(v)**
  - Árvores de busca
    - Tu não contido em Tv para profundidade d
- Next Query Vertex: Estratégia gulosa
  - Vértice com menor C(v) e ligados a vértices já comparados

SPath

- Listas para armazenamento
- Faz busca por caminhos ao invés de vértices
- Filter Candidates: Assinatura de vizinhança Spath
  - sigSPath(u) = conjunto de triplas (d, l, c) sobre Nk0 (u)
  - sigSPath(v7) = (1, B, 1), (2, A, 1)
  - sigSPath(v3) = (1,B, 1), (2, A, 2)
  - Se não satisfaz sigSPath(u) <= sigSPath(u) para todos os rótulos e k < k0 retira.

SPath

- sigSPath(v7) = (1, B, 1), (2, A, 1)
- sigSPath(v3) = (1,B, 1), (2, A, 2)



$$M_1 = \{(u_1, v_3), (u_2, v_5), (u_3, v_9), (u_4, v_6)\}$$
$$M_2 = \{(u_1, v_6), (u_2, v_5), (u_3, v_9), (u_4, v_3)\}$$

resulting subgraph isomorphisms.

SPath

Subroutine SUBGRAPHSEARCH (q, g, M, k_0)

```
1: if |M| = |V(q)| then
2:    report M;
3: else
4:    p_q := NEXTQUERYPATH (q, g, k_0);
5:    P := GETCANDIDATEPATHS (p_q, M, C);
6:    for each p_g ∈ P do
7:       if ISJOINABLE (p_q, p_g, M) then
8:          UPDATESTATE (p_q, p_g, M);
9:          SUBGRAPHSEARCH (q, g, M, k_0);
10:         RESTORESTATE (p_q, p_g, M);
11:      end if
12:   end for
13: end if
```

Experimentos

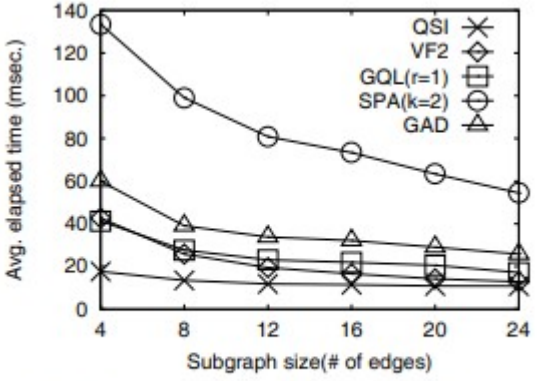
- 4 Banco de dados diferentes
- Todos os algoritmos foram implementados em C++
- Características observadas:
  - Espaço de armazenamento
  - Tempo de execução
  - Chamadas recursivas

Experimentos

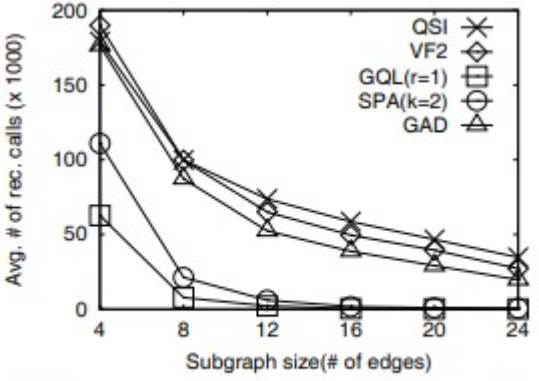
Dataset	AIDS	NASA	Yeast	Human
# of graphs	10000	36790	1	1
# of vertices	2∼214	2∼889	3112	4675
# of edges	1∼217	1∼888	12519	86282
Avg. degree	1.95	1	8.05	36.82
Max. degree	11	245	168	771
# of distinct v. labels	51	117302	184	90
# of distinct e. labels	4	0	0	0
Avg. # of labels per vertex	1	1	7.55	4.63

AIDS Dataset

Alg.	Size(MB)	# of total I/Os	Time(msec.)
VF2	9.25	1774	418
QuickSI	9.28	1780	671
GraphQL	12.59	2202	684
SPath($k_0 = 2$)	21.40	3330	1420
GADDI	21.47	3339	9064



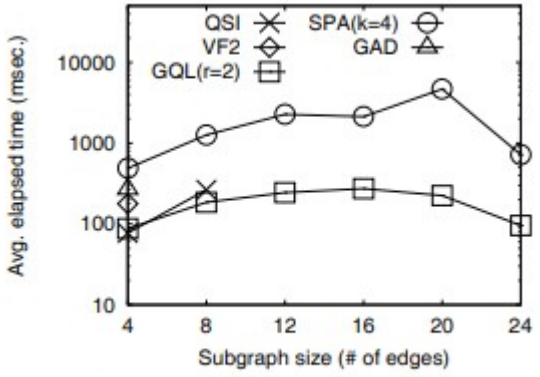
(a) average elapsed time.



