On the Evaluation of Pattern Match Queries in Large Graph Databases

Thesis defense by Bin Guo supervised by Pro. Yandjun Chen

Applied Computer Science

March 12, 2018

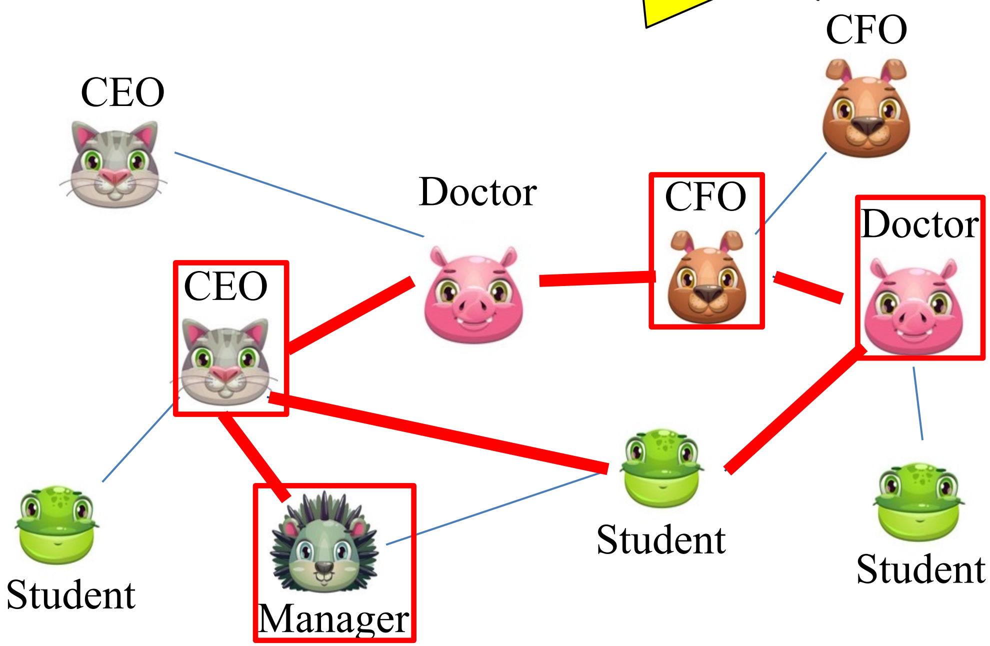
OUT LINE

1. What is Graph Pattern Matching Queries?
2. General Framework of Our Method.
3. **Δ-Transitive Closure Construction.**
4. **Domain Filtering.**
5. **Relation Filtering**
6. Related Work.
7. Experiments.
8. Future Work.
9. Reference.

1. What Is Graph Pattern Match Queries?

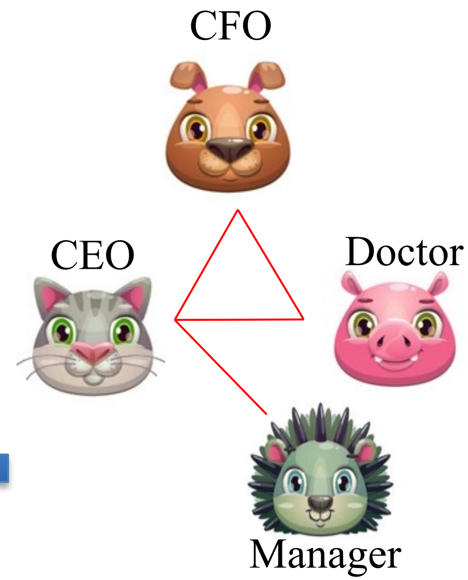
- Graph is an important data structure for many applications like social network.
- For example, “Facebook” is one of the largest social network which has 2.2 billion monthly active users.

Pattern Matching Queries



A example of social network Graph

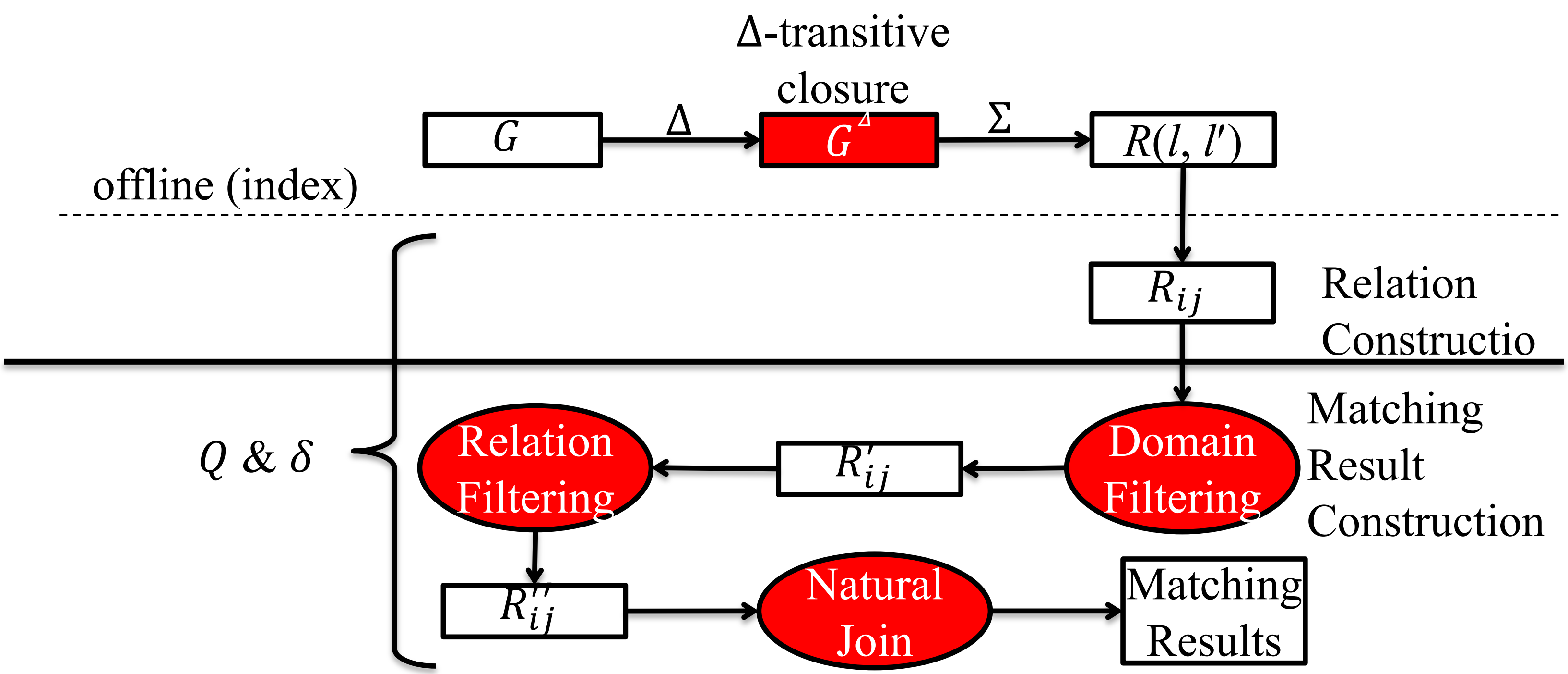
shortest-path distance $\delta \leq 2$



Query

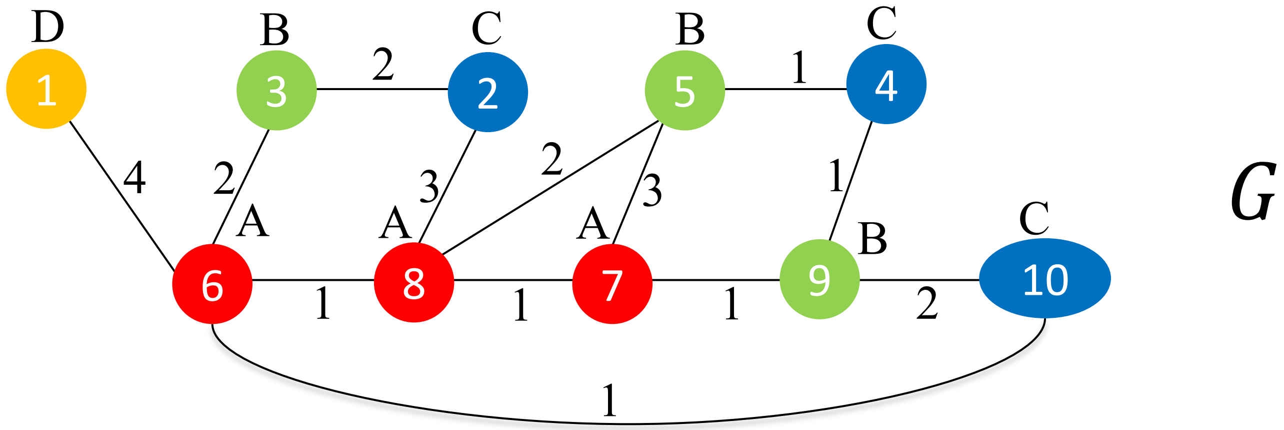
A successful CEO's circle of friends.

