Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching

Hyunjoon Kim[†↕], Yunyoung Choi[†], Kunsoo Park[†], Xuemin Lin[‡], Seok-Hee Hong[‖], Wook-Shin Han[◊]

[†] Seoul National University
[‡] University of New South Wales
[‖] University of Sydney
[◊] Pohang University of Science and Technology (POSTECH)
[↕]SAP Labs Korea

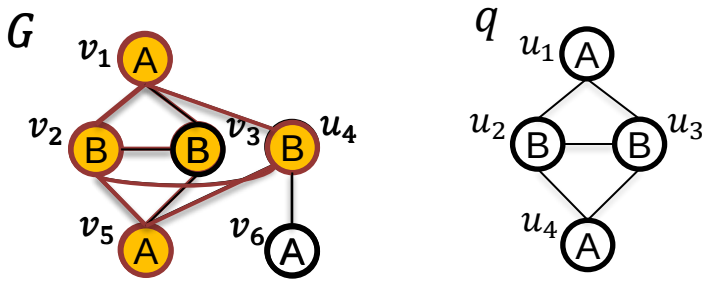
SIGMOD 2021

Outline

- Problem Statement
  - Subgraph Matching
  - Subgraph Search (Subgraph Query Processing)
- Overview of the algorithms
- Three main techniques
  - Filtering by Neighbor-Safety
  - Matching Order Based on Static Equivalence
  - Run-Time Pruning by Dynamic Equivalence
- Performance Evaluation
- Conclusion

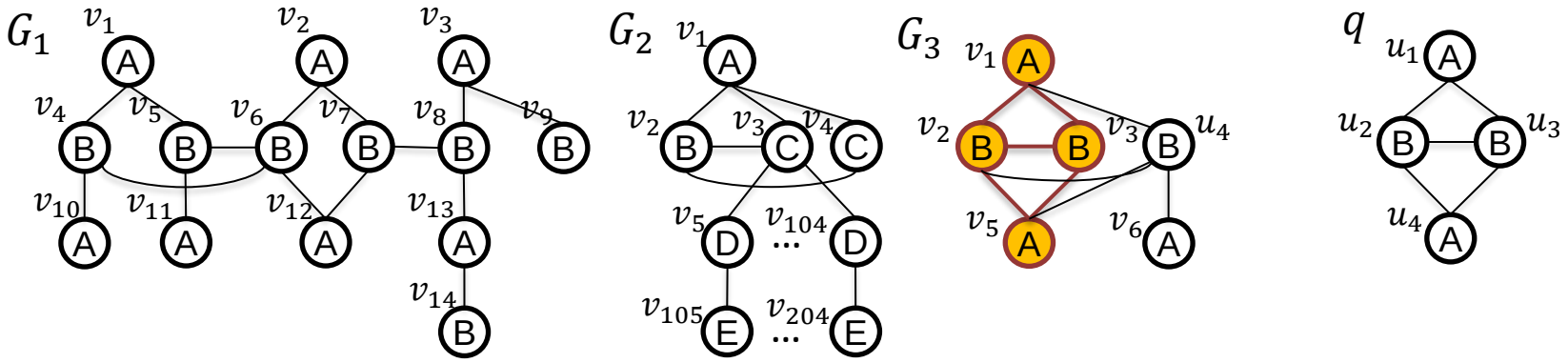
Problem Statement

- Given
  - a query graph $q = (V(q), E(q), l_q)$
  - a data graph $G = (V(G), E(G), l_G)$
- **Embedding** of q in G is a mapping $M : V(q) \rightarrow V(G)$ such that:
  1. M is injective. (i.e. $M(u) \neq M(u')$ for $u \neq u'$),
  2. $L_q(u) = L_G(M(u))$ for every $u \in V(q)$,
  3. $(M(u), M(u')) \in E(G)$ for every $(u, u') \in E(q)$.
- Subgraph Matching
  - Find all embeddings of q in G (NP-hard)



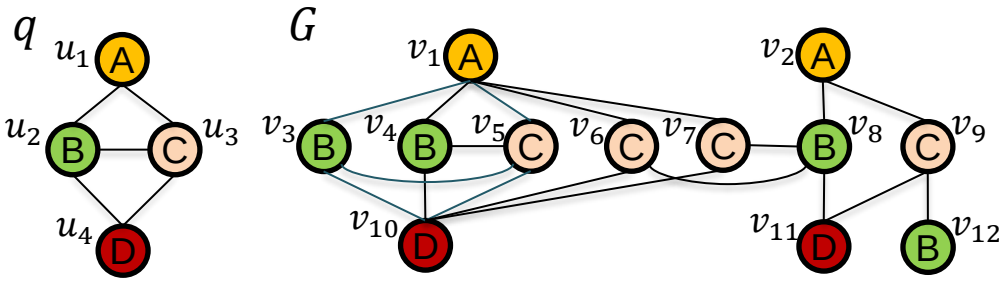
Problem Statement

- Given
  - a set of data graphs $D = \{G_1, G_2, \ldots, G_m\}$
  - a query graph q
- Subgraph Search (or Subgraph Query Processing)
  - Find all the data graphs in D that contain q as subgraphs (NP-hard)
    - $A_q = \{G \in D \mid q \subseteq G\}$.
      - $A_q = \{G_3\}$
- Applications
  - Social network analysis, Protein-protein interaction networks, chemical compound search, RDF query processing



Related Work

- Subgraph search
  - Indexing-filtering-verification
    - CT-index [ICDE 2011],
    - SING [BMC Bioinformatic 2010],
    - GraphGrepSX [IAPR 2010],
    - Grapes [PloS one 2013]
  - Preprocessing-enumeration
    - CFQL [ICDE 2019]
- Subgraph matching
  - Preprocessing-enumeration
    - $\text{Turbo}_{\text{iso}}$ [SIGMOD 2013],
    - CFL-Match [SIGMOD 2016],
    - DAF [SIGMOD 2019],
    - GQLfs [SIGMOD 2020]
  - Direct enumeration
    - RIfs [SIGMOD 2020]
  - Constraint programming
    - Glasgow [ICGT 2020]