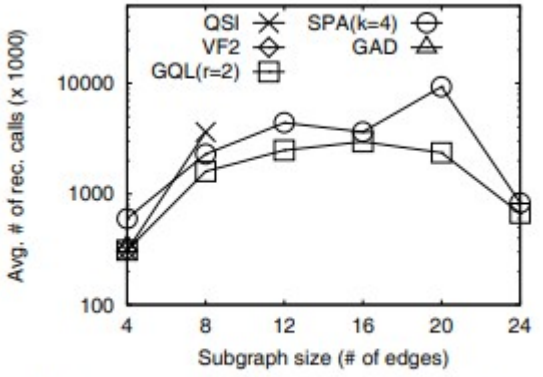
(b) avg. # of recursive calls.

NASA Dataset

Alg.	Size(MB)	# of total I/Os	Time(msec.)
VF2	44.04	8718	1697
QuickSI	58.99	10634	3769
GraphQL	57.65	10461	1950
SPath(k_0=4)	257.68	36089	21737
GADDI	95.33	15288	32167



(a) average elapsed time.

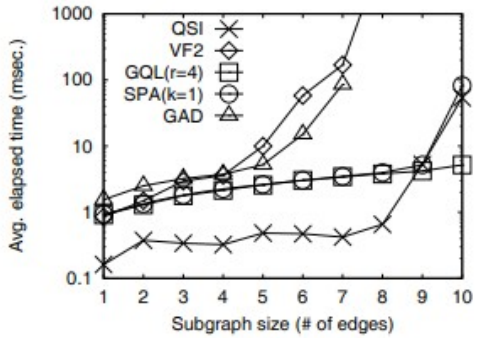


(b) avg. # of recursive calls.

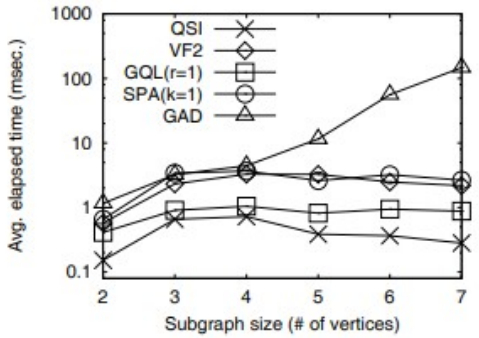
Yeast Dataset

Alg.	Size(MB)	# of total I/Os	Time(msec.)
VF2	0.41	89	46
QuickSI	1.07	175	378
GraphQL	2.03	296	284
SPath(k_0=1)	3.73	514	437
GADDI	0.91	152	25272

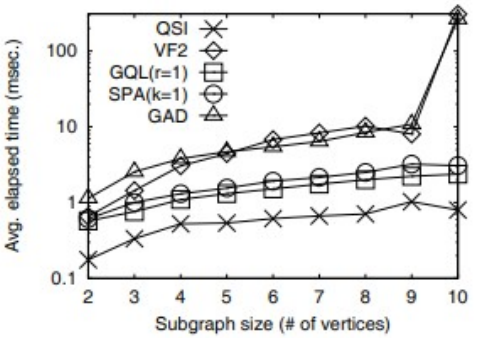
Yeast Dataset



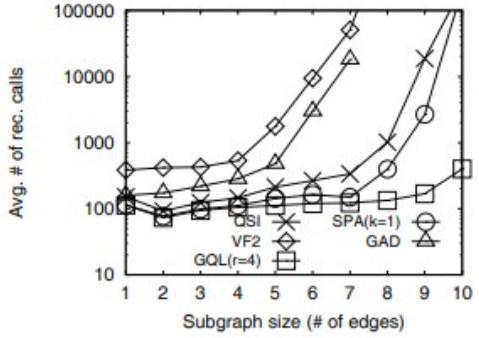
(a) subgraph queries.



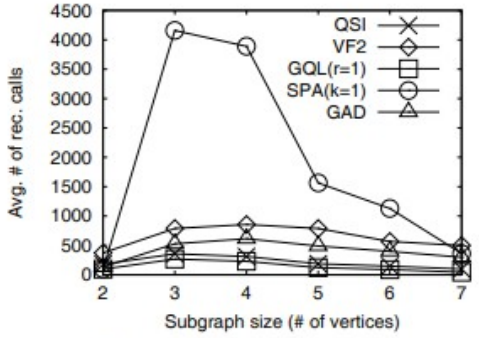
(b) clique queries (Y-axis in log scale).



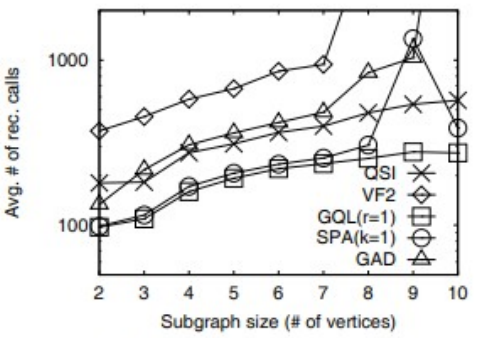
(c) path queries.