2. General Framework of Our Method

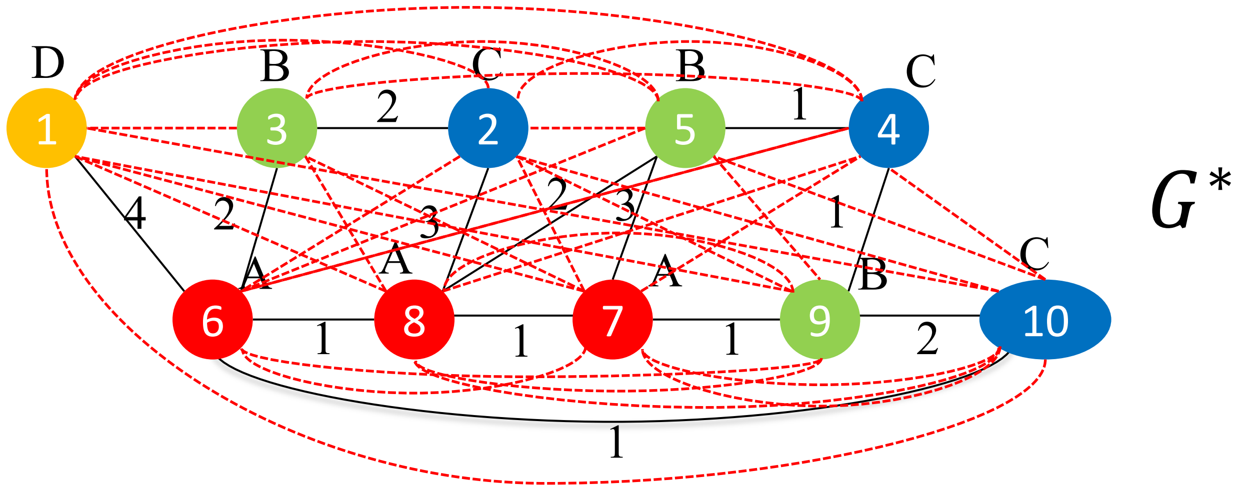


- δ is the shortest-path distance limitation and $\Delta \geq \delta$.
- R_{ij} : Original relations; R'_{ij}: filtered by Domain Filtering; R''_{ij}: filtered by Relation Filtering.

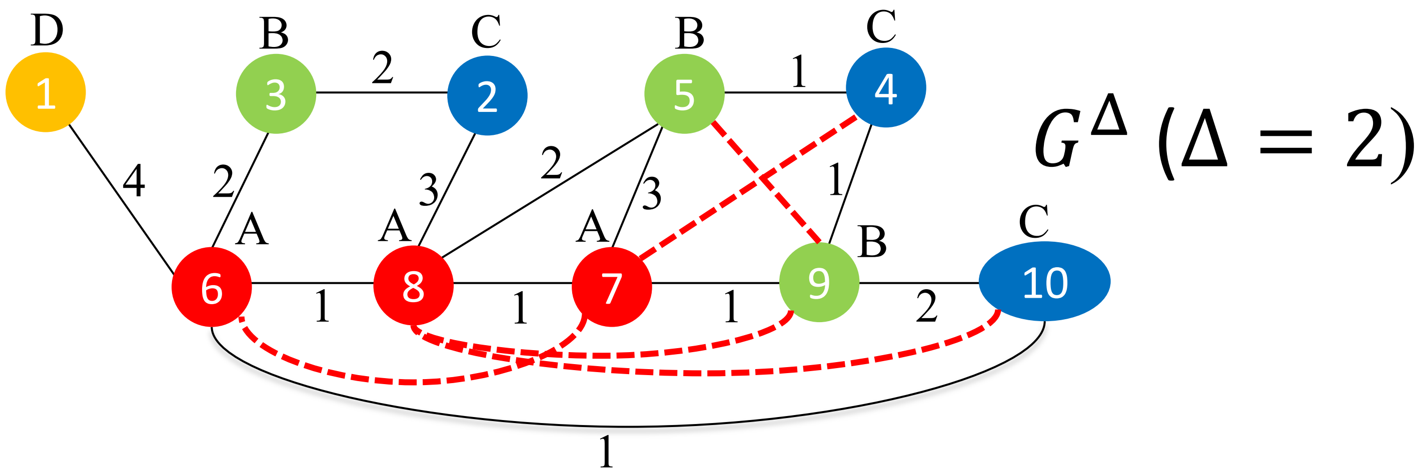
3. Δ-Transitive Closure Construction



An example of weighted labeled data graph G with 10 vertices labeled by A, B, C, D.



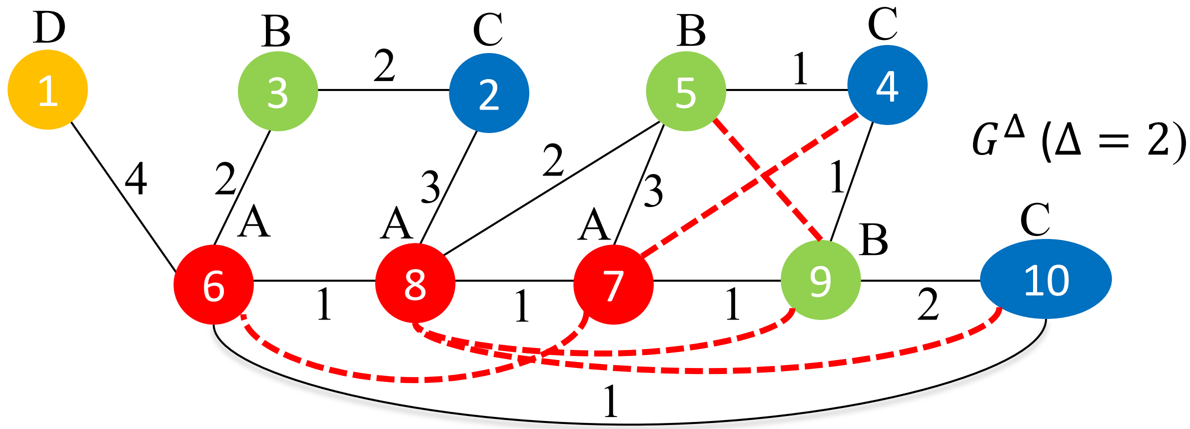
- **Transitive Closure G^*: each pair of vertices (u, u') are connected by an edge if they are reachable, which is weighted by $Dist_{sp}(u, u')$.**
- For example, for the pair of vertices (u_1, u_{10}), it has no edge in G. But in G^*, the new edge (u_1, u_{10}) is added weighted by $Dist_{sp}(u, u') = 5$.

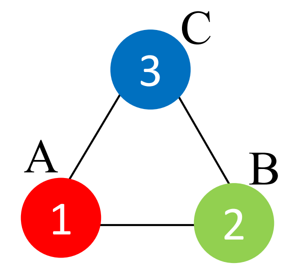


- **Δ-Transitive Closure G^{Δ}: each pair of vertices $(\boldsymbol{u}, \boldsymbol{u}')$ are connected by an edge if $\boldsymbol{Dist_{sp}}(\boldsymbol{u}, \boldsymbol{u}') \leq \boldsymbol{\Delta}$, which is weighted by $\boldsymbol{Dist_{sp}}(\boldsymbol{u}, \boldsymbol{u}')$.**
- For example, for the pair of vertices (u_1, u_{10}), it has no edge in G. In G^{Δ}, the edge (u_1, u_{10}) is not added since $Dist_{sp}(u, u') = 5 > \Delta$.
- For the pair of vertices (u_8, u_9), it has no edge in G. But in G^{Δ}, the new edge (u_8, u_9) is added since $Dist_{sp}(u, u') = 2 \leq \Delta$.