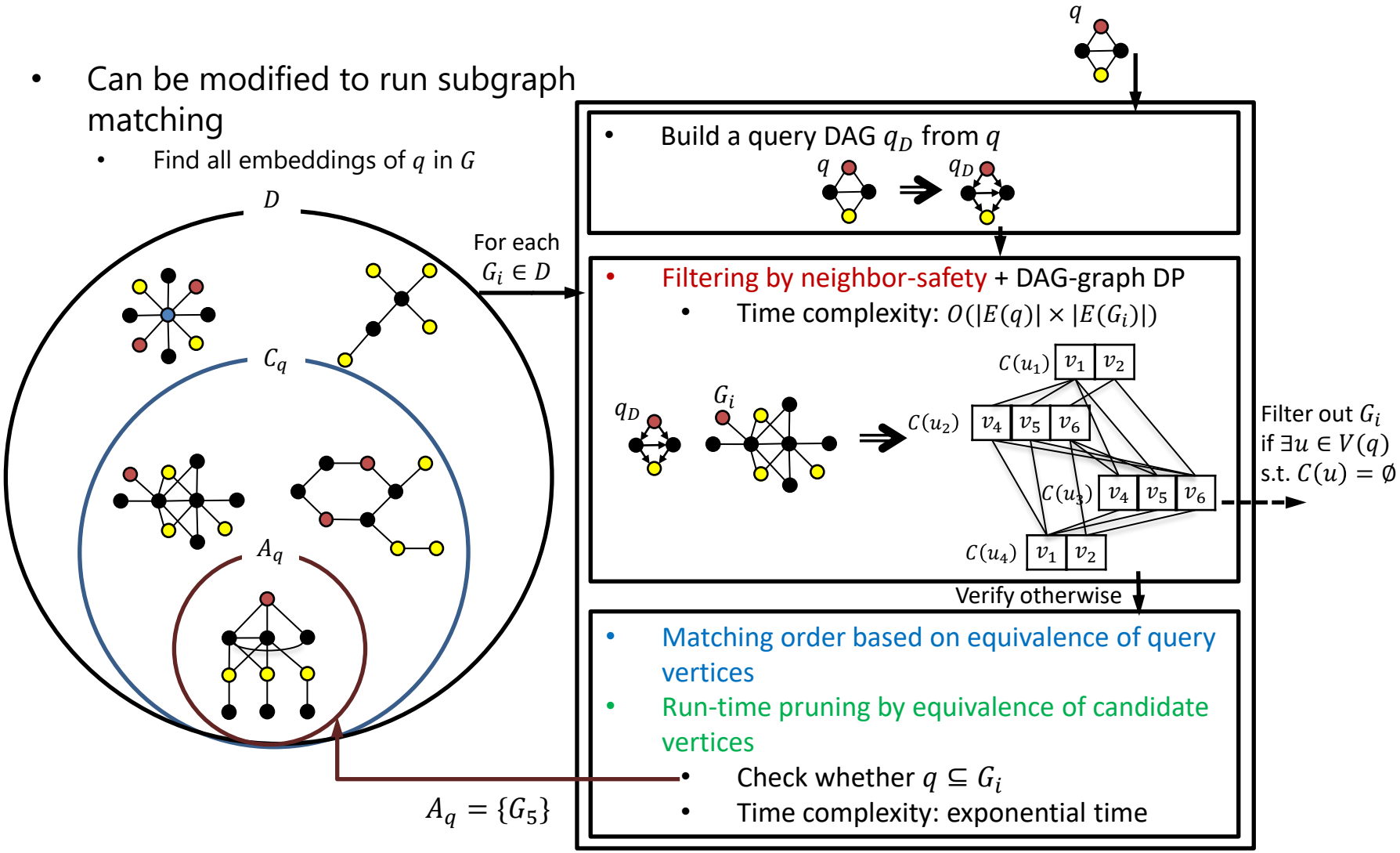


- Framework
  1. Adopt a filtering process to find a candidate set $C(u)$ for each $u \in V(q)$, where $C(u)$ is a subset of $V(G)$ which u can be mapped to (e.g., $C(u_1) = \{v_1, v_2\}$).
  2. Choose a linear order of the query vertices, called matching order, and apply backtracking based on the matching order (e.g., (u_1, u_2, u_3, u_4)).

- Challenges
  1. Limitations of existing indexing methods
  2. Sub-optimal matching order
  3. Redundant computations in search
- Preprocessing-enumeration
  - VEQ [SIGMOD 2021] → this
    - $>10^2 \times$ faster than DAF in subgraph search
    - $>10^2 \times$ faster than GQLfs in subgraph matching

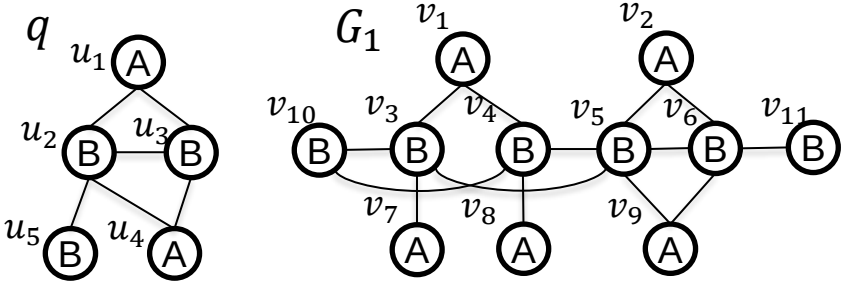
Overview of Our Subgraph Search Algorithm

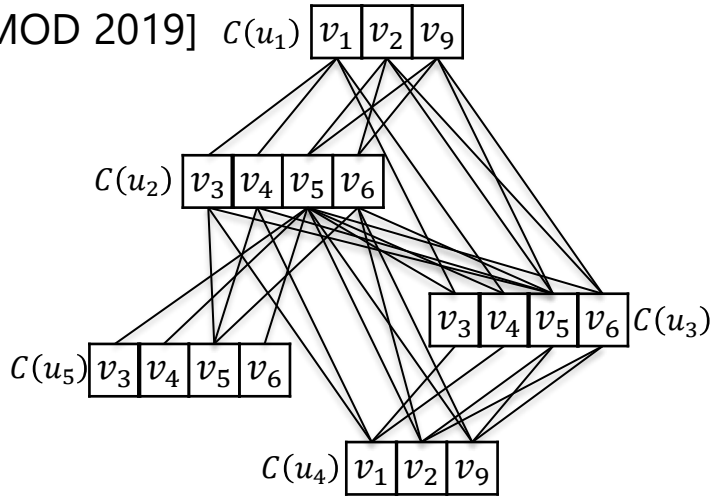
- Can be modified to run subgraph matching
  - Find all embeddings of q in G



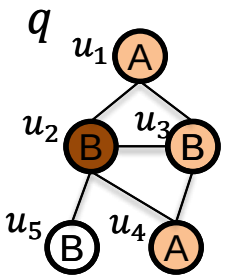
Candidate Space

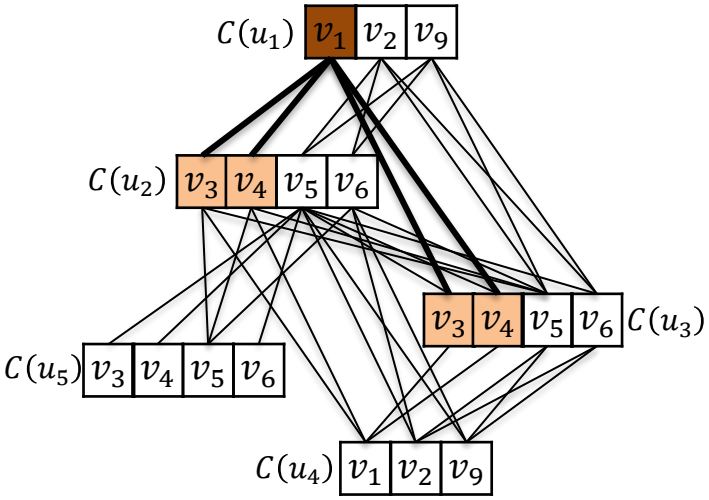
- **Candidate Space (CS) on q and G** consists of the candidate set $C(u)$ for each $u \in V(q)$, and edges between the candidates.
  - For each $u \in V(q)$ there is a **candidate set $C(u)$**, which is a set of vertices in G that u can be mapped to.
  - There is **edge** btw. $v \in C(u)$ and $v' \in C(u')$ iff $(u, u') \in E(q)$ and $(v, v') \in E(G)$.
- How do we get compact CS?
  - Filtering by neighbor-safety
  - DP between DAG and graph [Han et al., SIGMOD 2019]



