香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Rapids @ HKUST

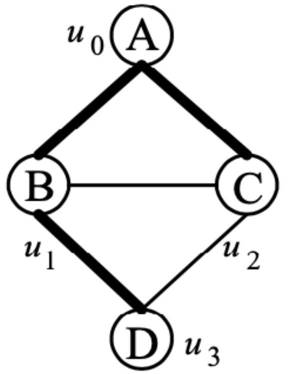
In-Memory Subgraph Matching: An In-depth Study

Shixuan Sun and Qiong Luo

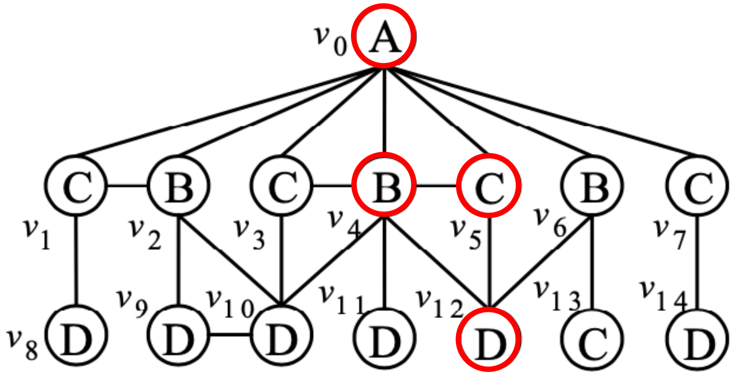
The Hong Kong University of Science and Technology

In-Memory Subgraph Matching

- Subgraph matching finds all subgraphs in a data graph G that are identical to a query graph q.
  - Both q and G are vertex-labeled.
  - q is connected and much smaller than G.
  - G resides in main memory.



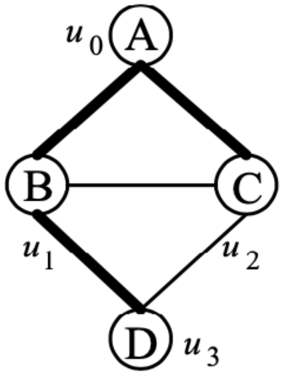
(a) Query graph q.



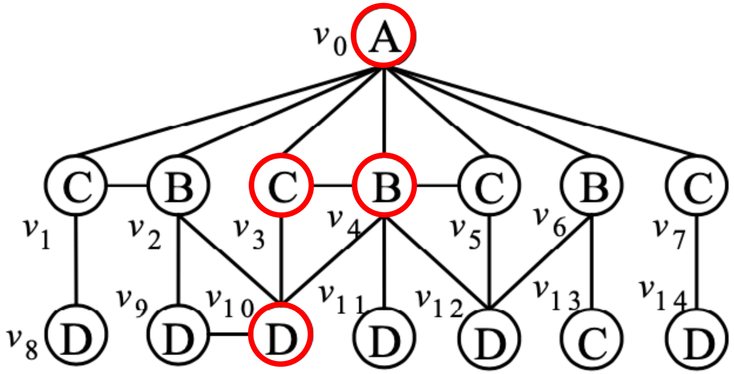
(b) Data graph G.

In-Memory Subgraph Matching

- Subgraph matching finds all subgraphs in a data graph G that are identical to a query graph q.
  - Both q and G are vertex-labeled.
  - q is connected and G is much larger than q.
  - G resides in main memory.



(a) Query graph *q*.



(b) Data graph *G*.

Applications

- Social network analysis.
- Protein interaction understanding.
- Graph database query.

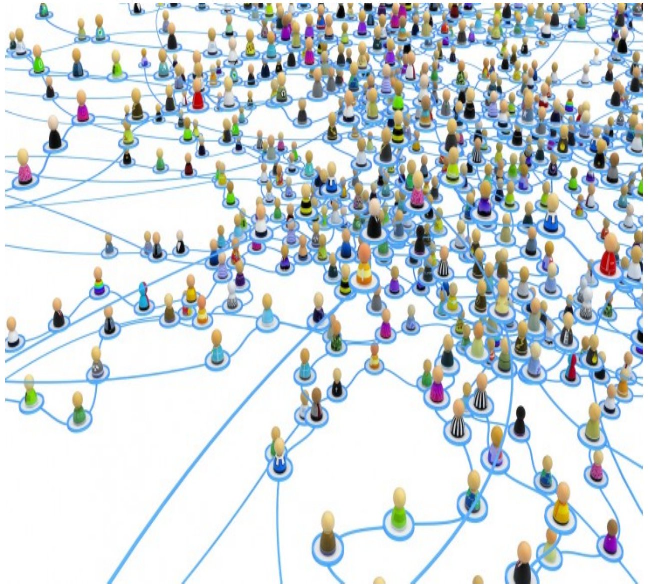
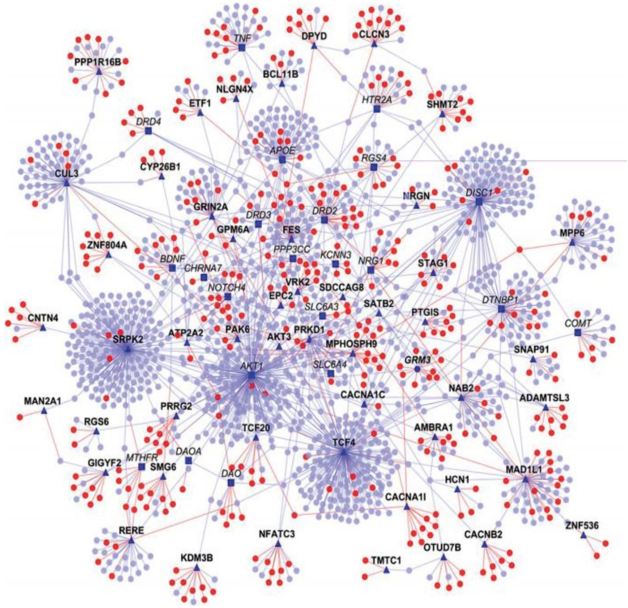
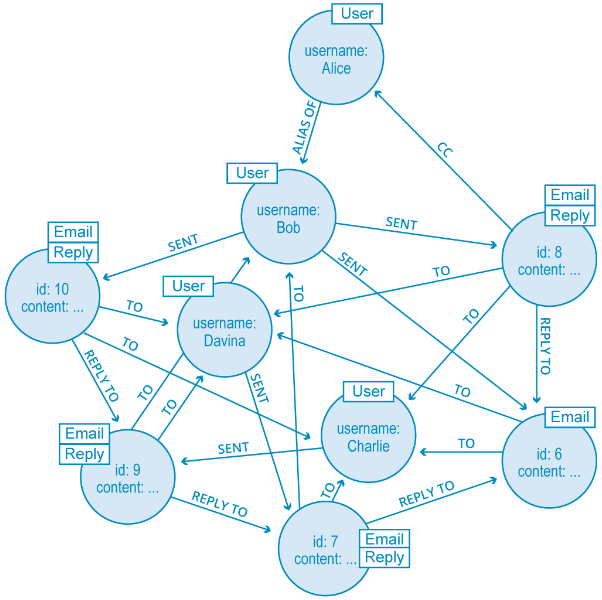