- Obviously, our **Δ-Transitive Closure G^{Δ}** require less running time and space than the traditional **Transitive Closure G^{*}**.
- G^{Δ} can be generated offline as index and spend no time during queries for relation construction.

4. Domain Filtering





Query Q with shortest-path distance $\delta = 2$

$(v_1, v_2) \in E(Q)$
R_{12}

A	B
6	3
8	5
8	9
7	9

$\bowtie$

$(v_1, v_2) \in E(Q)$
R_{23}

B	C
3	2
5	4
9	10
9	4

$\bowtie$

$(v_1, v_2) \in E(Q)$
R_{31}

C	A
10	6
10	8
4	7

Naïve Natural Join of Relations

R_{12}

A	B
6	3
8	5
8	9
7	9

$\bowtie$

R_{23}

B	C
3	2
5	4
9	10
9	4

$=$

R_{123}

A	B	C
6	3	2
8	5	4
8	9	10
8	9	4
7	9	10
7	9	4

$\bowtie$

R_{31}

C	A
10	6
10	8
4	7

$=$

R_{123}

A	B	C
8	9	10
7	9	4

2 matching results

- It need $|R_{12}| \cdot |R_{23}| \cdot |R_{31}| = 4 \times 3 \times 3 = 36$ times computation.
- Theoretically, the Natural Joins are NP-hard problem with running time $O(\prod_{ij} |R_{ij}|)$.

We can do better!

R_{12}

A	B
6	3
8	5
8	9
7	9

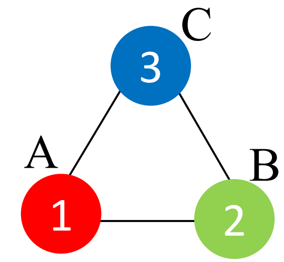
R_{23}

B	C
3	2
5	10
9	4

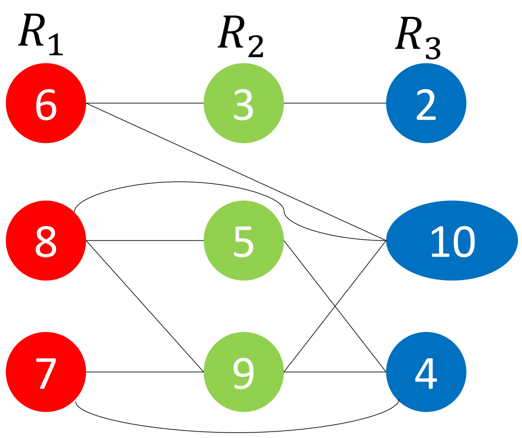
R_{31}

C	A
10	6
10	8
4	7

Relations



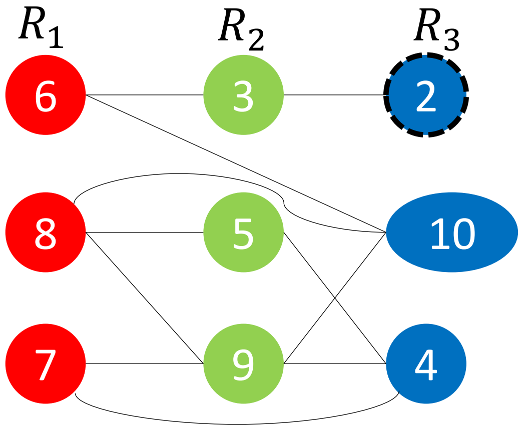
Query Q



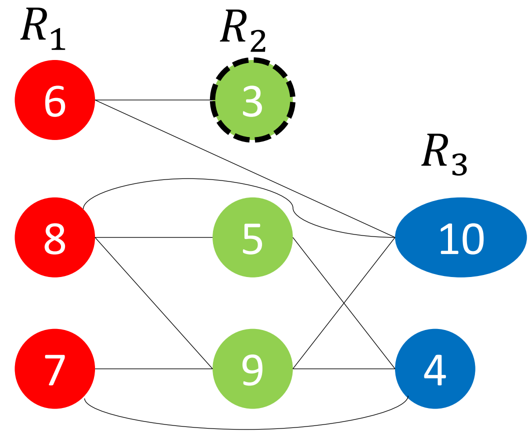
Relation Graph.

- ***DomainFiltering***: **Each $u \in R_i$ must appear in all relations R_{ij} where $(v_i, v_j) \in E(Q)$ $(j = 1, \dots, n)$. If not, it should be removed.**

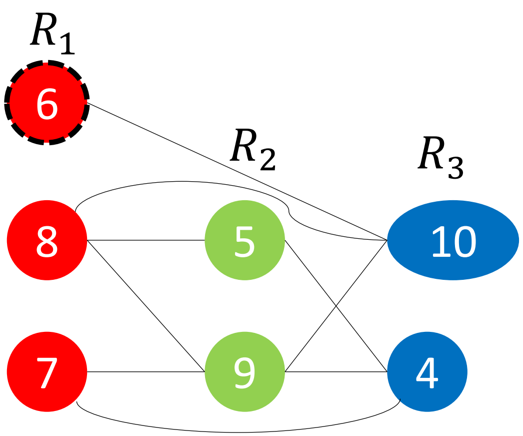
- For example, $u_6 \in R_1$ appear in both R_{12} and R_{31}, so do nothing.
- But u_2 not appear in R_{31}, so it should be removed.



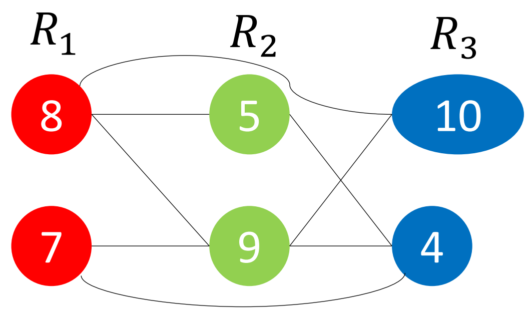
(1) u_2 not appear in R_{31}, so u_2 is removed.



(2) Removing u_2 causes u_3 not appear in R_{23}, so u_3 is removed.



(3) Removing u_3 causes u_6 not appear in R_{12}, so u_6 is removed.



(4) All $u \in R_i$ appear in R_{ij}.

The relations participating natural joins are reduced.

Algorithm *DomainFiltering*(Q, δ, all R_{ij})

Input: query Q, δ and all relations R_{ij}.
Output: reduced relations R'_{ij}.

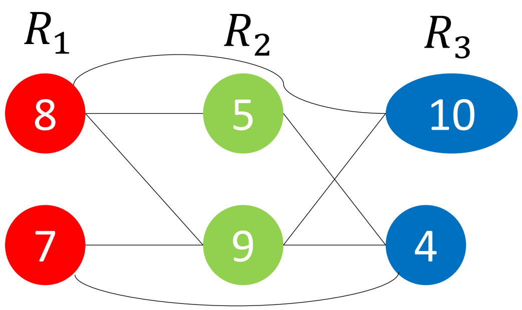
```
//phase 1: construct the support lists, counters and STACK.
STACK := Φ;
for each (v_i, v_j) ∈ E(Q) do
    get R_i, R_j, R_ij
    for each (u, u') ∈ R_ij do
        u.S.append(⟨j, u'⟩), u.C[j]++;
        u'.S.append(⟨i, u⟩), u'.C[i]++;
    for each u ∈ R_i do
        if u.C[j] == 0 then
            R_i.remove(u), STACK.push(⟨i, u⟩);
    for each u' ∈ R_j do
        if u'.C[i] == 0 then
            R_j.remove(u'), STACK.push(⟨j, u'⟩);
//phase 2: process the STACK.
while STACK ≠ Φ do
    ⟨i, u⟩ = STACK.pop();
    for each ⟨j, u'⟩ ∈ u.S do
        if (--u'.C[i]) == 0 then
            R_j.remove(u'), STACK.push(⟨j, u'⟩);
return R'_ij = R_ij − {(u, u'): u ∉ R_i or u' ∉ R_j};
```

m is the number of edges in query Q.
$D = \max\{|R_i|\}$.