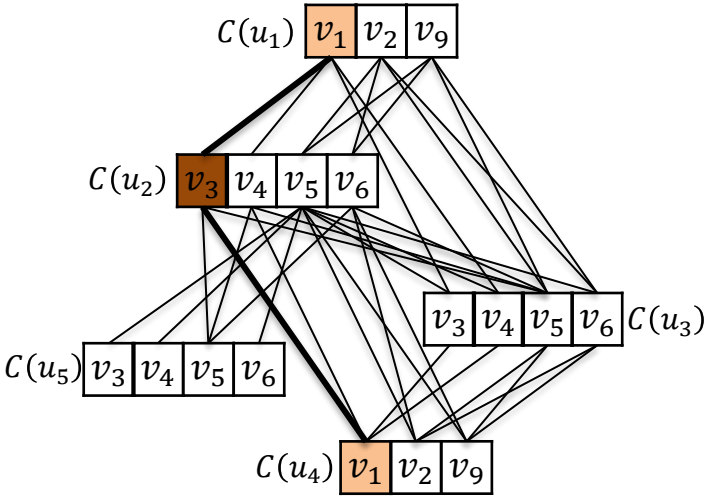


Filtering by Neighbor-Safety





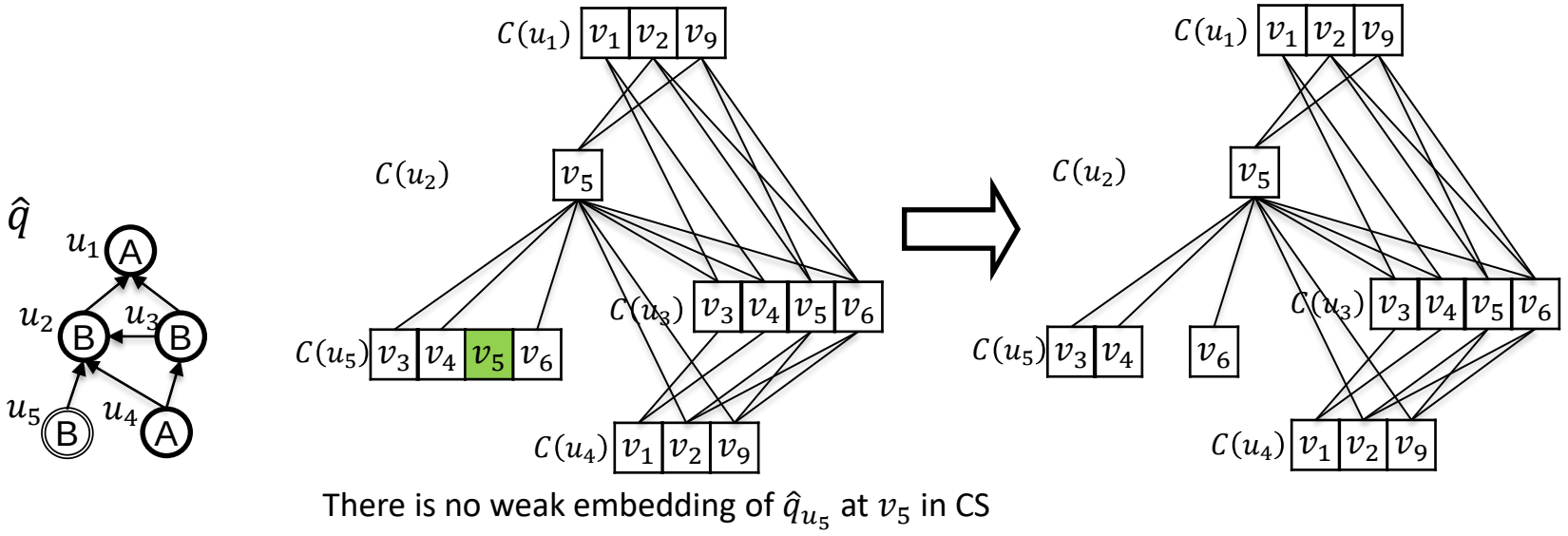
$v_1 \in C(u_1)$ is neighbor-safe regarding u_1



$v_3 \in C(u_2)$ is not neighbor-safe regarding u_2

- For every vertex $u \in V(q)$ and its candidate vertex $v \in C(u)$, $v \in C(u)$ is **neighbor-safe regarding $\boldsymbol{u}$** if for $\forall l \in \Sigma$, $|Nbr_q(u,l)| \leq |Nbr_{CS}(u,v,l)|$.
  - $\boldsymbol{Nbr_q(u,l)}$ is a set of neighbors of u labeled with l in q
  - $\boldsymbol{Nbr_{CS}(u,v,l)}$ $= \cup_{u_n \in Nbr_q(u,l)} \{v_n \in C(u_n) \mid v_n \text{ is adjacent to } v \in C(u) \text{ in } CS\}$.
- Filter out v from $C(u)$ if v is not neighbor-safe regarding u.
  - Necessary condition $h(u,v)$ of embedding

DAG-Graph DP [Han et al., SIGMOD 2019]

