Subgraph Search & Subgraph Matching

20231218

What is subgraph search?

- **Subgraph query processing (also known as subgraph search).**
- Given a query graph q and a set D of data graphs, the subgraph search problem is to find all data graphs in D that contains q as subgraphs. That is, subgraph search is to compute the answer set $A_q = \{G \in D | q \subseteq G\}$.

Related Problems

- **Subgraph Matching.**
- Given a query graph q and a data graph G, the subgraph matching problem is to find all embeddings of q in G.

- Subgraph matching and subgraph search are closely related.

- **Subgraph isomorphism** (i.e., "Does G contain a subgraph isomorphic to q?") is NP-complete.
- Subgraph matching and subgraph search are NP-hard.

Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching

Hyunjoon Kim
Seoul National University
SAP Labs Korea
hjkim@theory.snu.ac.kr

Yunyoung Choi
Seoul National University
yychoi@theory.snu.ac.kr

Kunsoo Park*
Seoul National University
kpark@theory.snu.ac.kr

Xuemin Lin
University of New South Wales
lxue@cse.unsw.edu.au

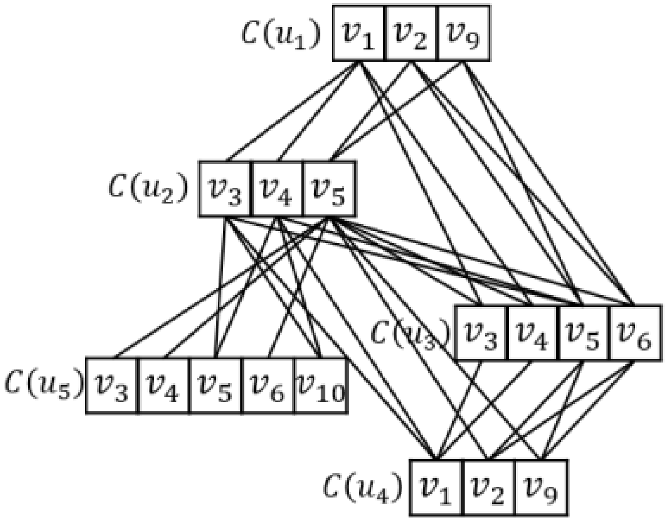
Seok-Hee Hong
University of Sydney
seokhee.hong@sydney.edu.au

Wook-Shin Han*
Pohang University of Science and Technology (POSTECH)
wshan@dblab.postech.ac.kr

Fast subgraph query processing and subgraph matching via static and dynamic equivalences

Hyunjoon Kim[1,2] · Yunyoung Choi[3] · Kunsoo Park[4] · Xuemin Lin[5] · Seok-Hee Hong[6] · Wook-Shin Han[7]

Candidate Space (CS)