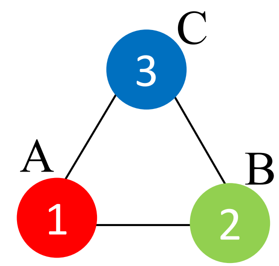
- Both running time and space are bounded by $O(mD^2)$.
- It is much more efficient than Natural Joins.
- It is valuable to do *DomainFiltering* before Natural Joins.

We can further do better!

5. Relation Filtering



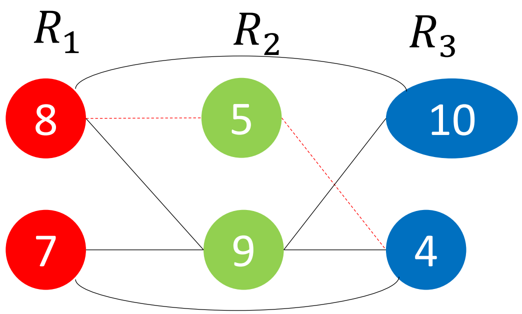
The relation graph after *DomainFiltering*



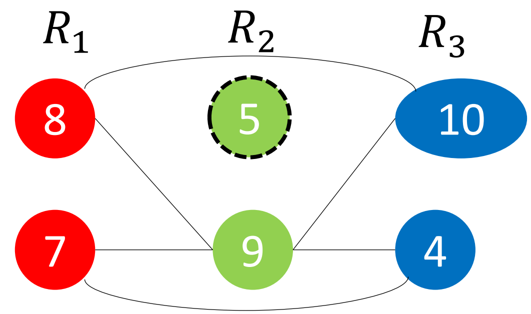
query Q has ***triangle edges***.

- ***RelationFiltering*: For each $(u, u') \in R_{ij}$, there must be a $u'' \in R_k$ such that $(u, u'') \in R_{ik}$ and $(u', u'') \in R_{jk}$. If not, (u, u') will be removed from R_{ij}.**

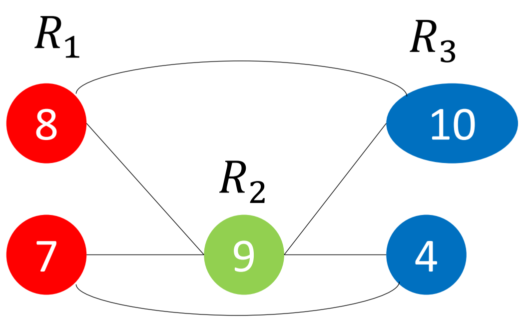
- For example, the tuple $(u_8, u_9) \in R_{12}$ has $u_{10} \in R_3$ such that $(u_8, u_{10}) \in R_{13}$ and $(u_9, u_{10}) \in R_{23}$, so do nothing.
- The tuple $(u_8, u_5) \in R_{12}$ do not have such $u \in R_3$, so removed.
- The tuple $(u_5, u_4) \in R_{23}$ do not have such $u \in R_1$, so removed.



(1) (u_8, u_5) and (u_5, u_4) is not satisfied, so they are removed.



(2) Removing (u_8, u_5) and (u_5, u_4) causes $u_5 \in R_2$ not satisfy R_{12} and R_{23} (*DomainFiltering*), so u_5 is removed.



(3) all tuples in R_{ij} are satisfied.

The relations participating natural joins are further reduced.

- Both running time and space are bounded by $O(n^3 D'^3)$.
- It is much more efficient than Natural Joins.
- It is valuable to do *RelationFiltering* after *DomainFilering* before Natural Joins.

n is the number of vertices in query Q.
$D' = \max\{|R_i|\}$ filtered by DomainFiltering.

Naïve Natural Joins

R_{12}

A	B
6	3
8	5
8	9
7	9

$\bowtie$

R_{23}

B	C
3	2
5	4
9	10
9	4

$\bowtie$

R_{31}

C	A
10	6
10	8
4	7

$=$

R_{123}

A	B	C
8	9	10
7	9	4

The relation Natural Joins becomes much easer and faster.

Filtered Natural Joins

R_{12}

A	B
8	9
7	9

$\bowtie$

R_{23}

B	C
9	10
9	4

$\bowtie$

R_{31}

C	A
10	8
4	7

$=$

R_{123}

A	B	C
8	9	10
7	9	4

6. Related Work

- *R-join* (Reachability Join) [1]:
  - The 2-hop labeling [3,4] is used to calculate the reachability relations, which can be extended to shortest-path distance relations.
  - The 2-hop labeling need too much indexing time and space.
- *MD-join* (Multi Distance-based Join) [2].
  - the filtering technique *LLR-embedding* [5] is used to speedup the shortest-path distance calculation.
  - The 2-hop labeling is used to verify the shortest-path distance.
  - The LLR-embedding and 2-hop labeling need too much indexing time and space.
- Both R-join and MD-join has no filtering and speed-up in Natural joins.

6. Experiments

- All methods are implemented by C++ on Visual Studio 2013.
- All methods run on a desktop computer with win10 64-bit operating system, Intel I7-7700 3.6GHz CPU and 14G RAM.
- All methods are evaluated over a variety of real data graphs.
- We expect to see our method has less index time and space, and also better performance, compared with other methods.

	Relation Construction (RC)	Matching Result Construction (MC)
ER-join (Extend Reachability Join) [1]	2-hop labeling [3,4]	Natural Join
MD-join (Multi Distance-based Join) [2]	2-hop labeling [3,4] LLR-embedding [5]	Natural Join
Ours	G^{Δ}	Domain Filtering Relation Filtering Natural Join

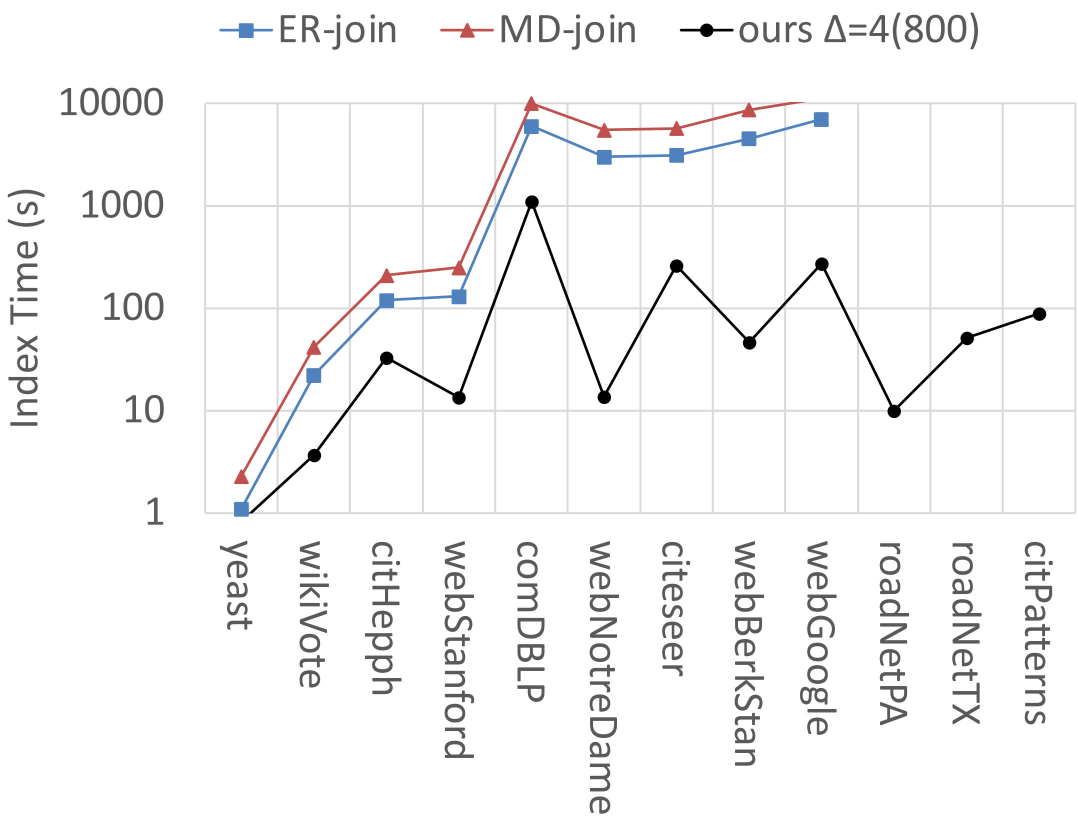
- All tested methods each include two parts: Relation construction and Matching Result Construction.

	index (offline)			query (online)	
	algorithms	time	space	algorithms	time
ER-join	2-hop labeling	$O(N^4)$	$O(N\sqrt{M})$	Extend R-join (ER-join)	$O(\sum_{ij} \sqrt{M}\lvert R_i\rvert\lvert R_j\rvert)$
				Natural Join (N-join)	$O(\prod_{ij} \lvert R_{ij}\rvert)$
MD-join	2-hop labeling	$O(N^4)$	$O(N\sqrt{M})$	D-join	$O(\sum_{ij} \sqrt{M}\lvert R_i\rvert\lvert R_j\rvert)$
	LLE-Embedding	$O(N^2 logN)$	$O(Nlog^2 N)$	Natural Join (N-join)	$O(\prod_{ij} \lvert R_{ij}\rvert)$
Ours	G^{Δ} by Dijkstra	$O(Nd^{\Delta} logN)$	$O(Nd^{\Delta})$	Domain Filtering(DF)	$O(mD^2)$
				Relation Filtering(RF)	$O(n^3 D'^3)$
				Natural Join (N-join)	$O(\prod_{ij} \lvert R''_{ij}\rvert)$

- The theoretical running index and query time.
- Our *DomainFiltering* and *RelationFiltring* are much faster than Natural Joins.