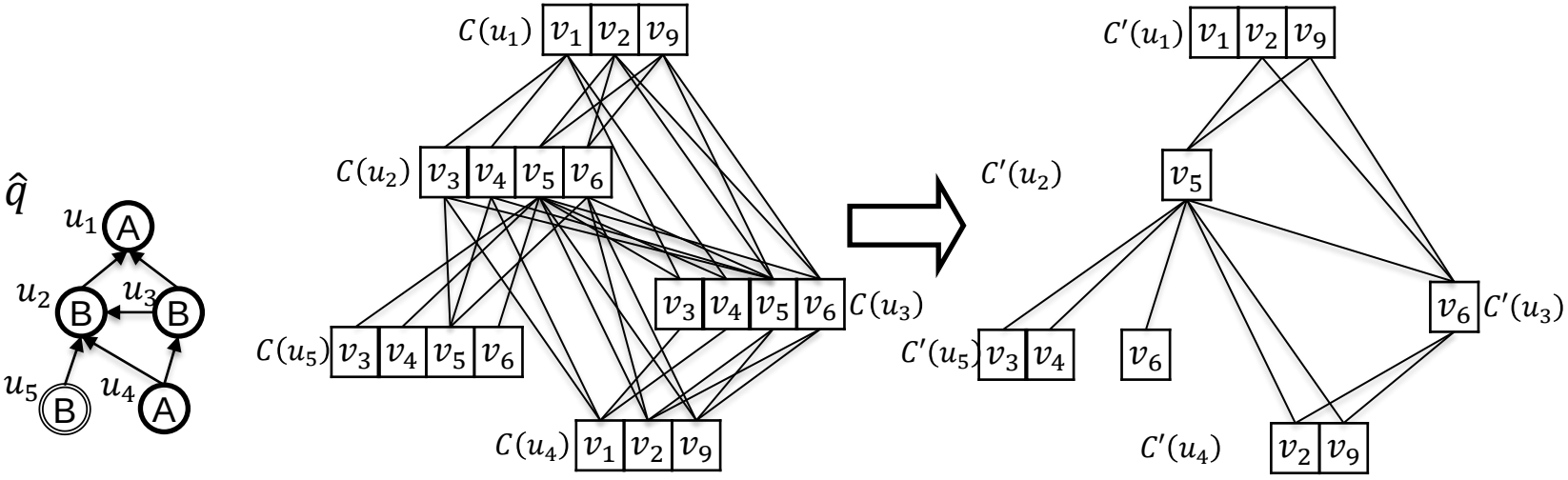


There is no weak embedding of $\hat{q}_{u_5}$ at v_5 in CS

- Given query DAG $\hat{q}$ built from q
- Filter out $v \in C(u)$ from $C(u)$ if there is no weak embedding of $\hat{q}_u$ at v in CS
  - Necessary condition $f(u, v)$ of embedding

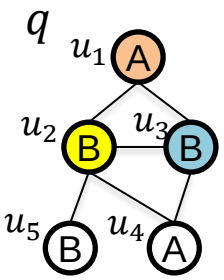
Extended DAG-Graph DP

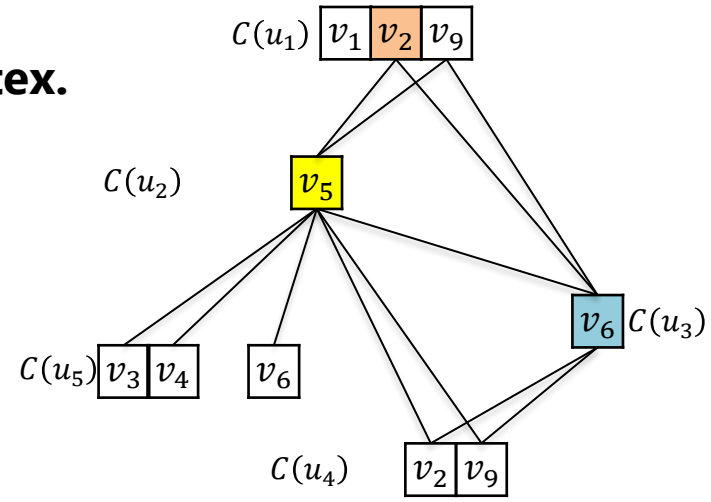
- From every $u \in V(q)$ and $v \in C(u)$, we compute more compact candidate set $C'(u)$
  - $\boldsymbol{v \in C'(u)}$ if $\boldsymbol{v \in C(u)}$ and the following necessary conditions for an embedding that maps u to v holds.
    1. There is a **weak embedding M of a sub-DAG q_u at v** in the CS.
    2. Necessary condition $\boldsymbol{h(u, v)}$ is true in the CS.
- Compute new $C'(u)$ by dynamic programming in bottom-up fashion.



Matching Order

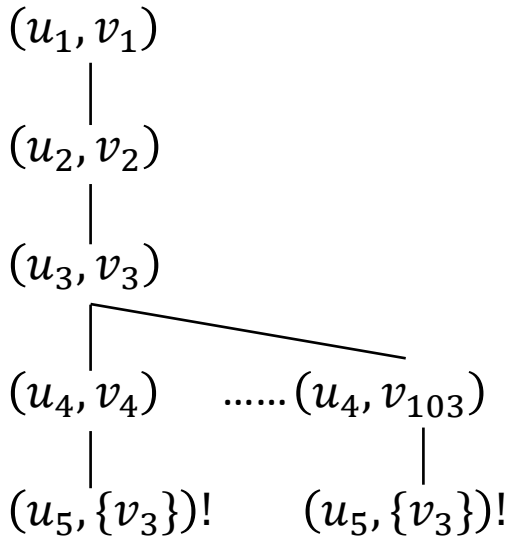
- Unmapped vertex $u \in V(q)$ in M is called **extendable** regarding M if at least one neighbor of u is matched in M
- Set $C_M(u)$ of extendable candidates of u regarding M: set of vertices $v \in C(u)$ adjacent to $M(u_n)$ in CS for every mapped neighbor u_n of u.
- Set $U_M(u)$ is the set of unmapped extendable candidates of u in $C_M(u)$
- Given a mapping $M = \{(u_1, v_2), (u_2, v_5), (u_3, v_6)\}$
  - u_4, u_5 are extendable vertices.
  - $C_M(u_4)$ of extendable candidate is $\{v_2, v_9\}$.
  - $U_M(u_4)$ of is $\{v_9\}$.
- **Select extendable vertex as the next vertex.**



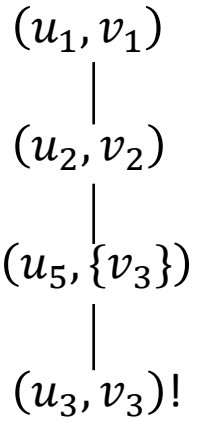


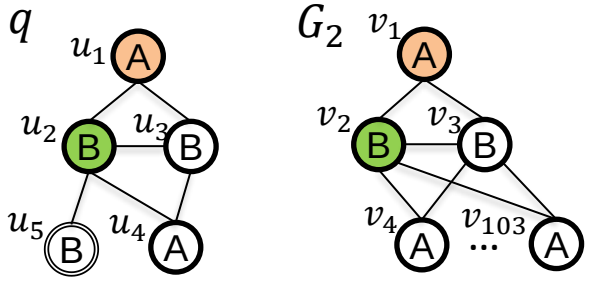
Matching Order Based on Static Equivalence

