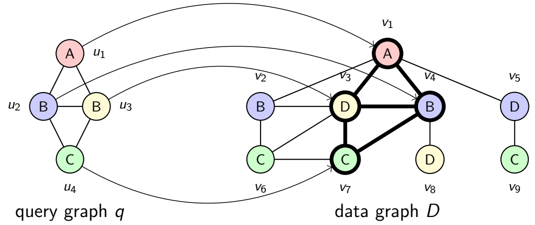
BICE: Exploring Compact Search Space by Using Bipartite Matching and Cell-Wide Verification

Yunyoung Choi (Alsemy);
Kunsoo Park (Seoul National University);
Hyunjoon Kim (Hanyang University)

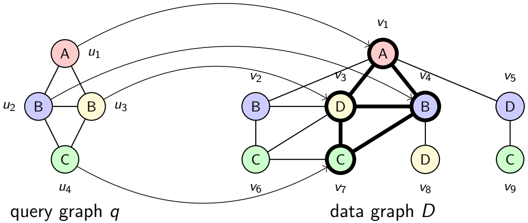
Table of Contents

Embedding



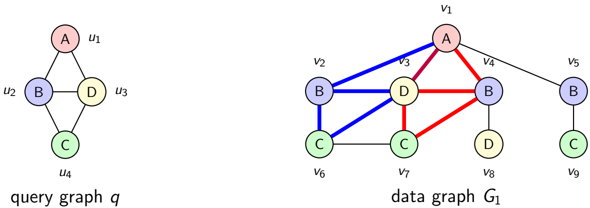
- Given a query graph $q = (V(q), E(q), l_q)$ and a data graph $G = (V(G), E(G), l_G)$
- **Embedding** of q in G is a mapping $M : V(q) \to V(G)$ such that:
  - M is injective.(i.e. $M(u) \neq M(u')$ for $u \neq u'$,
  - $L_q(u) = L_G(M(u))$ for every $u \in V(q)$,
  - $(M(u), M(u')) \in E(G)$ for every $(u, u') \in E(q)$

Embedding



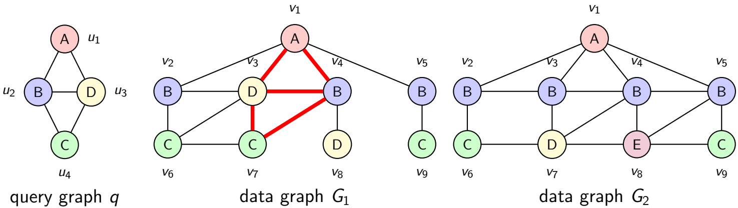
- Embedding.
  - e.g. $M = \{(u_1, v_1), (u_2, v_4), (u_3, v_3), (u_4, v_7)\}$
- An embedding of an induced subgraph of q is a **partial embedding**.
  - e.g. $M = \{(u_1, v_1), (u_2, v_3), (u_3, v_3)\}$

Subgraph Matching



- Given a query graph q and data graph G.
- Subgraph Matching
  - Find all embeddings of q in G (NP-hard).
- Two embeddings
  - $M_1 = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_6), \}$
  - $M_2 = \{(u_1, v_1), (u_2, v_4), (u_3, v_3), (u_4, v_7), \}$

Subgraph Search

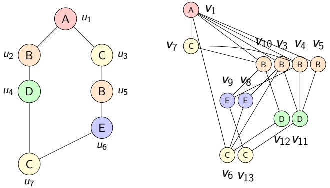


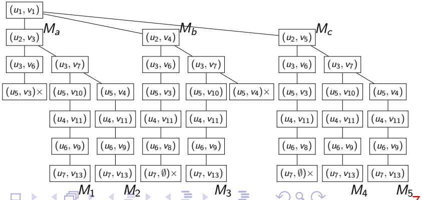
- Given a query graph q and a set of data graphs $D = \{G_1, G_2, ..., G_m\}$
- Subgraph Search
  - Find all the data graphs in D that contains q as subgraphs (NP-hard)
  - $A_q = \{G \in D | q \subset G\}$.
  - $A_q = \{G_1\}$