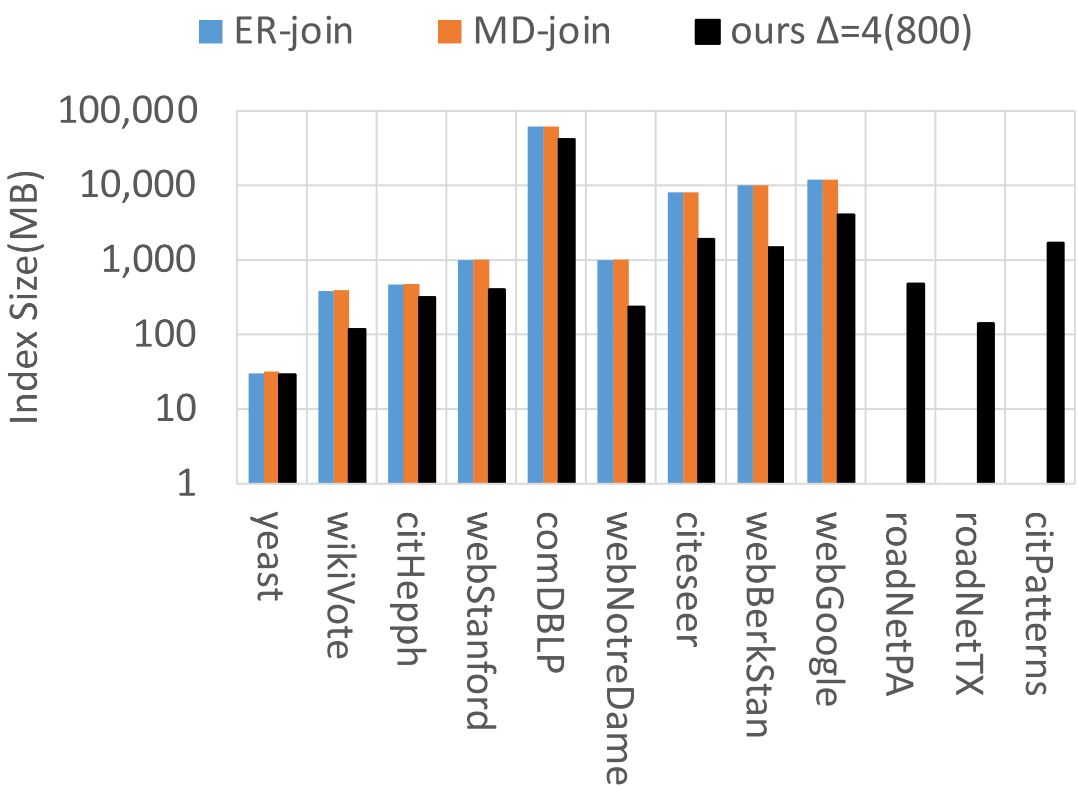
data graphs	directed	$N = \lvert V(G) \rvert$	$M = \lvert E(G) \rvert$	labeled	Label Size	weighted	avg. degree	density	avg. $\lvert R(l) \rvert$
yeast	no	2,361	7,182	yes	13	no	6.1	2.58E-03	196.8
wikiVote	yes	7,115	103,689	no	100	no	14.6	2.05E-03	71.2
citeHepph	yes	34,546	421,578	yes	124	1-1000	12.2	5.33E-04	278.6
webStanford	yes	281,903	2,312,497	no	100	no	8.2	3.53E-04	2,919.0
comDBLP	no	317,080	1,049,866	no	500	no	6.6	2.67E-04	23.5
webNotreDame	yes	325,729	1,497,134	no	100	1-1000	4.5	2.91E-05	3,257.3
citeseer	yes	384,413	1,751,463	no	500	no	4.6	2.09E-05	768.8
webBerkStan	yes	685,230	7,600,595	no	500	no	11.1	1.41E-05	3,426.2
webGoogle	yes	875,713	5,105,039	no	500	1-1000	5.8	1.19E-05	1751.4
roadNetPA	no	1,088,092	1,541,898	no	50	no	2.8	1.62E-05	21761.8
roadNetTX	no	1,379,917	1,921,660	no	20	1-1000	2.8	6.66E-06	68995.8
citePatterns	yes	3,774,768	16,518,948	no	100	1-1000	4.4	2.60E-06	37747.7

- There are totally 12 tested real graphs.
- All sorted by the number of vertices.
- The last 3 graphs are much larger than others.



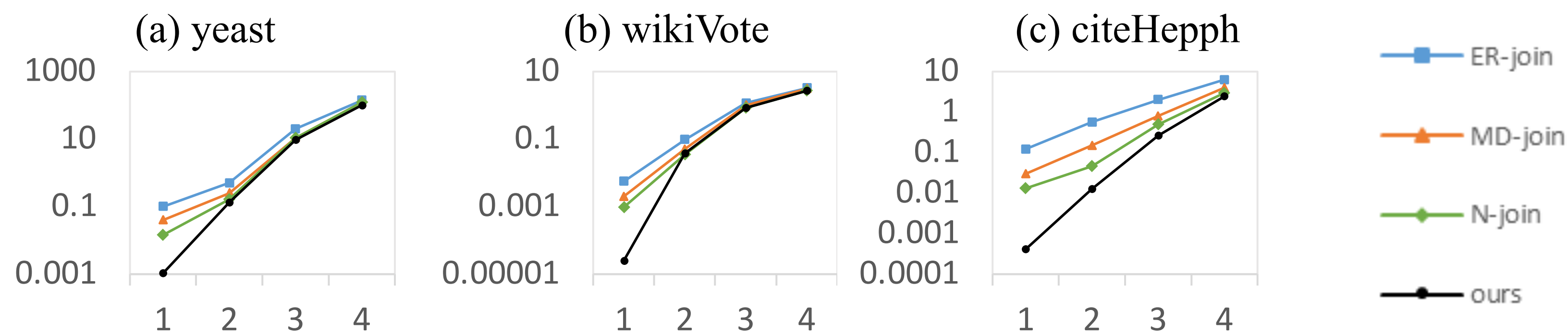
Exp. 1.1
Index Time

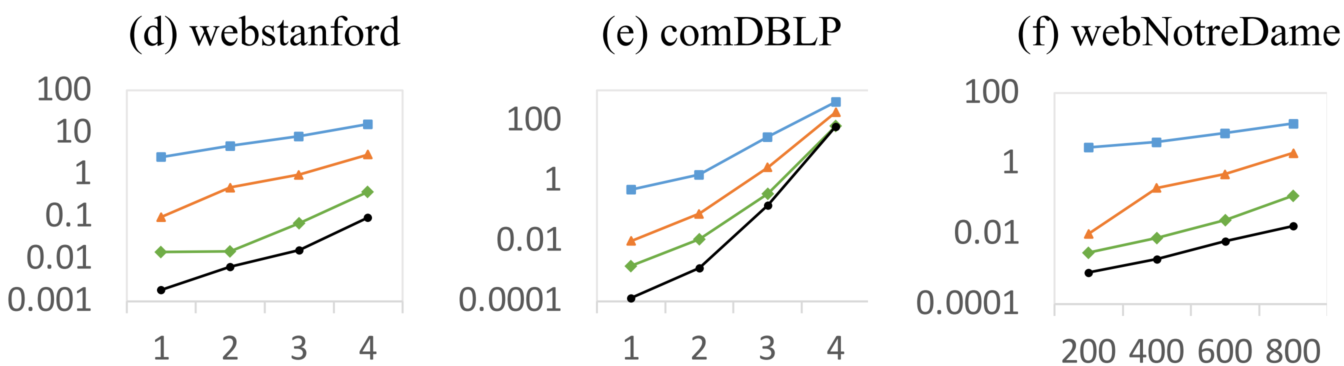
- Our method has much less index time.
- The index times of ER-join and MD-join for 3 largest graphs are not listed here. All of them over 3 hours.