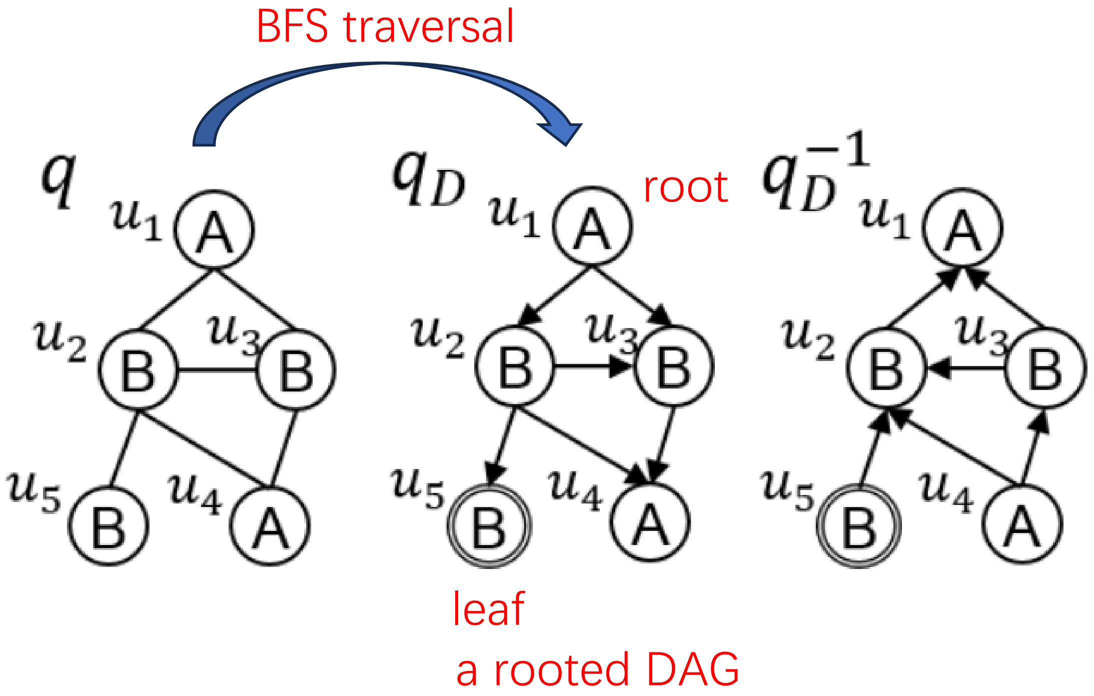


- An auxiliary data structure
- For each $u \in V(q)$, there is a candidate set $C(u)$.
- There is an edge between $v \in C(u)$ and $v' \in C(u')$ iff. $(u, u') \in E(q)$ and $(v, v') \in E(G)$.

- **In this paper**: a more **compact** CS by using **extended DAG-graph DP** (dynamic programming) with an additional filtering function that utilizes a concept called **neighbor-safety**.

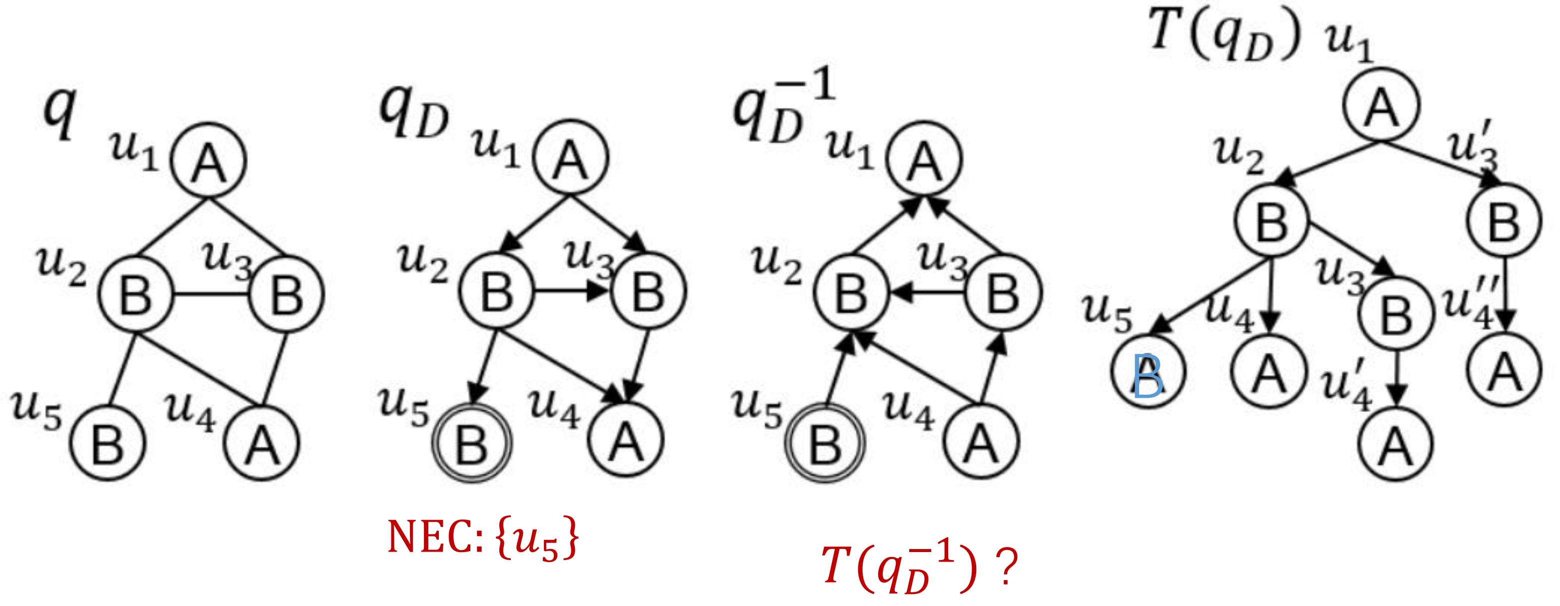
Query DAG

- The vertex with an infrequent label and a large degree is selected as the root r of q_D.
- BFS traversal is performed from r in order to build q_D.