Existing Matching Order



New Matching Order

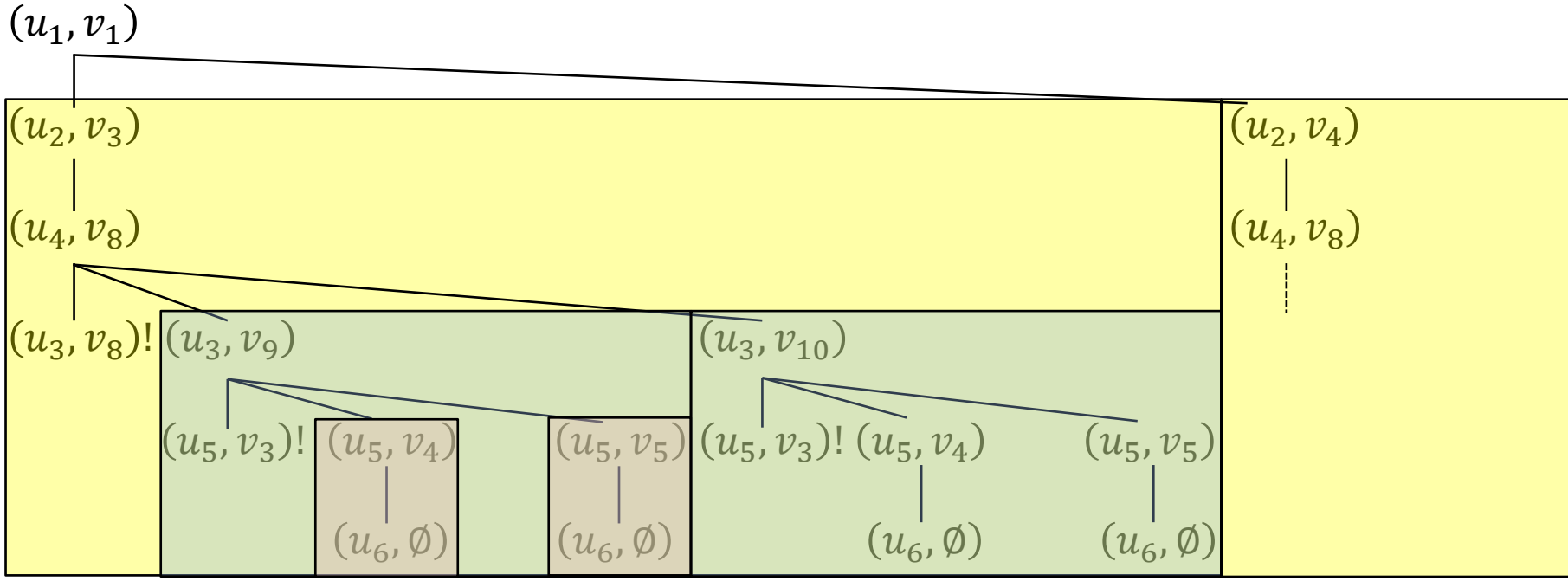




- Neighbor Equivalence Class (NEC)
  - Set of query vertices with the same label and the same neighbors [Han et al., SIGMOD 2013]
  - $NEC(u_5) = \{u_5\}$
  - $U_M(u_5) = \{v_3\}$

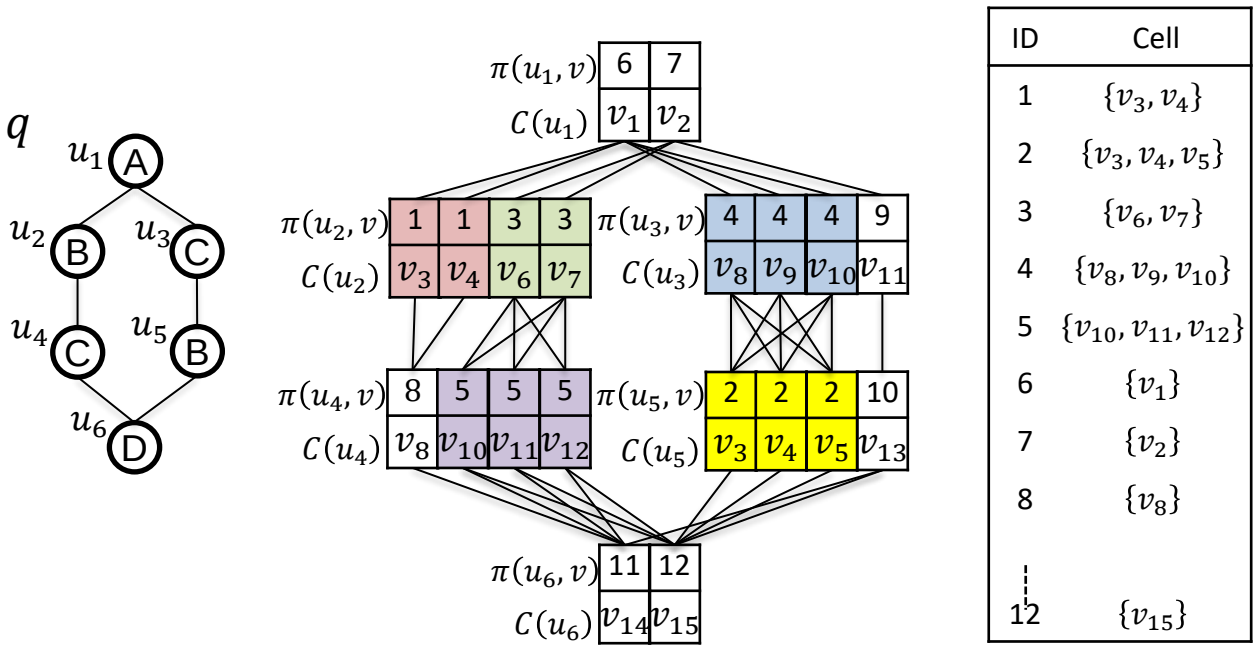
- If there is a degree-one extendable vertex u s.t.
  - $|NEC(u)| > |U_M(u)|$, backtrack.
  - $|NEC(u)| = |U_M(u)|$, select u as the next vertex.
- Otherwise,
  - If there are only degree-one vertices, select a degree-one vertex.
  - Otherwise, select an extendable vertex u s.t. $|C_M(u)|$ is minimum among non-degree-one vertices.

Search Tree of Backtracking



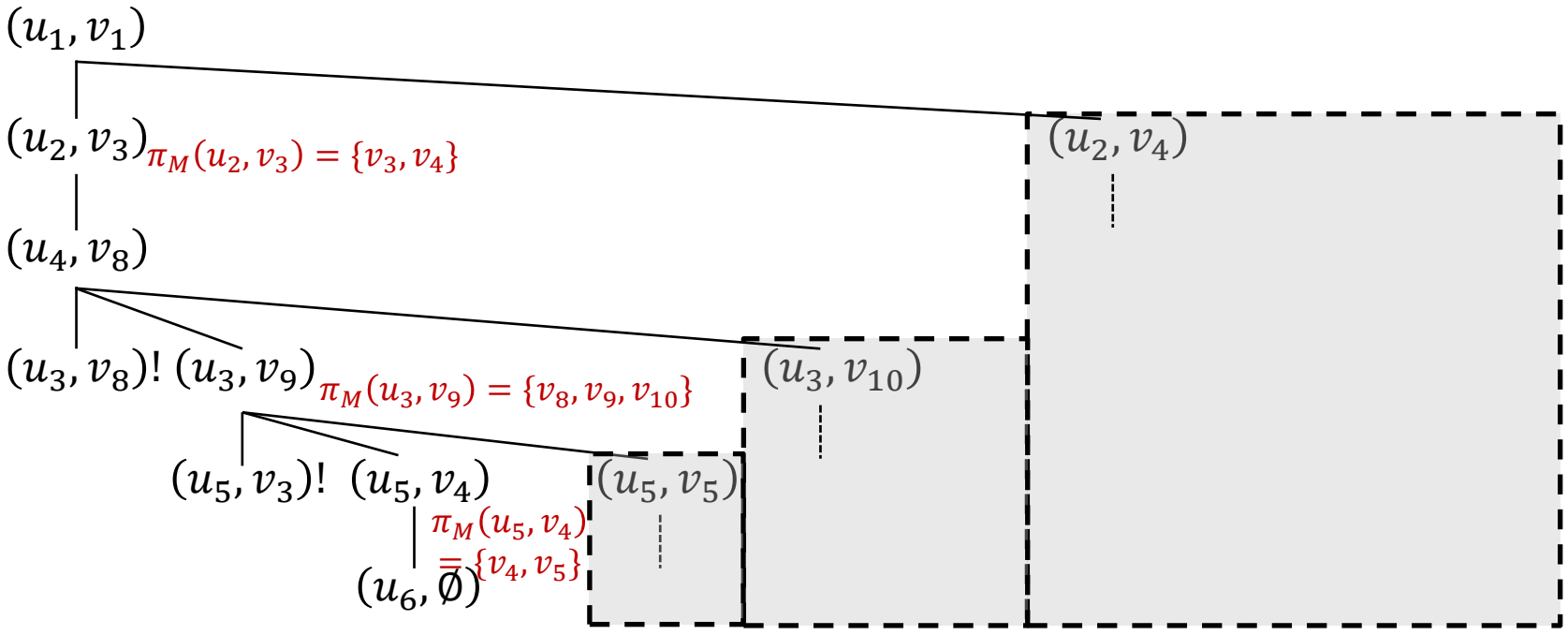
- Finished the exploration of the subtree rooted at $M \cup \{(u, v_i)\}$ in backtracking
- Given:
  - a partial embedding M,
  - an extendable vertex $u \in V(q)$, and its candidate $v_i \in C_M(u)$.
- Prune out partial embeddings (subtrees) equivalent to node $M \cup \{(u, v_i)\}$ by utilizing equivalence of candidate vertices in CS.