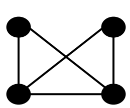
Exp. 1.2
Index Size

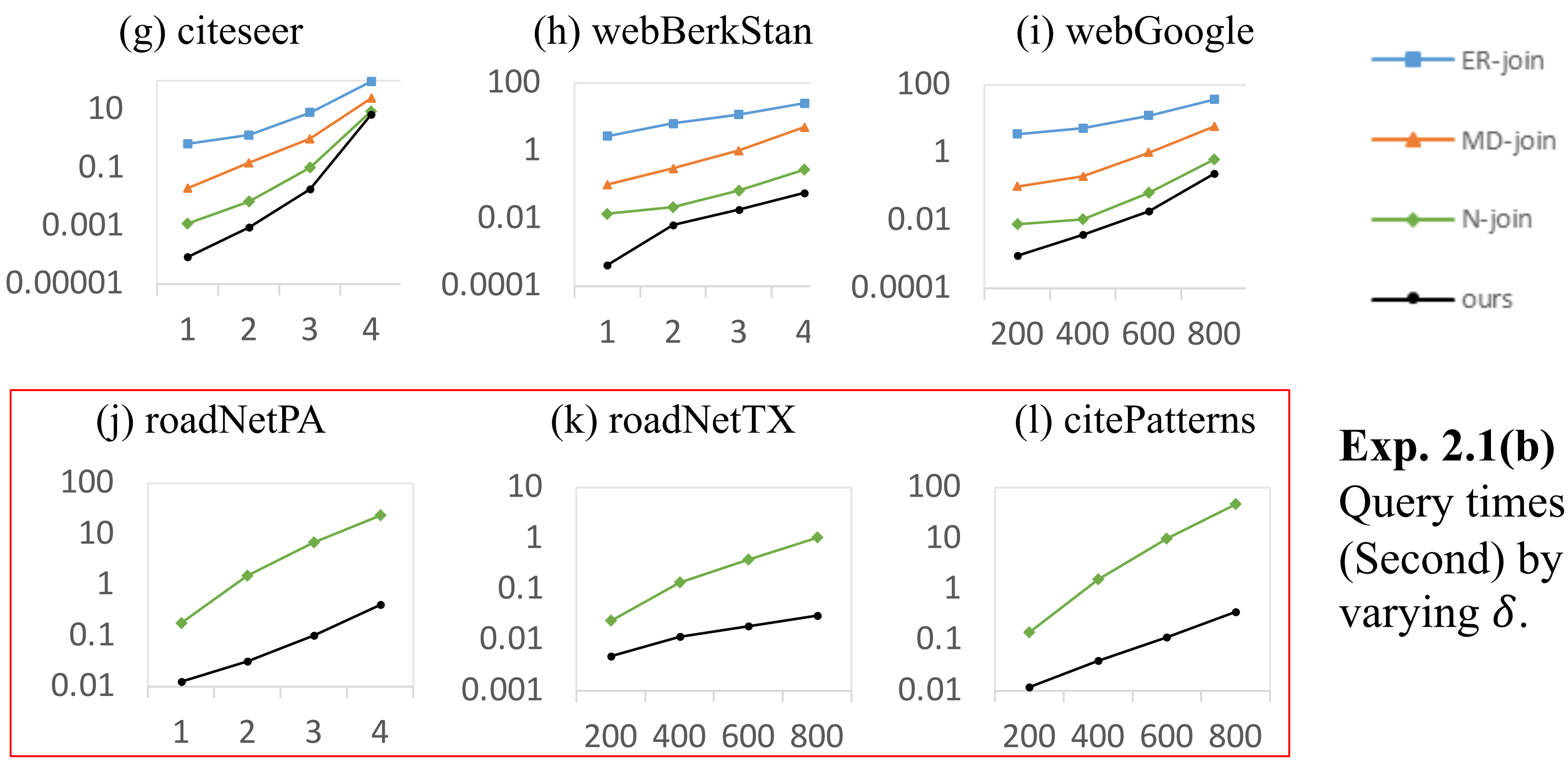
- Our method has much less index size.
- The index size of 3 largest graphs are not listed, since they over 3 hours.





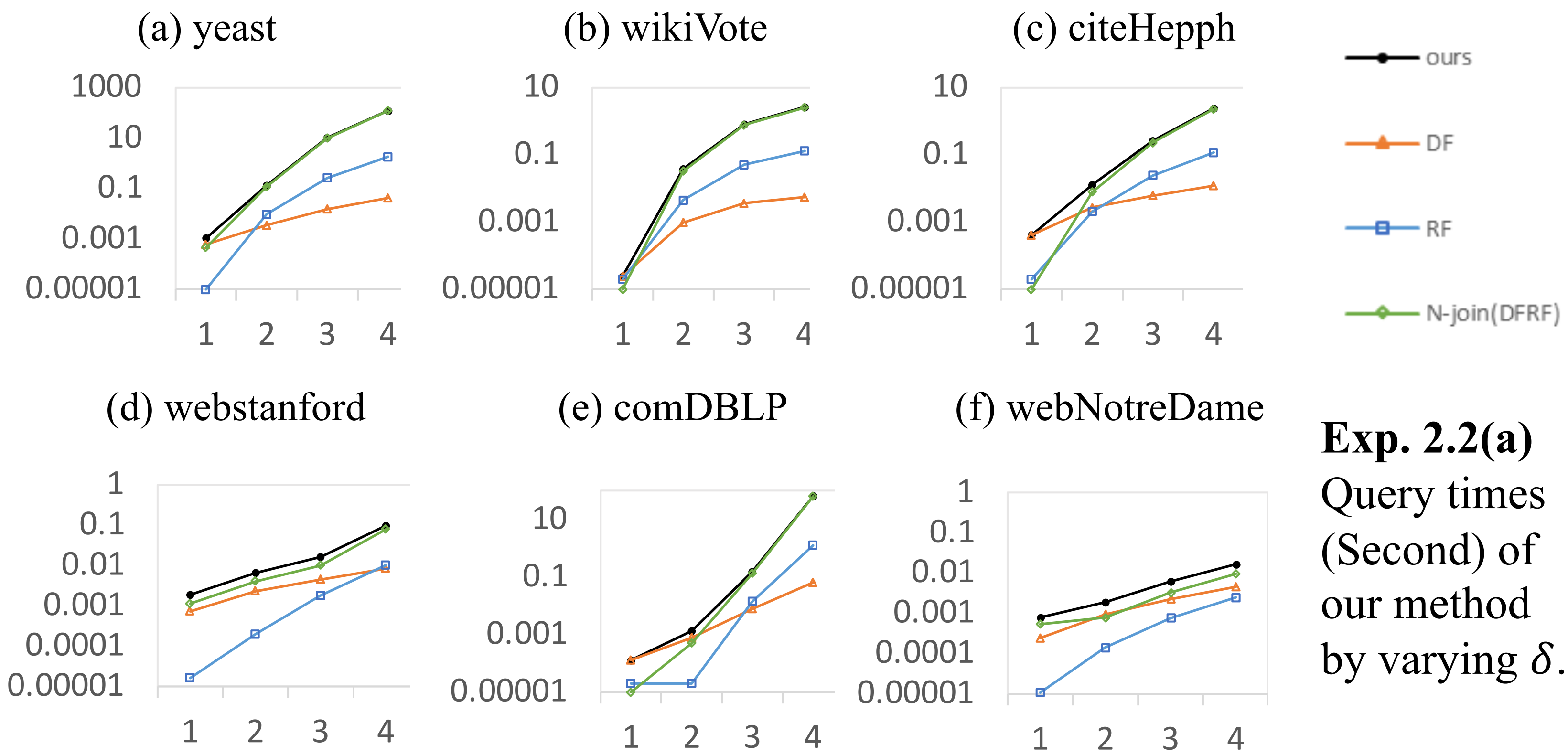
Exp. 2.1(a) Query times (Second) by varying δ.