Table of Contents

Related Work

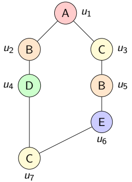
- Subgraph search
  - Preprocessing-enumeration
    - CFQL [ICDE 2019]
    - VEQ [SIGMOD 2021, VLDB Journal 2022]
- Subgraph matching
  - Preprocessing-enumeration
    - Turbo$_{iso}$ [SIGMOD 2013]
    - CFL-Match [SIGMOD 2019]
    - DAF [SIGMOD 2019]
    - GQLfs [SIGMOD 2020]
  - Direct enumeration
    - RIfs [SIGMOD 2020]
  - Contraint programming
    - Glasgow [ICGT 2020]
- Challenges
  - Redundant computations in search.

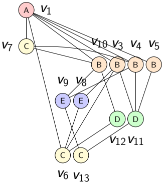


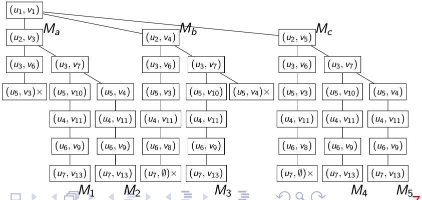


Related Work

- Subgraph search
  - Preprocessing-enumeration
    - CFQL [ICDE 2019]
    - VEQ [SIGMOD 2021, VLDB Journal 2022]
    - BICE [VLDB 2023] → ours
- Subgraph matching
  - Preprocessing-enumeration
    - Turbo$_{\text{iso}}$ [SIGMOD 2013]
    - CFL-Match [SIGMOD 2019]
    - DAF [SIGMOD 2019]
    - GQLfs [SIGMOD 2020]
    - BICE [VLDB 2023] → ours
  - Direct enumeration
    - RIfs [SIGMOD 2020]
  - Contraint programming
    - Glasgow [ICGT 2020]
- Challenges
  - Redundant computations in search.



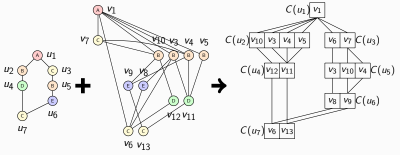




Framework

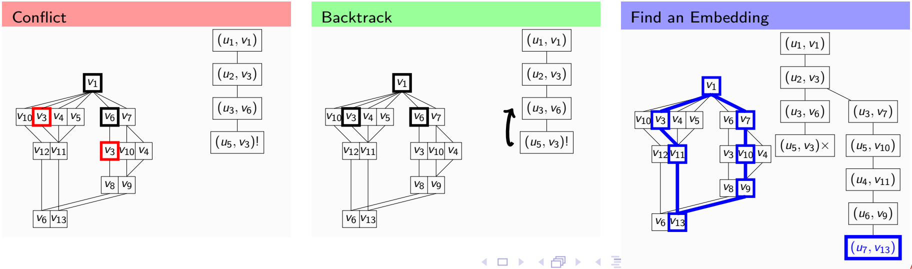
- Framework
  1. **Preprocessing**. Adopt a filtering process to find a **candidate set** $C(u)$ for each $u \in V(q)$, where $C(u)$ is a subset of $V(G)$ which u can be mapped to.
     - For subgraph search, if there is $u \in V(q)$ such that $C(u) = \emptyset$, we skip to search.
  2. **Enumeration**. Choose a linear order of the query vertices, called **matching order**, and apply backtracking based on matching order.