Run-Time Pruning By Dynamic Equivalence



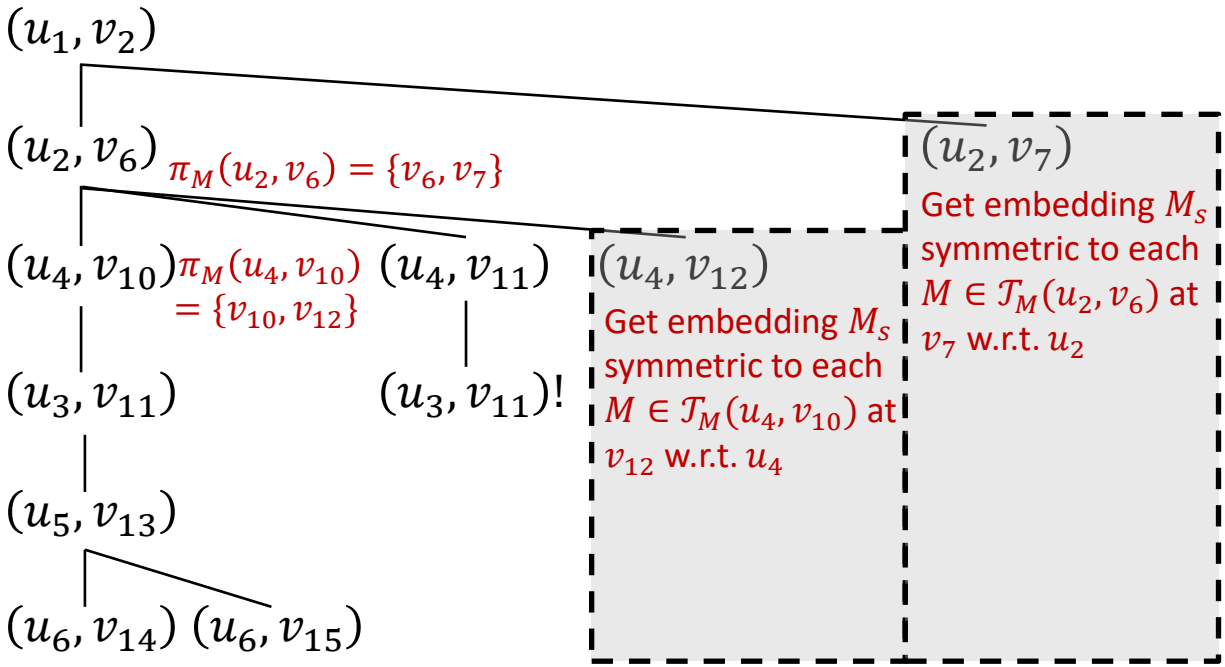
- For $u \in V(q)$ and $v \in C(u)$,
- **Cell $\boldsymbol{\pi(u, v)}$**: set of vertices $v' \in C(u)$ that share neighbors with v in CS.
  - $\pi(u_2, v_3) = \{v_3, v_4\}$
- We define **equivalence set** $\pi_M(u, v_i)$, which is **a subset of $\boldsymbol{\pi(u, v_i)}$**.

Run-Time Pruning By Dynamic Equivalence



- $\mathcal{T}_M(u, v_i)$: set of embeddings in the subtree rooted at $M \cup \{(u, v_i)\}$.
- If $\mathcal{T}_M(u, v_i) = \emptyset$
  - Avoid exploring the subtree rooted at $M \cup \{(u, v_j)\}$ in which no embedding will be found for each $\boldsymbol{v_j} \in \boldsymbol{\pi_M(u, v_i)}$.

Run-Time Pruning By Dynamic Equivalence



- Otherwise ($\mathcal{T}_M(u, v_i) \neq \emptyset$),
  - For each $\boldsymbol{v_j} \in \boldsymbol{\pi_M(u, v_i)}$, obtain all embeddings in $\mathcal{T}_M(u, v_j)$ without exploring the subtree rooted at $M \cup \{(u, v_j)\}$.
    - Only used in subgraph matching