- Fixed query graph: 
- For unweighted graph $\delta = 1,2,3,4$; for weighted graphs $\delta = 200,400,600,800$.

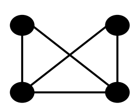


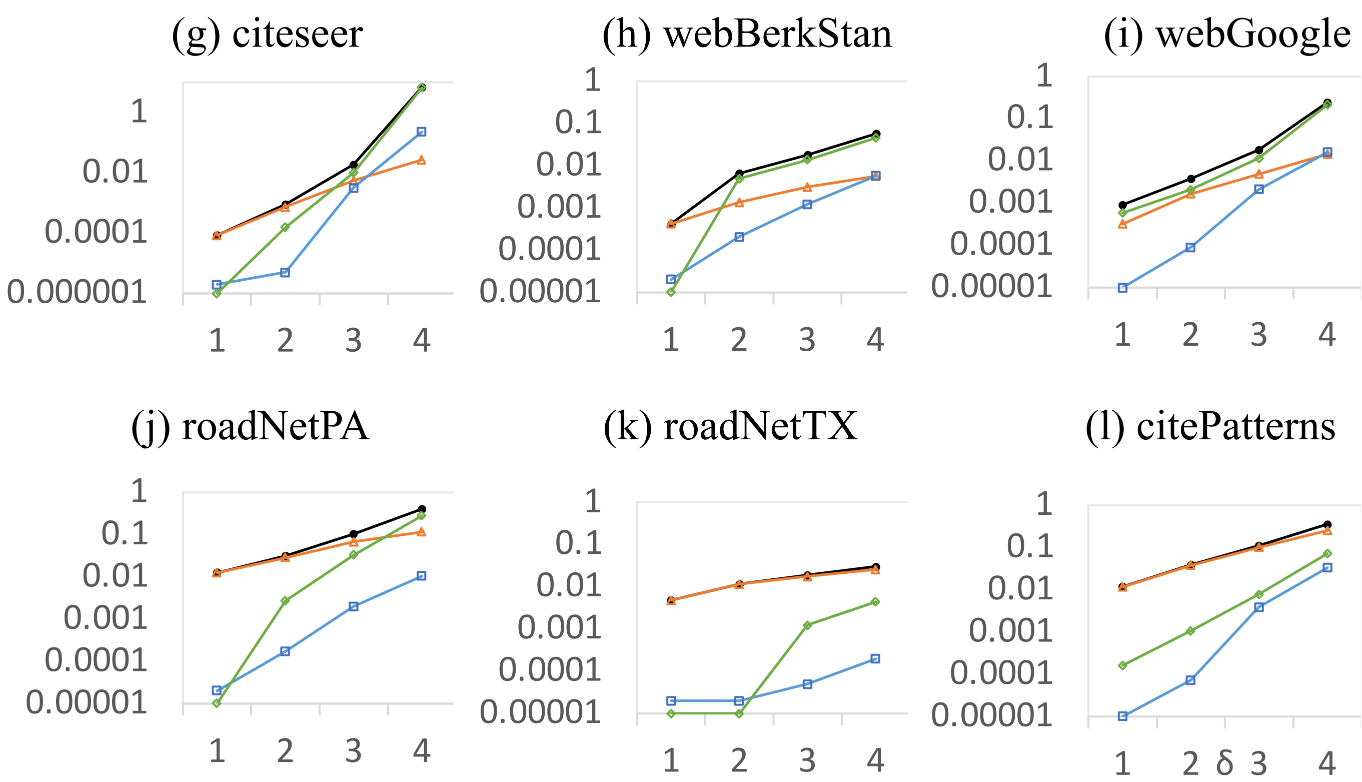
Exp. 2.1(b) Query times (Second) by varying δ.

- Our method is much faster than others, especially when δ is small.
- Our method even faster than Natural Joins since it does not use time for relation construction and our *DomainFiltering* and *RelationFiltering* can efficiently reduce relations participating Natural Joins.



Exp. 2.2(a) Query times (Second) of our method by varying δ.

- Fix query graph: 
- For unweighted graph $\delta = 1,2,3,4$; for weighted graphs $\delta = 200,400,600,800$.



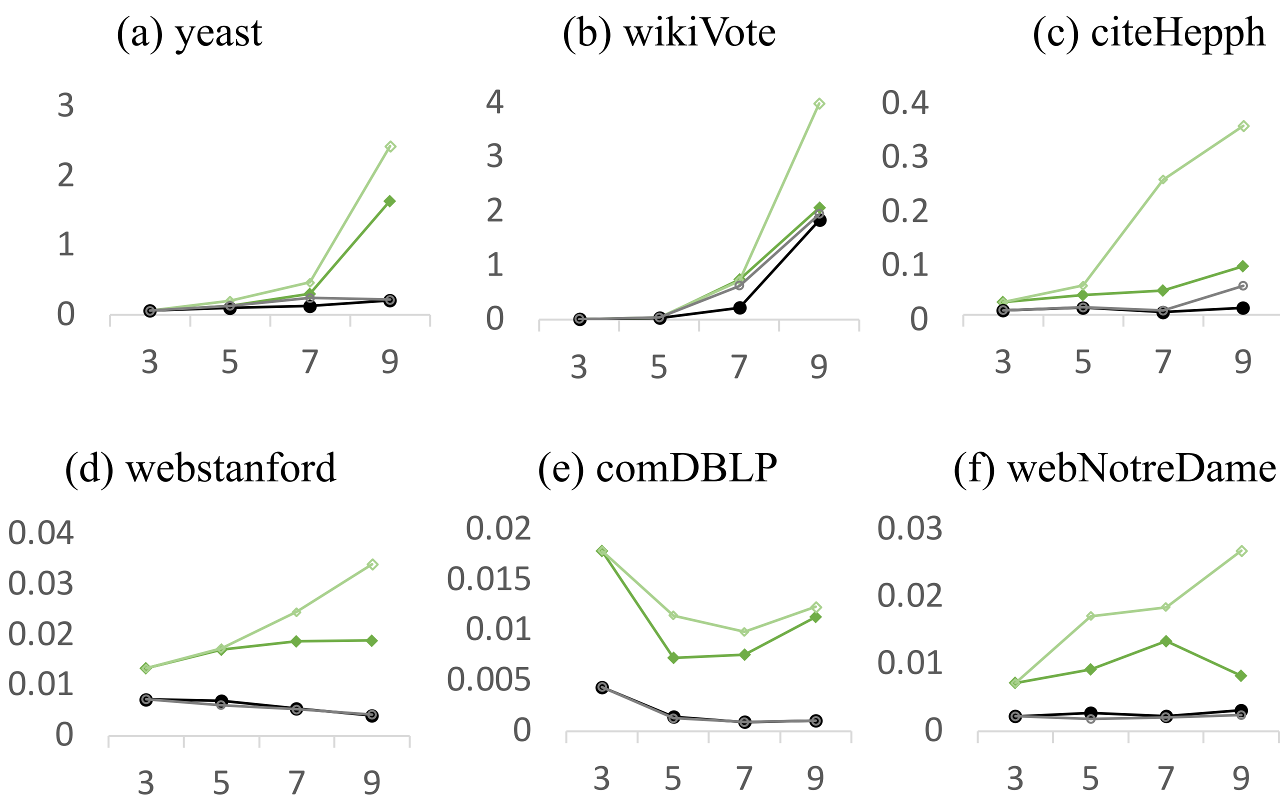
Exp. 2.2(b) Query times (Second) of our method by varying δ.