Preprocessing



Framework

- Framework
  1. **Preprocessing**. 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.
  2. **Enumeration**. Choose a linear order of the query vertices, called **matching order**, and apply backtracking based on matching order.
     - Iteratively, map candidate to query vertex

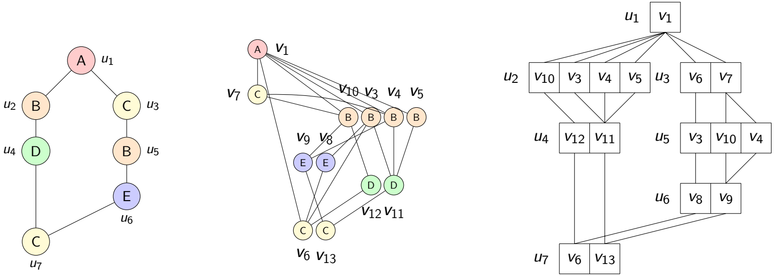


Framework

- Framework
  1. **Preprocessing**. 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.
  2. **Enumeration**. Choose a linear order of the query vertices, called **matching order**, and apply backtracking based on matching order.
     - Three new techniques are applied in the enumeration stage.
     - Pruning by bipartite matching, Pruning by Failing sets with bipartite matching, and Cell-wide verification.

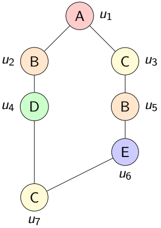
Table of Contents

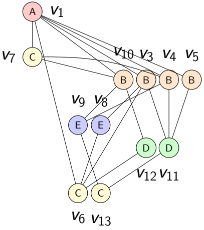
Candidate Space (CS)

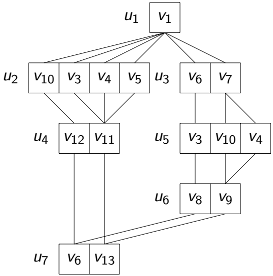


- Candidate Space (CS) on q and G consists of the candidates set $C(u)$ for each $u \in V(q)$, and between candidates.
  - For each $u \in V(q)$ there is a candidate set $C(u)$, which is a set of vertices in G that u can be mapped to (e.g. $C(u_2) = \{v_2, v_4\}$)
  - There is an edge btw. $v \in C(u)$ and $v' \in C(u')$ iff $(u, u') \in E(q)$ and $(v, v') \in E(G)$.

Candidate Space (CS)



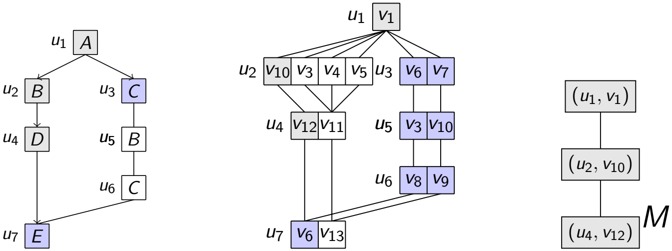




- How do we get compact CS?
  - Extended DAG-Grpah DP [Kim et al., SIGMOD 2021]

Table of Contents

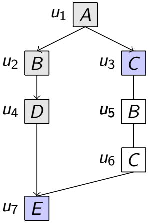
Backtracking Framework



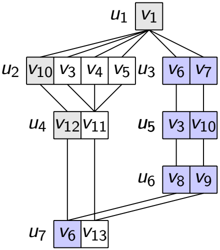
(a) query graph (b) candidate space (c) search tree

- 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 of **extendable candidates** u regarding M
  - If there are no mapped neighbors of u, $C_M(u) = C(u)$.
  - Otherwise, $C_M(u)$ is the set of vertices $v \in C(u)$ adjacent to $M(n_i)$ in CS for every mapped neighbor n_i of u.