Figure source:
https://thenextweb.com/socialmedia/2013/11/24/facebook-grandparents-need-next-gen-social-network/
https://www.genengnews.com/insights/protein-protein-interactions-get-a-new-groove-on/
https://neo4j.com/blog/graph-theory-predictive-modeling/

Representative Algorithms

Communities	Methodologies	Algorithms
Database	Backtracking Search	QuickSI, GADDI, SPath, GraphQL, TurboIso, BoostIso, CFL, SGMatch, CECI, DP-iso, PGX, PSM, STwig
	Multi-way Join	EmptyHeaded, Graphflow, LogicBlox, PostgreSQL, MonetDB, Neo4j, GpSM
Artificial Intelligence	Backtracking Search	Ullmann, VF2, VF2++, VF3, LAD, Glasgow
Bioinformatics	Backtracking Search	RI, VF2+, Grapes

Representative Algorithms

Communities	Methodologies	Algorithms
Database	Backtracking Search	QuickSI, GADDI, SPath, GraphQL, TurboIso, BoostIso, CFL, SGMatch, CECI, DP-iso, PGX, PSM, STwig
	Multi-way Join	EmptyHeaded, Graphflow, LogicBlox, PostgreSQL, MonetDB, Neo4j, GpSM
Artificial Intelligence	Backtracking Search	Ullmann, VF2, VF2++, VF3, LAD, Glasgow
Bioinformatics	Backtracking Search	RI, VF2+, Grapes

Category of Backtracking-Based Algorithms

- Direct-Enumeration: Directly explore G to find all results.
  - Example algorithms: QuickSI, RI and VF2++.

Category of Backtracking-Based Algorithms

- Direct-Enumeration: Directly explore G to find all results.
  - Example algorithms: QuickSI, RI and VF2++.

- Indexing-Enumeration: Construct indexes on G and answer all queries with the assistance of indexes.
  - Example algorithms: GADDI and SGMatch.

Category of Backtracking-Based Algorithms

- Direct-Enumeration: Directly explore G to find all results.
  - Example algorithms: QuickSI, RI and VF2++.

- Indexing-Enumeration: Construct indexes on G and answer all queries with the assistance of indexes.
  - Example algorithms: GADDI and SGMatch.

- Preprocessing-Enumeration: Generate candidate vertex sets per query at runtime and evaluate the query based on candidate vertex sets.
  - Widely used in the latest algorithms proposed in the database community.
  - Example algorithms: GraphQL, TurboISO, CFL, DP-iso and CECI.

Observation

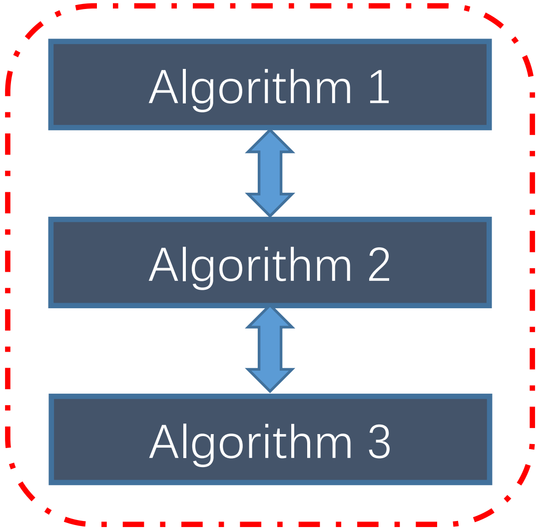
- Techniques in existing algorithms can be classified into several categories each of which have the same goal.
  - Example: Methods filtering candidates, methods optimizing the matching order.

Observation

- Techniques in existing algorithms can be classified into several categories each of which have the same goal.
  - Example: Methods filtering candidates, methods optimizing the matching order.
- The methods are closely related and all affect the evaluation performance.

Observation

- Techniques in existing algorithms can be classified into several categories each of which have the same goal.
  - Example: Methods filtering candidates, methods optimizing the matching order.
- The methods are closely related and all affect the evaluation performance.
- Previous studies regard each algorithm as a black box.
  - Hide effectiveness of individual techniques.