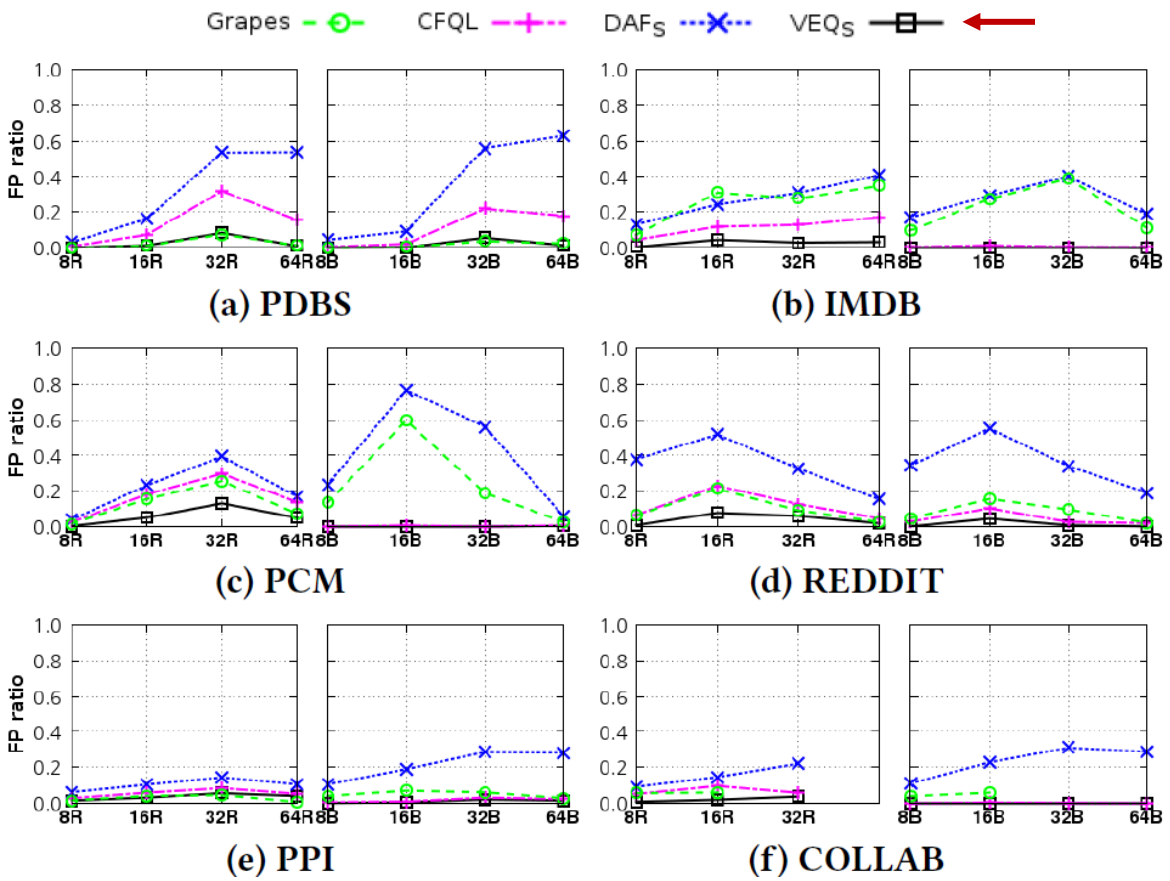
Performance Evaluation

Subgraph Search	Subgraph Matching
$\mathrm{VEQ_S}$, Grapes, CFQL, DAF	$\mathrm{VEQ_M}$, GQLfs, Rifs, Glasgow, CFL-Match, DAF
FP ratio, Query processing time	Size of auxiliary data structure, Query processing time
8 query sets for each dataset	
100 query graphs for each query set	
• Q_{iR}(or Q_{iB}): a set of random-walk (or BFS) query graphs with i edges where $i \in \{8,16,32,64\}$	• Q_{iS} (or Q_{iN}): a set of sparse (or non-sparse) query graphs with i vertices where $i \in \{10,20,30,40\}$ or $i \in \{50,100,150,200\}$

	Dataset		Average per graph			
	$\lvert D\rvert$	$\lvert\Sigma\rvert$	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	degree	$\lvert\boldsymbol{\Sigma}\rvert$
PDBS	600	10	2,939	3,064	2.06	6.4
PCM	200	21	377	4,340	23.01	18.9
PPI	20	46	4,942	26,667	10.87	28.5
IMDB	1,500	10	13	66	10.14	6.9
REDDIT	4,999	10	509	595	2.34	10.0
COLLAB	5,000	10	74	2,457	65.97	9.9

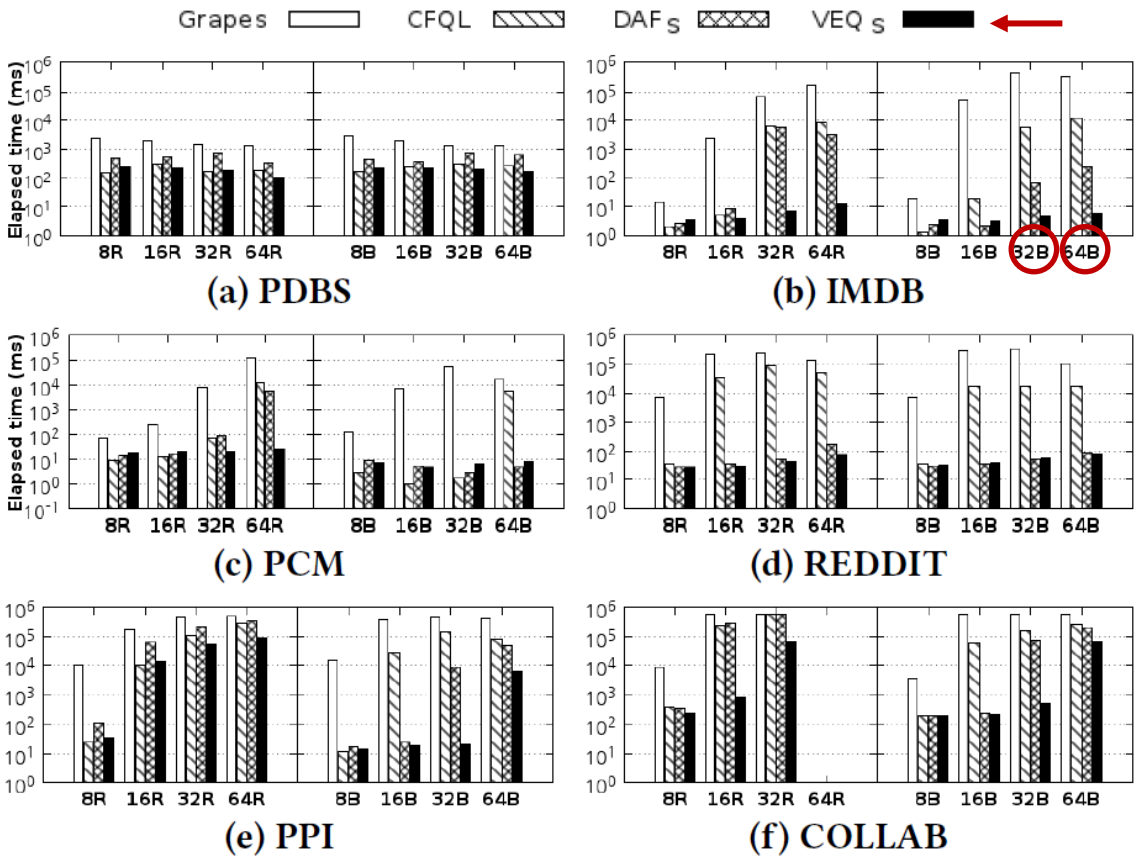
Data graph (G)	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	$\lvert\Sigma\rvert$	Avg degree
Yeast	3,112	12,519	71	8.04
Human	4,674	86,282	44	36.91
HPRD	9,460	37,081	307	7.83
Email	36,692	183,831	20	10.02
DBLP	317,080	1,049,866	20	6.62
YAGO	4,295,825	11,413,472	49,676	5.31

False Positive Ratio of Filtering (Subgraph Search)