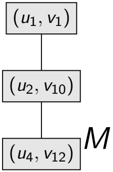
Backtracking Framework



(a) query graph



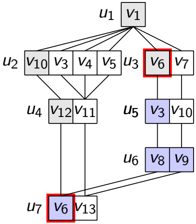
(b) candidate space



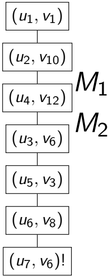
(c) search tree

- Given a partial embedding $M = \{(u_1, v_1), (u_2, v_{10}), (u_4, v_{12})\}$
  - u_1, u_2, u_4 are mapped vertices
  - u_3, u_7 are extendable vertices
  - $C_M(u_7) = \{v_6\}$, $C_M(u_5) = \{v_3, v_{10}\}$

Observation



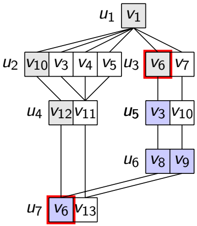
(a) Extendable candidates of M_2



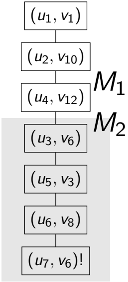
(b) Search tree

- M_2 will end up with a mapping conflict $(u_7, v_6)!$ between (u_3, v_6) and (u_7, v_6).

Observation



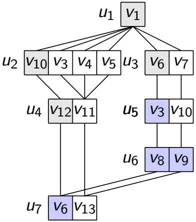
(a) Extendable candidates of M_2



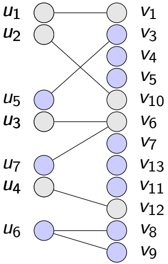
(b) Search tree

- M_2 will end up with a mapping conflict $(u_7, v_6)!$ between (u_3, v_6) and (u_7, v_6).
- Extending embedding M_2 could cause huge redundant search space

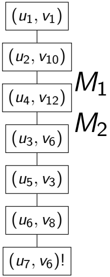
Candidate Bipartite Graph



(a) Extendable candidates of M_2



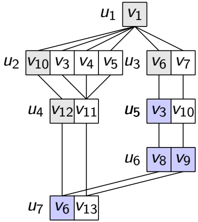
(b) Candidate bipartite graph of M_2.



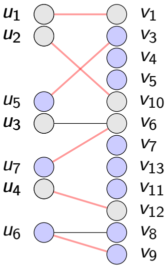
(c) Search tree

- Given a partial embedding M, define candidate bipartite graph B_M.
  - Given a query graph q and a data graph G, $V(B_M) = V(q) \cup V(G)$.
  - There is an edge $(u, M(u))$ if u is mapped in M; there is an edge between $u \in V(q)$ and every $v \in C_M(u)$ otherwise.

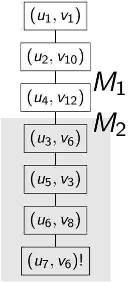
Candidate Bipartite Graph



(a) Extendable candidates of M_2



(b) Candidate bipartite graph of M_2.

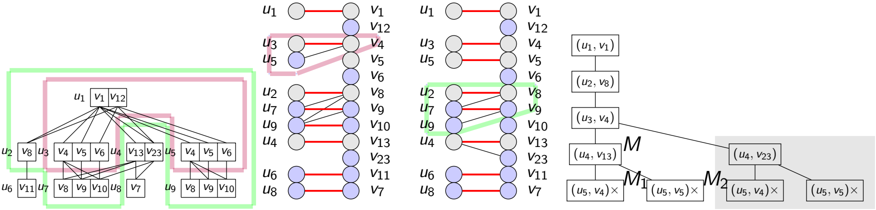


(c) Search tree

- **Lemma.** a maximum bipartite matching H in B_M, partial embedding M is redundant if $|H| < |V(q)|$.
- **e.g.** M_2 is pruned out.