Our Work

- Study individual techniques in the algorithms within a common framework.
  - Compare and analyze individual techniques in existing algorithms.
  - Conduct extensive experiments to evaluate the effectiveness of the techniques.
  - Pinpoint techniques leading to the performance differences and make recommendation.

Our Work

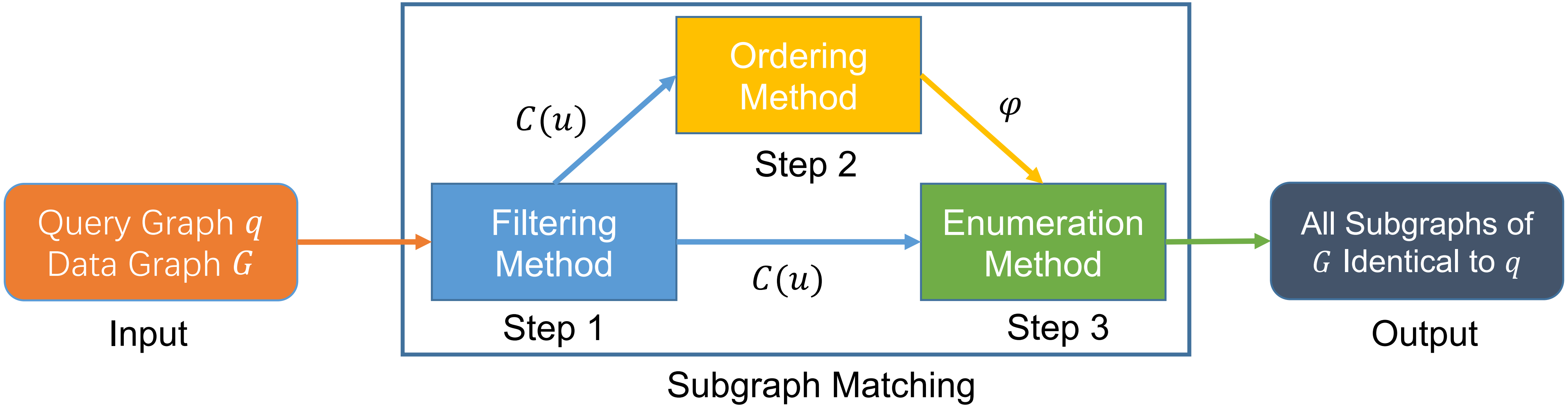
- Study individual techniques in the algorithms within a common framework.
  - Compare and analyze individual techniques in existing algorithms.
  - Conduct extensive experiments to evaluate the effectiveness of the techniques.
  - Pinpoint techniques leading to the performance differences and make recommendation.

- Select seven algorithms from three different communities.
  - GraphQL [SIGMOD'08]
  - CFL [SIGMOD'16]
  - CECI [SIGMOD'19]
  - DP-iso [SIGMOD'19]
  - QuickSI [VLDB'08]
  - RI [BMC Bioinformatics'13]
  - VF2++ [Discrete Applied Mathematics'18]