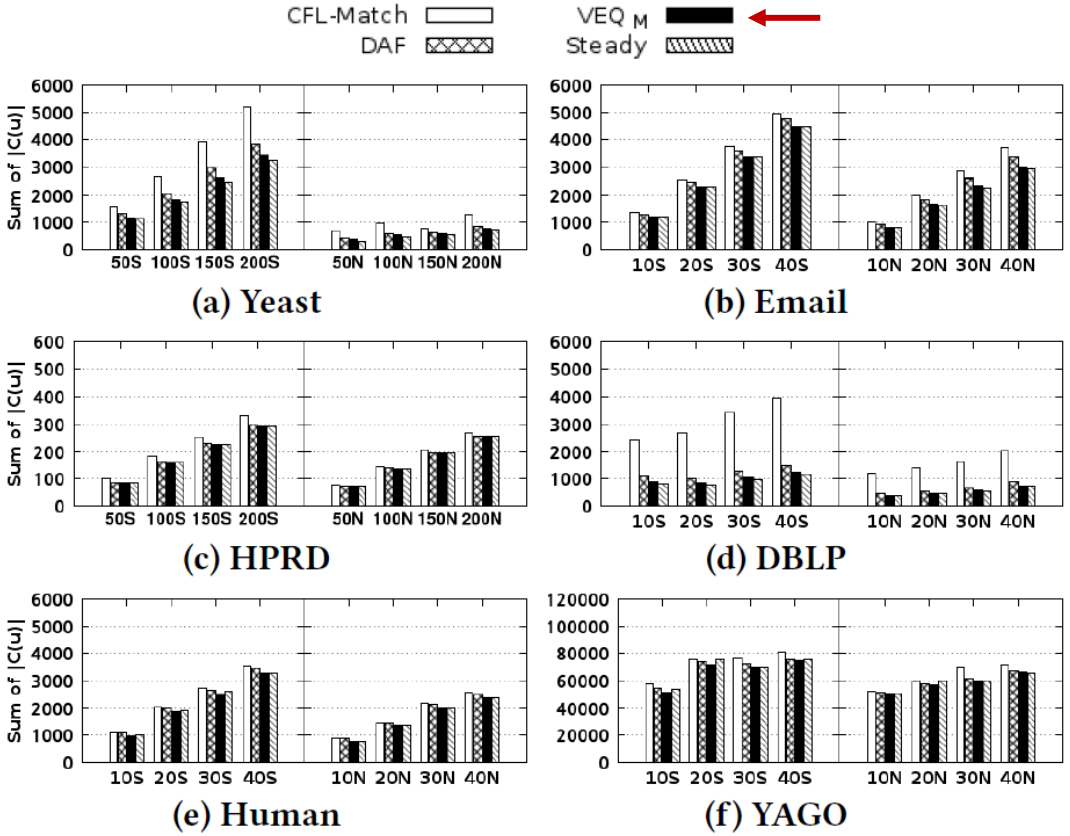
- Lower false positive ratio (better in filtering)
  - VEQ_S < CFQL, Grapes < DAF

Query Processing Time (Subgraph Search)



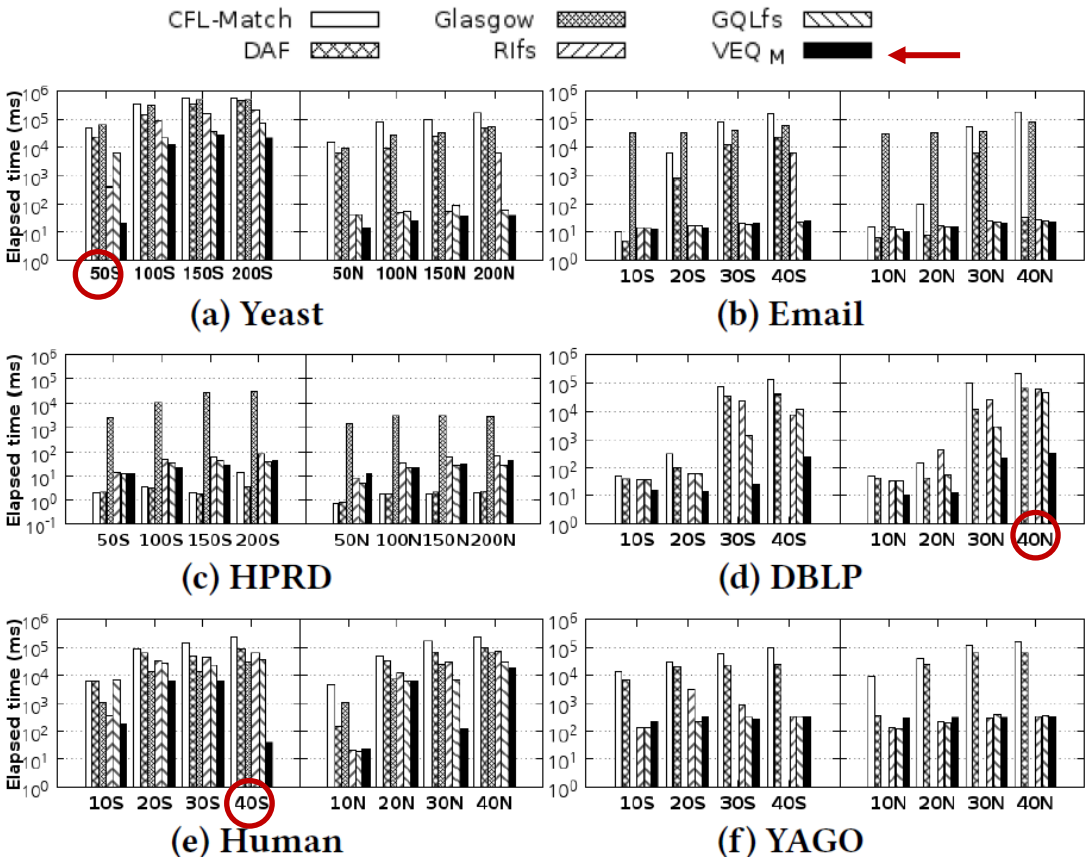
- Faster in query processing
  - > 10^2 × faster than DAF (IMDB, PPI, COLLAB)
  - > 10^3 × faster than CFQL (IMDB, PPI)
  - > 10^5 × faster than Grapes (Q_{32B} and Q_{64B} of IMDB)

Size of Auxiliary Data Structure (Subgraph Matching)



- Size of data structure: $\Sigma_{u \in V(q)} |C(u)|$
  - The smaller the size is, the smaller the search space is.
- More compact (up to 20% less candidates than DAF)
- Comparable to the optimal CS, i.e., Steady

Query Processing Time (Subgraph Matching)



- Faster in query processing than the state-of-the-art methods
  - > 10^2 × faster than GQLfs (Q_{50S} of Yeast, Q_{40S} of Human, Q_{40N} of DBLP)
  - > 10^3 × faster than Rifs (Q_{40S} of Human)

Conclusion

- VEQ
  - **Extended DAG-graph DP**
    - Compute a compact CS = small search space
  - **Matching order based on static equivalence**
    - Consider equivalent query vertices in the matching order
  - **Run-time pruning by dynamic equivalence**
    - Remove equivalent subtrees obtained from equivalent candidate vertices
- Further discussion in performance evaluation of the paper.
  - Sensitivity analysis
  - Ratio of filtering time to verification time
  - Effectiveness of individual techniques