Table of Contents

Pruning by Failing Sets with Bipartite Matching

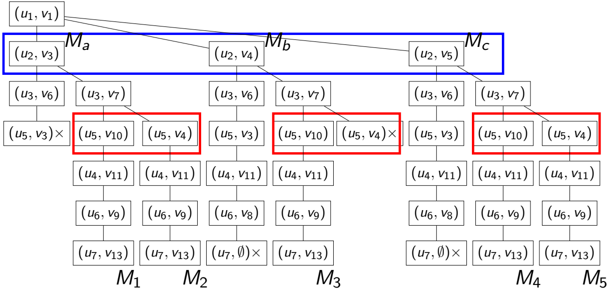


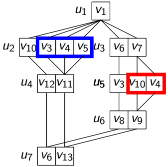
(a) Candidate space (b) C_{M_1} (c) C_{M_2} (d) Search tree

- u_4 was not relevant to any failures (M_1 and M_2).
  - no matter now we change the mapping of u_4, an extension will end up with failure.
- Define the set of vertices that is relevant to failures F_M
  - $F_M = \{u_1, u_2, u_3, u_5, u_7, u_9\}$.
  - Since $u_4 \notin F_M$, a sibling node of (u_4, u_{13}) is redundant.

Table of Contents

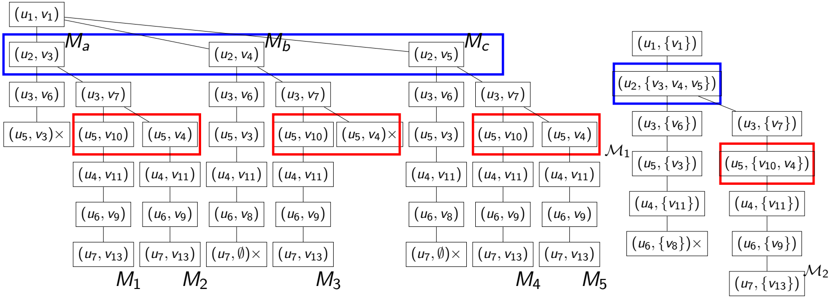
Observation





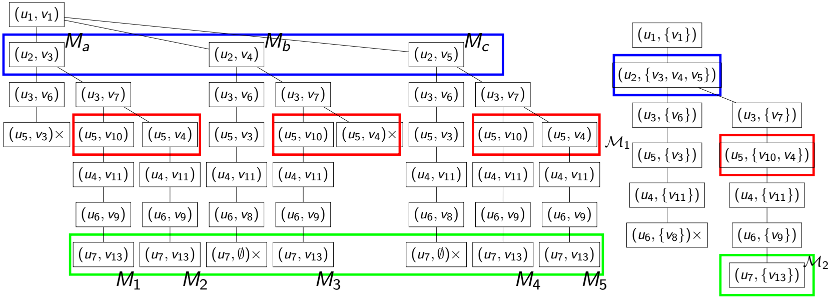
- *cell* $\gamma(u, v)$ is a subset of $C(u)$ such that:
  - $w \in \gamma(u, v)$ if and only if v and w have the same set of neighbors in CS.
  - **e.g.** Cell $\gamma(u_2, v_3) = \{v_3, v_4, v_5\}$, $\gamma(u_5, v_4) = \{v_{10}, v_4\}$.
- the similar subtrees are generated regardless of which candidate in the cell is mapped in backtracking.

Cell-Wide Verification



- Definition (Hypermapping). $(\mathcal{M} : V(q) \to \{\gamma(u, v) | v \in C_{\mathcal{M}}(u)\})$
  - a hypermapping of an induced subgraph $q[S]$ is called a partial hypermapping.
  - **e.g.** $\mathcal{M}_1(u_2) = \{v_3, v_4, v_5\}$
- Search space with hyper mapping is more compact.

Cell-Wide Verification



- $\Pi_{u \in V(q)} \{(u, v) \mid v \in \mathcal{M}(u)\}$ is the set of homomorphisms of q in G.
- injective mappings in $\Pi_{u \in V(q)} \{(u, v) \mid v \in \mathcal{M}(u)\}$ are embeddings.
  - **e.g.** injective mappings in $\Pi_{u \in V(q)} \{(u, v_i) \mid v_i \in \mathcal{M}_2(u)\}$ are M_1, M_2, M_3, M_4 and M_5.