The preprocessing-enumeration algorithms

The direct-enumeration algorithms

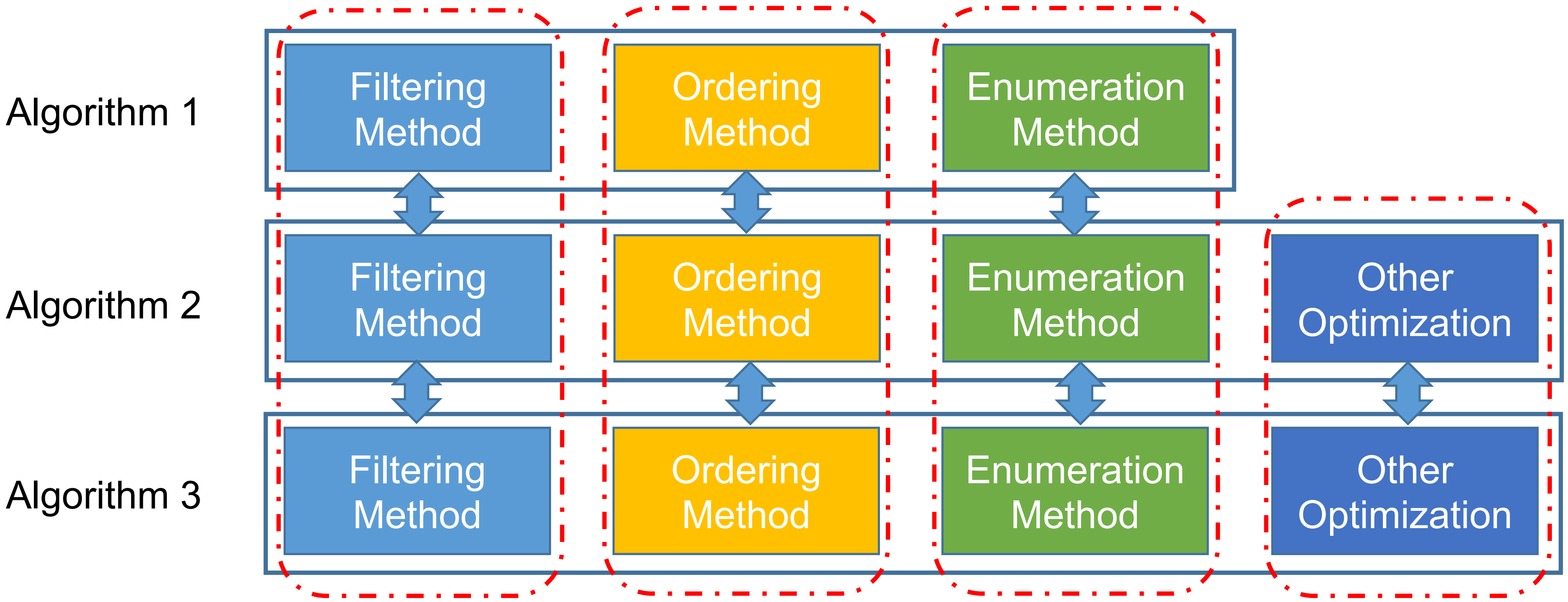
Common Framework



- Filtering Method: Given q and G, minimize candidate vertex sets $C(u)$ for each $u \in V(q)$.
  - $C(u)$: A set of data vertices $v \in V(G)$ that can be mapped to u.
- Ordering Method: Optimize the matching order φ based on the statistics of candidate vertex sets.
  - φ: A sequence of query vertices $V(q)$.
- Enumeration Method: Iteratively extend partial results M by mapping $u \in V(q)$ to $v \in C(u)$ along φ.
  - M: A dictionary storing mappings between query vertices to data vertices.

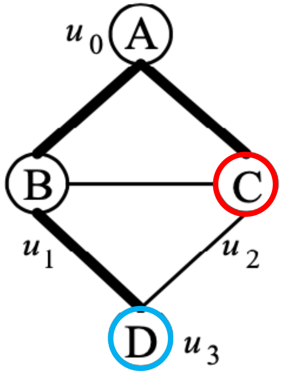
Principles of Our Study

- Study the performance of the algorithms from four aspects.
- When comparing one component, fix the others for fair comparison.

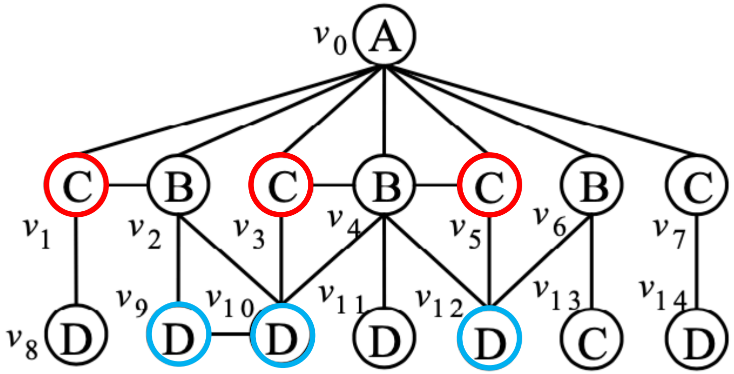


Filtering Method

- Basic Method: Filtering $C(u)$ based on the label $L(u)$ and degree $d(u)$ of u, i.e., $C(u) = \{v \in V(G) | L(v) = L(u) \land d(v) \geq d(u)\}$
  - Take u_2 and u_3 as examples: $C(u_2) = \{v_1, v_3, v_5\}, C(u_3) = \{v_9, v_{10}, v_{12}\}$



(a) Query graph q.

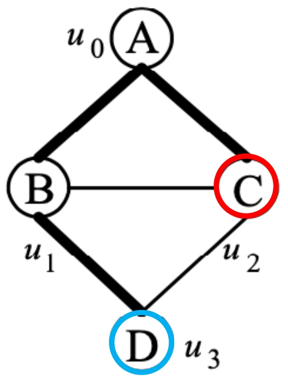


(b) Data graph G.

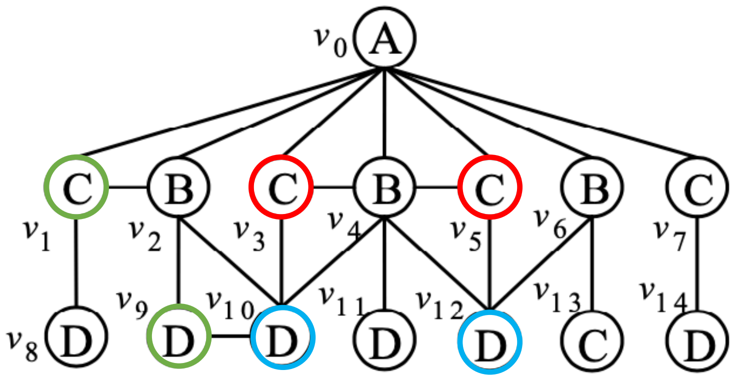
Filtering Method

- Filtering Rule: Given $v \in C(u)$, if there exists $u' \in N(u)$ such that $N(v) \cap C(u') = \emptyset$, then v can be removed from $C(u)$.

- Advanced Method: Filtering $C(u)$ with the rule along a sequence of $u \in V(q)$.
  - Example algorithms: GraphQL, CFL, CECI and DP-iso.
  - Major differences: The filtering sequence and the number of rounds repeated.



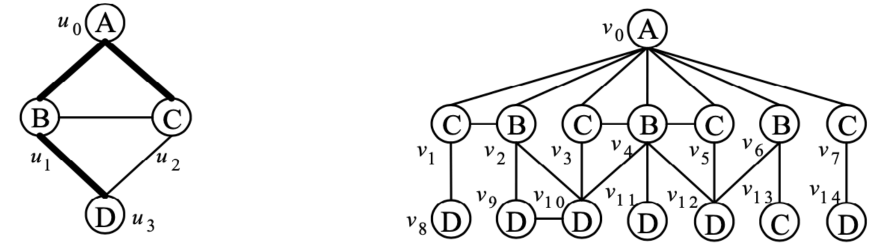
(a) Query graph *q*.