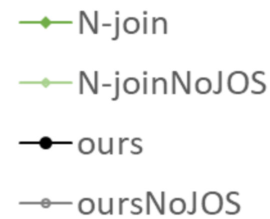
- Our *DomainFiltering* and *RelationFiltering* are efficient, nearly linearly increasing with δ.
- Most of running time is spent on Natural Joins after filtering.

graphs	δ = 1(200)			δ = 2(400)			δ = 3(600)			δ = 4(800)		
	total	after DF	after RF	total	after DF	after RF	total	after DF	after RF	total	after DF	after RF
yeast	540	9	9	7440	3220	2822	44442	22113	22049	125000	62470	62470
wikiVote	60	0	0	1407	1266	1145	6233	5794	5774	10461	9794	9794
citeHepph	308	0	0	2146	1386	392	7011	6347	4356	15417	14893	12954
webStanford	388	18	18	1466	127	127	3214	1092	632	6628	3819	2407
comDBLP	108	0	0	1140	14	14	12926	4832	3257	120904	59709	58617
webNotreDame	134	12	12	600	267	25	1886	1196	66	4296	3165	170
citeseer	45	0	0	536	5	5	4268	2626	504	28259	27217	22774
webBerkStan	264	0	0	1215	159	150	2782	693	529	5449	2090	1342
webGoogle	175	9	9	929	68	68	3359	1168	843	9161	5924	4397
roadNetPA	12380	0	0	35144	98	98	71500	1114	1113	125004	5488	5405
roadNetTX	3776	0	0	9624	0	0	18200	42	42	29860	131	131
citePatterns	5538	5	5	18378	40	30	44625	3515	446	92373	24361	2818

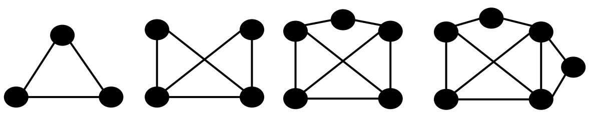
Exp. 2.3 The total tuple numbers of relations by varying δ.

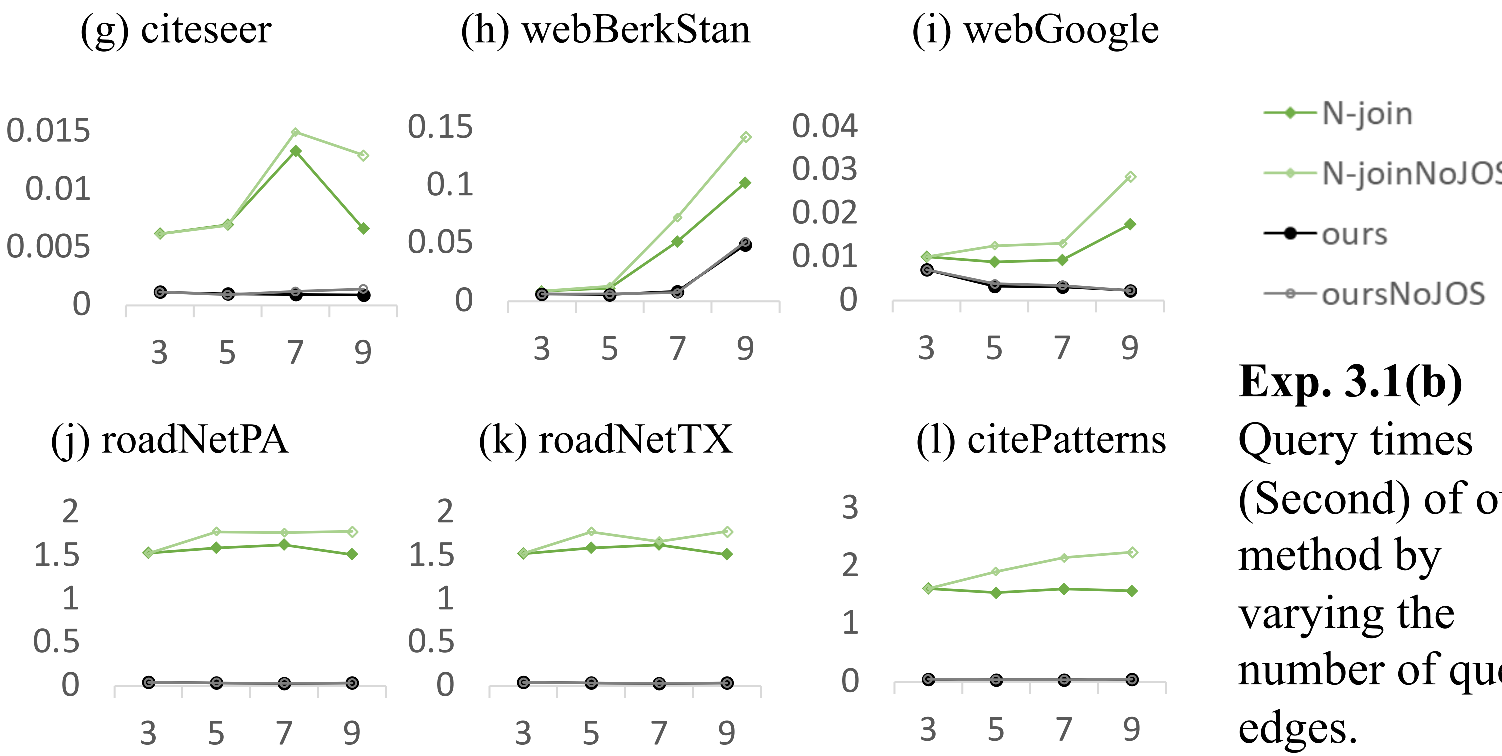
- The *DomainFiltering* can remove much more useless tuples than *RelationFiltering*.
- When δ is smaller only a few tuples are left after DF&RF, which leads to great speed-up to the Natural Joins.





Exp. 3.1(a) Query times (Second) of our method by varying the number of query edges.

- Vary query graph:  with the number of edges = 3, 5, 7, 9.
- For unweighted graph fix $\delta = 2$; for weighted graphs fix $\delta = 400$.



Exp. 3.1(b) Query times (Second) of our method by varying the number of query edges.

- Our method has a higher speed-up over Natural Joins with the increasing of query edges.

	$\lvert E(Q)\rvert = 3$			$\lvert E(Q)\rvert = 5$			$\lvert E(Q)\rvert = 7$			$\lvert E(Q)\rvert = 9$		
graphs	total	after DF	after RF	total	after DF	after RF	total	after DF	after RF	total	after DF	after RF
yeast	5300	2364	2115	7440	3220	2822	8062	3049	2506	9980	3624	2962
wikiVote	808	741	692	1407	1266	1145	2223	1978	1833	3067	2714	2532
citeHepph	1280	931	336	2146	1386	392	2821	1676	423	3435	1885	479
webStanford	945	197	194	1466	127	127	2171	92	91	3011	44	44
comDBLP	708	59	56	1140	14	14	1532	0	0	1884	0	0
webNotreDame	398	208	33	600	267	25	746	35	34	1037	47	46
citeseer	319	7	6	536	5	5	786	0	0	1017	0	0
webBerkStan	560	109	91	1215	159	150	1736	223	214	2058	227	227
webGoogle	511	88	76	929	68	68	1254	62	62	1549	34	34
roadNetPA	21148	789	788	35144	98	98	49246	7	7	63686	0	0
roadNetTX	5914	114	114	9624	0	0	13588	0	0	17534	0	0
citePatterns	11015	458	362	18378	40	30	25810	10	10	33262	0	0

Exp. 3.2 The total tuple numbers of relations by varying the number of query edges.

- More query edges produce more total tuples in all relations.
- But more query edges lead to more restrictions. Therefore, *DomainFiltering* and *RelationFiltering* can filter more useless tuples.

9. Future Work

- The G^{Δ} is proposed to the relation construction. The *DomainFiltering* and *RelationFiltering* is proposed to reduce the searching space of final Natural Join.
- However, the index time and size should be optimized in order to well handle large data graphs.
- Also, the shortest-path distance limitation may be extended to other kind of distance.

10. Reference

- [1] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang, “Fast graph pattern matching,” *Proc. - Int. Conf. Data Eng.*, pp. 913–922, 2008.
- [2] L. Chen, “Distance-Join : Pattern Match Query In a Large Graph,” *Vldb*, vol. 2, no. 1, pp. 886–897, 2009.
- [3] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and Distance Queries via 2-Hop Labels,” *SIAM J. Comput.*, vol. 32, pp. 1338–1355, 2003.
- [4] J. Cheng and J. X. Yu, “On-line exact shortest distance query processing,” *Proc. 12th Int. Conf. Extending Database Technol. Adv. Database Technol. EDBT 09*, p. 481, 2009.
- [5] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, “A road network embedding technique for K-nearest neighbor search in moving object databases,” *Geoinformatica*, vol. 7, no. 3, pp. 255–273, 2003.

Thanks!
Bin Guo