(a) subgraph queries.



(b) clique queries.

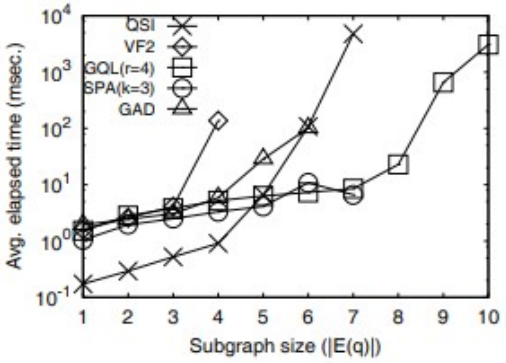


(c) path queries.

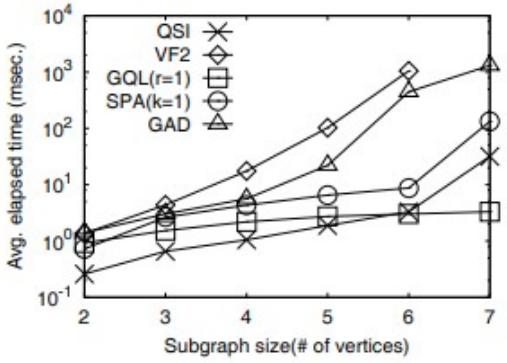
Human Dataset

Alg.	Size(MB)	# of total I/Os	Time(msec.)
VF2	1.55	349	93
QuickSI	1.72	373	964
GraphQL	8.13	1193	774
SPath(k_0=1)	11.28	1596	1669
SPath(k_0=3)	17.09	2340	3323
GADDI	4.88	775	624265

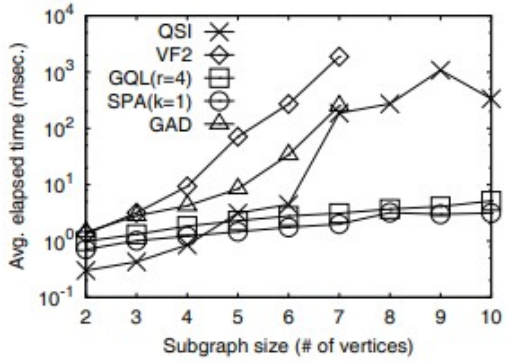
Human Dataset



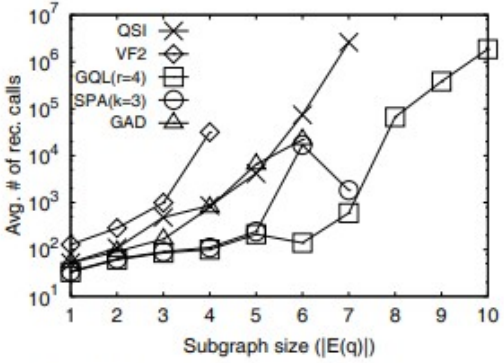
(a) subgraph queries.



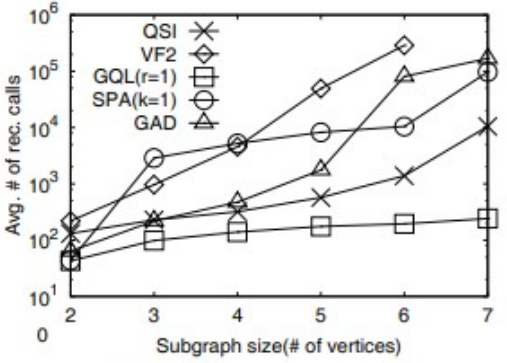
(b) clique queries.



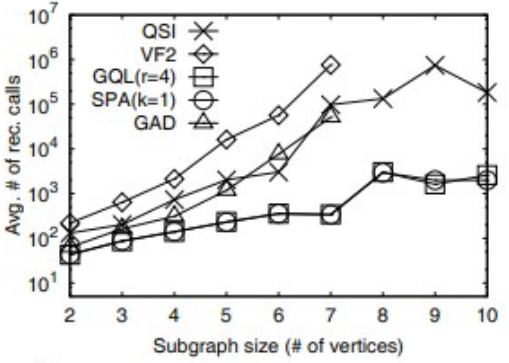
(c) path queries.



0 (a) subgraph queries.



0 (b) clique queries.



0 (c) path queries.

Conclusão

- QuickSI apresentou melhor rendimento para algumas bases de dados (AIDS e YEAST)
- QuickSI, VF2, GADDI não apresentaram tempo de execução adequado para NASA
- GADDI apresentou o pior rendimento do geral
- GraphQL apresentou tempo de execução adequado para todas as buscas
- GraphQL foi melhor que Spath
- Todos algoritmos apresentaram dificuldade em escolher próximo passo