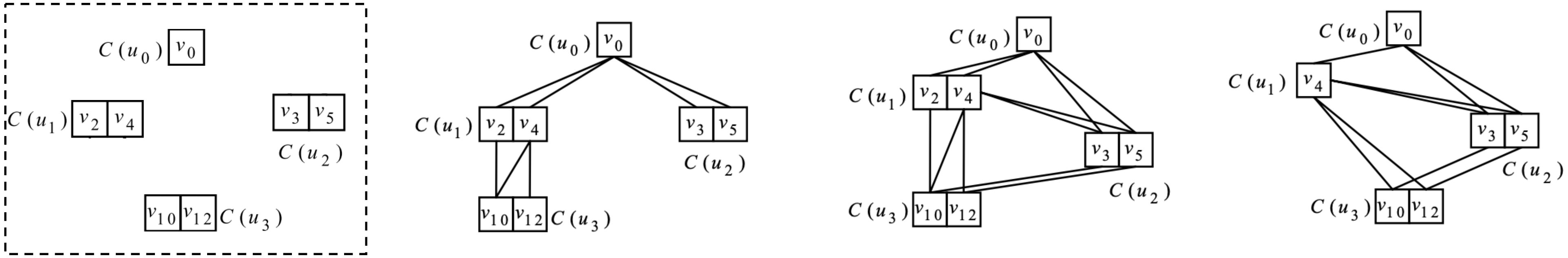
(b) Data graph *G*.

Filtering Method

- Build an auxiliary data structure A to record edges between candidate vertex sets.
  - Serve the cardinality estimation in the ordering method.
  - Accelerate the subsequent enumeration method.



(a) Query graph q. **(b) Data graph G.**



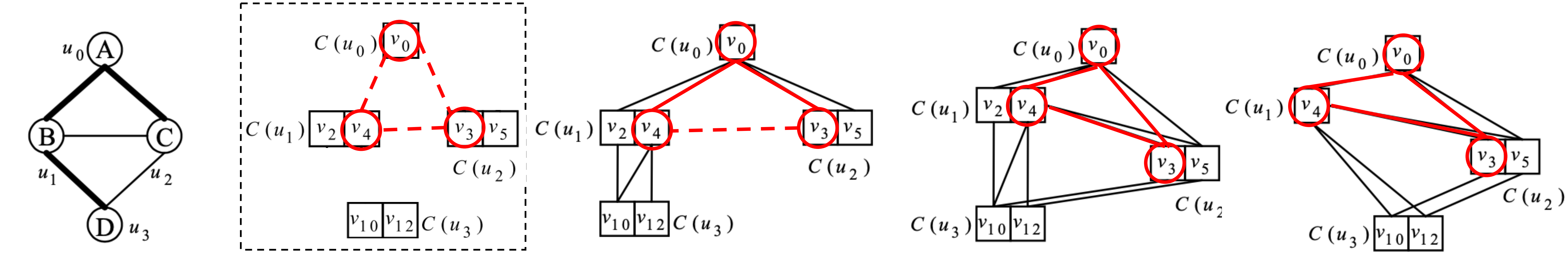
GraphQL A of CFL A of CECI A of DP-iso

Ordering Method

- Adopt the greedy method that (1) selects a start vertex; and (2) iteratively adds unselected query vertices to φ according to the cost estimation based on C and A.
  - The major difference is the cost function.
    - GraphQL: Select the vertex u with the minimum $|C(u)|$ at each step.
    - CFL/DP-iso: Select the path of q with the minimum number of embeddings in A at each step.

Enumeration Method

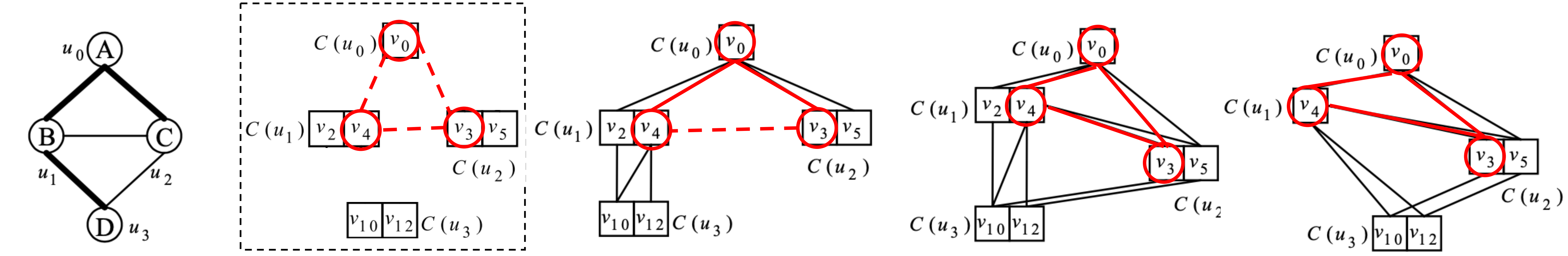
- Extend partial results by mapping $u \in V(q)$ to $v \in C(u)$ along φ with the assistance of A.
  - GraphQL: Probe G for all edge validation.
  - CFL: Probe G and A for the non-tree and tree edge validation, respectively.
  - DP-iso/CECI: Probe A for all edge validation.



Query graph q GraphQL A of CFL A of CECI A of DP-iso

Enumeration Method

- Extend partial results by mapping $u \in V(q)$ to $v \in C(u)$ along φ with the assistance of A.
  - GraphQL: Probe G for all edge validation.
  - CFL: Probe G and A for the non-tree and tree edge validation, respectively.
  - DP-iso/CECI: Probe A for all edge validation.



Query graph q GraphQL A of CFL A of CECI A of DP-iso

Recommendation: Use the DP-iso/CECI-style auxiliary data structure and enumeration method.