Table of Contents

Experiment

<table>
<tr><td>Subgraph Search</td><td>Subgraph Matching</td></tr>
<tr><td>$\mathrm{BICE_S}, \mathrm{VEQ_S}, \mathit{CFQL}$</td><td>$\mathrm{BICE_M}, \mathrm{VEQ_M}, \mathrm{GQL}, \mathrm{RapidMatch}, \mathit{RLFs}$</td></tr>
<tr><td colspan="2">8 query sets for each dataset</td></tr>
<tr><td colspan="2">100 query graphs for each query set</td></tr>
<tr><td colspan="2">Query Processing time</td></tr>
<tr><td>Q_{iR}(or Q_{iB}) : a set of random-walk (or BFS) query graphs with i edges where $i \in \{8, 16, 32, 64\}$.</td><td>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 $i \in \{50, 150, 150, 200\}$.</td></tr>
</table>

G	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	avg deg	$\lvert\Sigma\rvert$
Berkstan	685,230	6,649,470	19.41	20
DBLP	317,080	1,049,866	6.62	20
Google	875,713	4,332,051	9.87	20
Human	4,674	86,282	36.91	44
Patents	3,774,768	16,518,948	8.75	20
Yeast	3,112	12,519	8.04	71
Twitter	41,652,230	1,468,364,884	70.51	1,000

Table: Data sets for subgraph matching

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

Table: Data sets for subgraph search

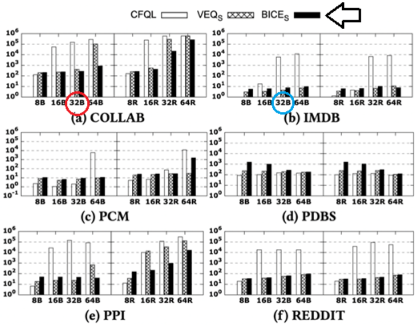
Compression Power of Hypermapping

	Patents							
Query	50S	100S	150S	200S	50N	100N	150N	200N
Ratio	15.45	15.16	12.66	3.25	11.64	12.76	6.61	231.41
	COLLAB							
Query	8B	16B	32B	64B	8R	16R	32R	64R
Ratio	1.00	1.00	1.49	52.26	1.07	100.33	352.69	—

- Average number of partial embeddings covered by one partial hypermapping generated by cell-wide verification.
- A larger ratio is better.
- The ratio increases as the size of a query graph grows

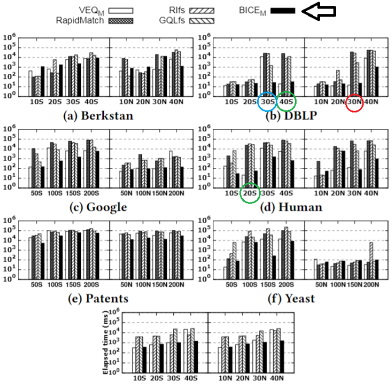
Query Processing Time(Subgraph Search)

- $> 10^2x$ faster than VEQ_s (COLLAB 32R)
- $> 10^3x$ faster than $CFQL$ (IMDB 64B)



Query Processing Time (Subgraph Matching)

- $> 10^3 x$ faster than RIfs and RapidMatch (DBLP 30N)
- $> 10^2 x$ faster than GQL (DBLP 40S and Human 20S)
- $> 10^2 x$ faster than VEQ_M (30S DBLP)



(a) Berkstan (b) DBLP (c) Google (d) Human (e) Patents (f) Yeast (g) Twitter

Conclusion

- BICE
    - Three techniques
        - Pruning by bipartite matching
        - Pruning by failing set with bipartite matching
        - Cell-wide verification
- Further discussion in performance evaluation of the paper.
    - Sensitivity analysis
    - Effective of individual techniques