Optimization Method

- Failing set pruning: During the enumeration, utilize the information obtained from the explored part of the search tree to prune invalid partial results.
  - Proposed by DP-iso.
  - Other algorithms can adopt the optimization as well.

Experimental Setup

- All algorithms are implemented in C++ and run on a machine with 2.3GHz CPUs and 128GB RAM.
- Real-world data graphs:

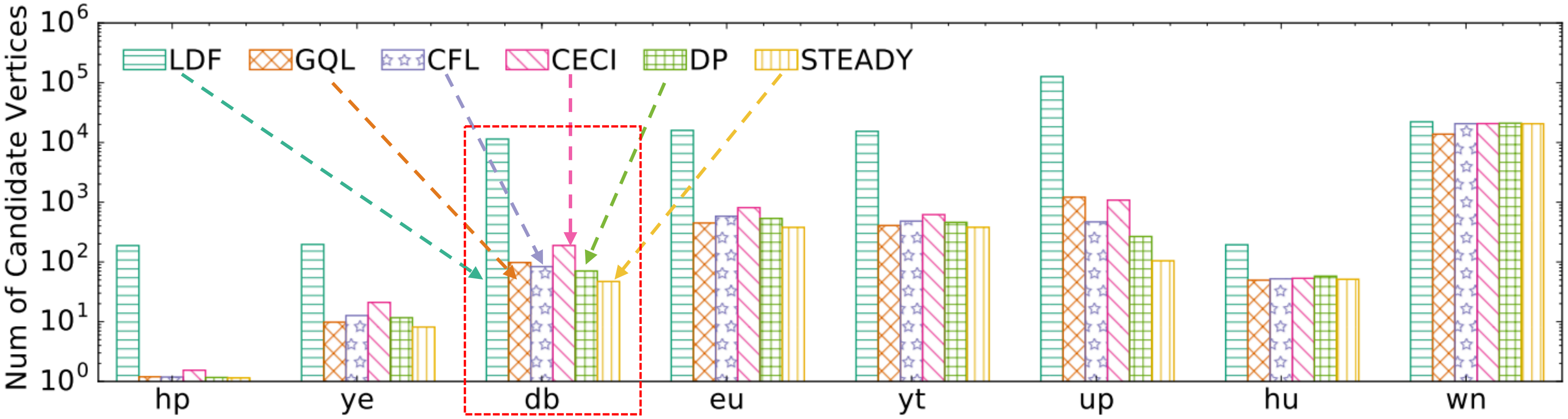
Category	Dataset	Name	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert \Sigma \rvert$	d
Biology	Yeast	*ye*	3,112	12,519	71	8.0
	Human	*hu*	4,674	86,282	44	36.9
	HPRD	*hp*	9,460	34,998	307	7.4
Lexical	WordNet	*wn*	76,853	120,399	5	3.1
Citation	US Patents	*up*	3,774,768	16,518,947	20	8.8
Social	Youtube	*yt*	1,134,890	2,987,624	25	5.3
	DBLP	*db*	317,080	1,049,866	15	6.6
Web	eu2005	*eu*	862,664	16,138,468	40	37.4

- Query sets:
  - Query graphs are randomly extracted from the data graph.
  - Each query set contains 200 connected graphs with the same number of vertices.
  - Q_{iD} and Q_{iS} denote dense $(d(q) \geq 3)$ and sparse $(d(q) < 3)$ query sets containing graphs with i vertices.
  - Each data graph has 1800 queries in total.

Dataset	Query Set	Default
Yeast, HPRD, US Patents, Youtube, DBLP, eu2005	Q_4, Q_{8D}, Q_{16D}, Q_{24D}, Q_{32D}, Q_{8S}, Q_{16S}, Q_{24S}, Q_{32S}	Q_{32D}, Q_{32S}
Human, WordNet	Q_4, Q_{8D}, Q_{12D}, Q_{16D}, Q_{20D}, Q_{8S}, Q_{12S}, Q_{16S}, Q_{20S}	Q_{20D}, Q_{20S}

Effectiveness of Filtering Methods

- **Metrics:** Num of Candidate Vertices = $\frac{1}{|Q|}\sum_{q\in Q}\frac{1}{|V(q)|}\sum_{u\in V(q)}|C(u)|$.
- **Finding:** GraphQL, CFL and DP-iso are competitive with each other, and they are close to STEADY.

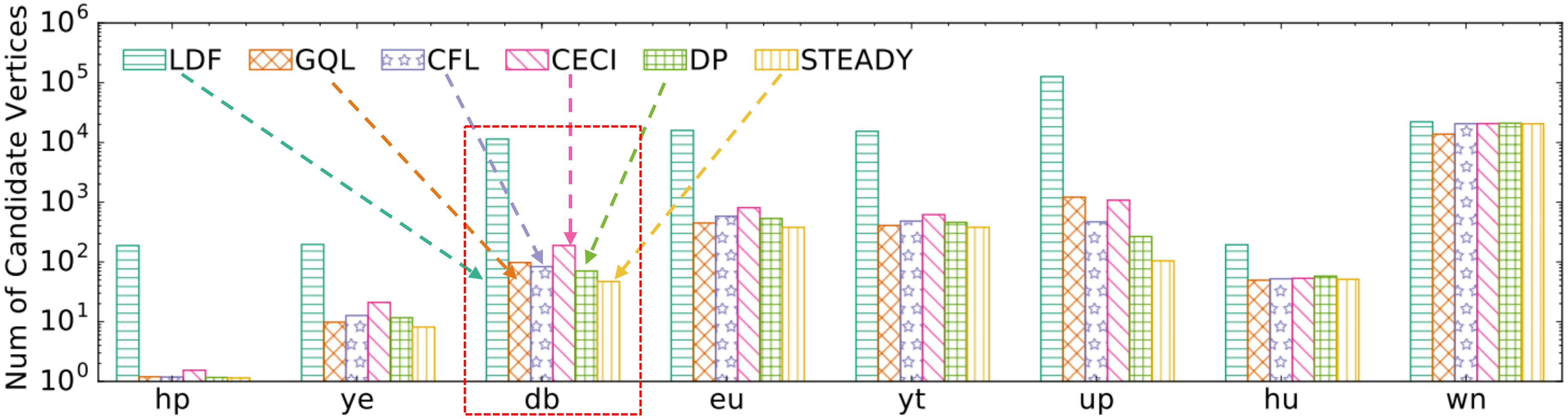


Varying datasets on dense query sets.

LDF: Label and degree filter.
GQL: Filtering of GraphQL.
CFL: Filtering of CFL.
CECI: Filtering of CECI.
DP: Filtering of DP-iso.
STEADY: Given $v \in C(u)$, it satisfies that $\forall u' \in N(u), N(v) \cap C(u') \neq \emptyset$.

Effectiveness of Filtering Methods

- **Metrics:** Num of Candidate Vertices = $\frac{1}{|Q|}\sum_{q\in Q}\frac{1}{|V(q)|}\sum_{u\in V(q)}|C(u)|$.
- **Finding:** GraphQL, CFL and DP-iso are competitive with each other, and they are close to STEADY.
- **Recommendation:** Adopt the filtering method of GraphQL/CFL/DP-iso to prune candidate vertex sets.

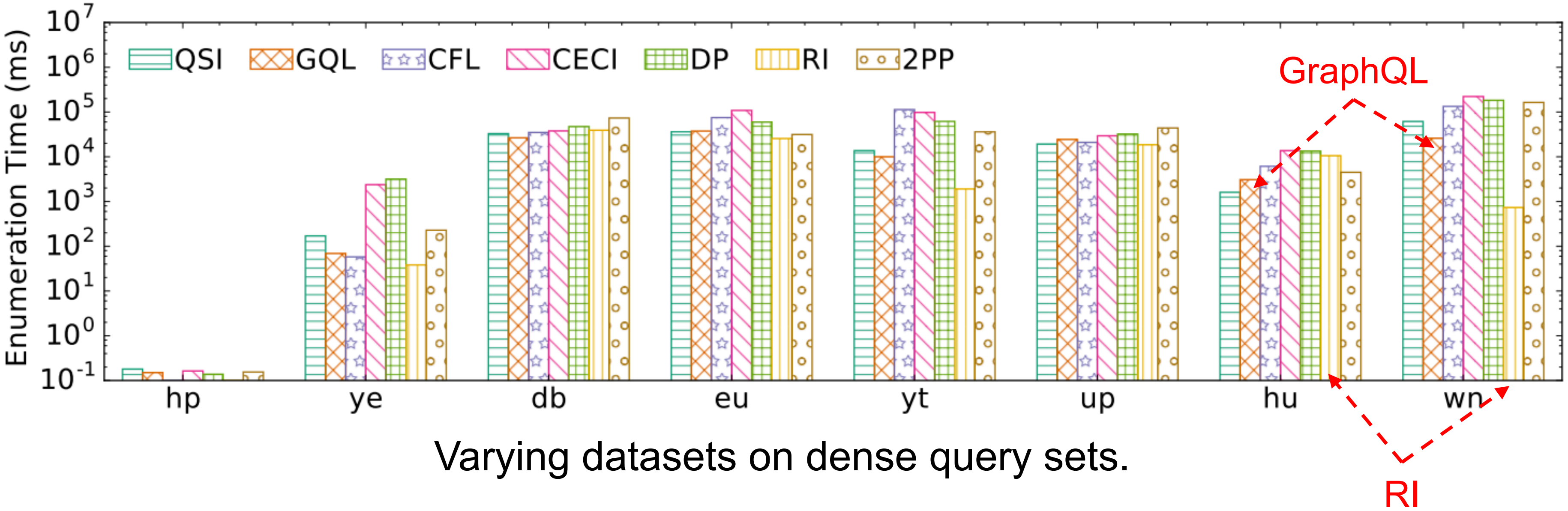


Varying datasets on dense query sets.

LDF: Label and degree filter.
GQL: Filtering of GraphQL.
CFL: Filtering of CFL.
CECI: Filtering of CECI.
DP: Filtering of DP-iso.
STEADY: Given $v \in C(u)$, it satisfies that $\forall u' \in N(u), N(v) \cap C(u') \neq \emptyset$.

Effectiveness of Ordering Methods

- **Setup:** Use the DP-iso/CECI-style auxiliary data structure and enumeration method and adopt candidate vertex sets of GraphQL.
- **Metrics:** Enumeration Time = $\frac{1}{|Q|}\sum_{q \in Q} T(A, q)$.
- **Finding:** GraphQL and RI are usually the most effective among competing methods.

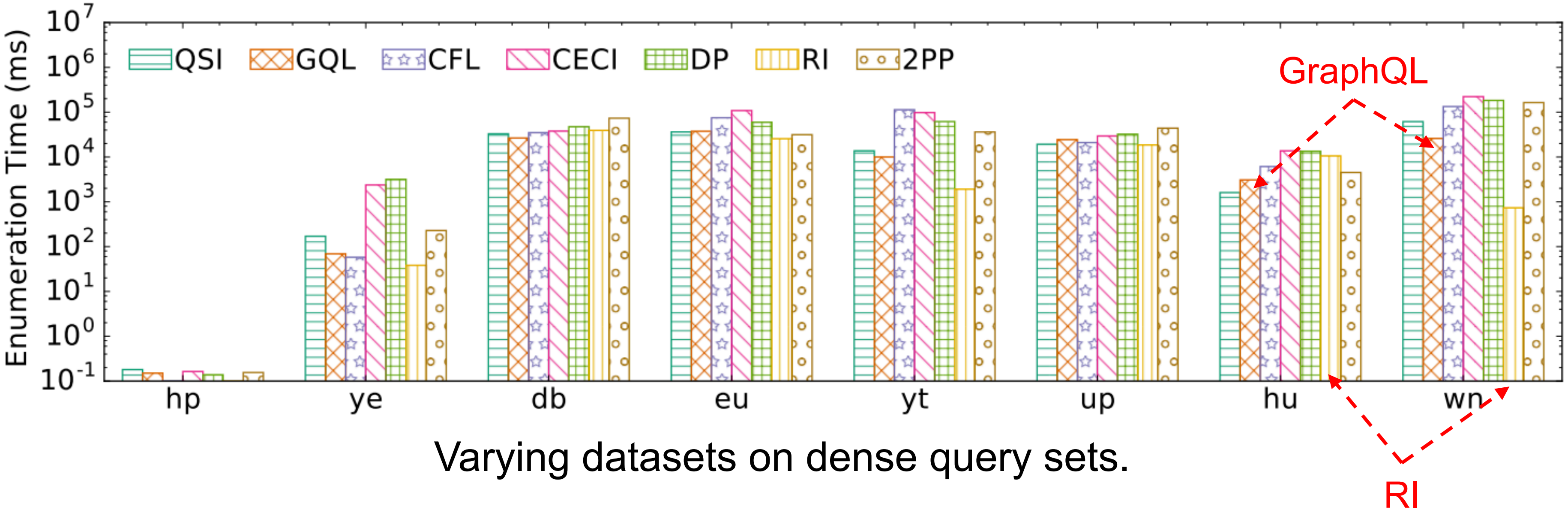


Varying datasets on dense query sets.

QSI: Ordering of QuickSI.
GQL: Ordering of GraphQL.
CFL: Ordering of CFL.
CECI: Ordering of CECI.
DP: Ordering of DP-iso.
RI: Ordering of RI.
2PP: Ordering of VF2++.

Effectiveness of Ordering Methods

- **Setup:** Use the DP-iso/CECI-style auxiliary data structure and enumeration method and adopt candidate vertex sets of GraphQL.
- **Metrics:** Enumeration Time = $\frac{1}{|Q|}\sum_{q \in Q} T(A, q)$.
- **Finding:** GraphQL and RI are usually the most effective among competing methods.
- **Recommendation: Adopt GraphQL and RI on dense and sparse data graphs respectively.**



Varying datasets on dense query sets.

QSI: Ordering of QuickSI.
GQL: Ordering of GraphQL.
CFL: Ordering of CFL.
CECI: Ordering of CECI.
DP: Ordering of DP-iso.
RI: Ordering of RI.
2PP: Ordering of VF2++.

Effectiveness of Failing Set Pruning

- **Setup:** Continue with the experiments on ordering methods and enable the failing set pruning.
- **Metrics:** Count the number of unsolved queries within 5 minutes.
- **Finding:** (1) Failing set pruning can significantly reduce the number of unsolved queries; and (2) all competing algorithms can generate ineffective matching orders.

Algorithm	*yt*		*up*		*hu*		*wn*	
	wo/fs	w/fs	wo/fs	w/fs	wo/fs	w/fs	wo/fs	w/fs
QSI	14	0	26	9	12	6	69	20
GQL	11	0	23	8	10	2	17	3
CFL	95	6	24	12	16	8	191	139
CECI	161	5	39	7	40	9	547	351
DP	70	6	40	13	30	20	307	221
RI	2	0	18	8	23	9	0	0
2PP	49	3	49	17	12	7	270	220
Fail-All	**0**	**0**	**7**	**3**	**2**	**0**	**0**	**0**

Number of unsolved queries among 1800 queries for each data graph.

wo/fs: Enumeration without the failing set pruning.
w/fs: Enumeration with the failing set pruning.
Fail-ALL: Number of queries that no competing algorithms can complete within 5 minutes.

Effectiveness of Failing Set Pruning

- **Setup:** Continue with the experiments on ordering methods and enable the failing set pruning.
- **Metrics:** Count the number of unsolved queries within 5 minutes.
- **Finding:** (1) Failing set pruning can significantly reduce the number of unsolved queries; and (2) all competing algorithms can generate ineffective matching orders.
- **Recommendation:** Enable failing set pruning for large queries.

Algorithm	*yt*		*up*		*hu*		*wn*	
	wo/fs	w/fs	wo/fs	w/fs	wo/fs	w/fs	wo/fs	w/fs
QSI	14	0	26	9	12	6	69	20
GQL	11	0	23	8	10	2	17	3
CFL	95	6	24	12	16	8	191	139
CECI	161	5	39	7	40	9	547	351
DP	70	6	40	13	30	20	307	221
RI	2	0	18	8	23	9	0	0
2PP	49	3	49	17	12	7	270	220
Fail-All	**0**	**0**	**7**	**3**	**2**	**0**	**0**	**0**

Number of unsolved queries among 1800 queries for each data graph.

wo/fs: Enumeration without the failing set pruning.
w/fs: Enumeration with the failing set pruning.
Fail-ALL: Number of queries that no competing algorithms can complete within 5 minutes.

Conclusion

- Compare and analyze individual techniques in seven algorithms from three communities within a common framework.
- Conduct extensive experiments to evaluate the effectiveness of each kind of methods respectively.
- Report our new findings and make the recommendation through experiments and analysis.

Checkout source code and datasets at: github.com/RapidsAtHKUST/SubgraphMatching