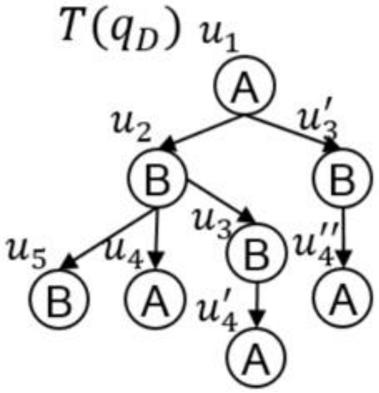
Path Tree

- Let a path tree $T(q)$ of a DAG q be the tree s. t.
  - each root-to-leaf path corresponds to a distinct root-to-leaf path in q,
  - and $T(q)$ shares common prefixes of root-to-leaf paths of q.



DAG-graph Dynamic Programming

- $D[u,v]$ can be computed in a **bottom up order** from leaf vertices to the root vertex. a topological order in DAG



- If $v \notin C(u), D[u,v] = 0$, else
- $D[u,v] = \wedge_{u_c \in Child(u)} f(D[u_c,\cdot], v)$

- $f(D[u_c,\cdot], v) = 1$ if there is v_c adjacent to v in the **CS** such that $D[u_c, v_c] = 1$; 0 otherwise.

Pruning, from RapidFlow

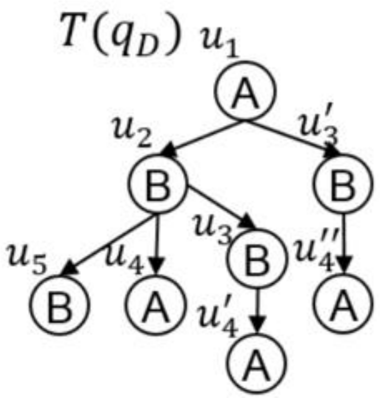
foreach $u' \in N_+^{\delta}(u)$ **do**

$\quad C_A(u) \leftarrow C_A(u) \bigcap (\bigcup_{v \in C_A(u')} I_u^{u'}(v));$

Weak Embedding

- A **weak embedding** M of a rooted DAG q with root u at $v \in V(G)$ is defined as a homomorphism of $T(q)$ such that $M(u) = v$.

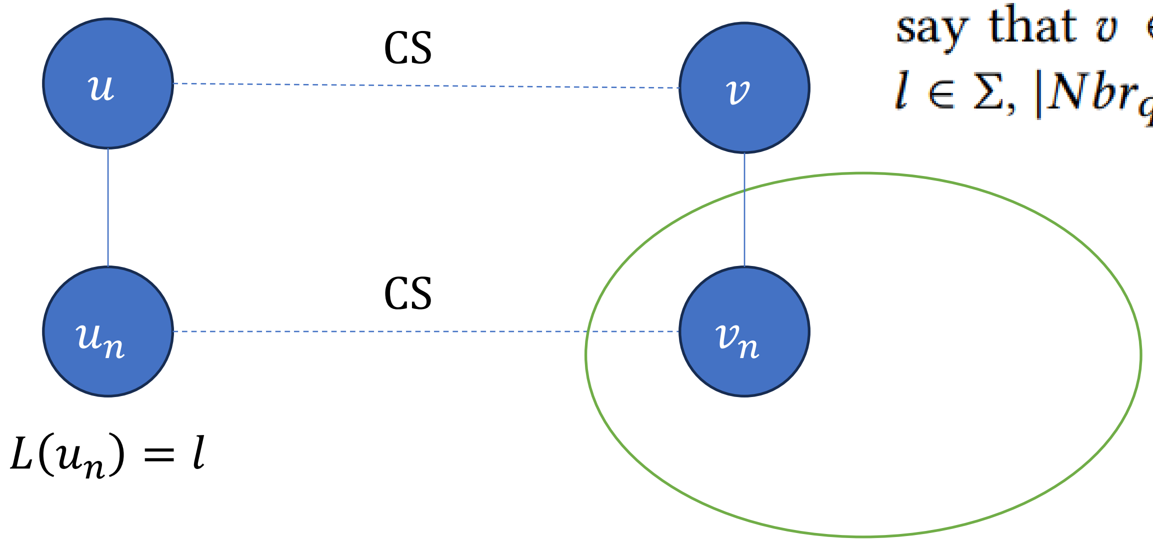
- Intuitively, $D[u, v] = 1$ means that there is a weak embedding M of a sub-DAG q_u at v (i.e., a homomorphism of $T(q_u)$ such that $M(u) = v$) in the CS.



Filtering techniques: neighbor safety

- $Nbr_q(u,l)$ is the set of neighbors of u labeled with l.
- $Nbr_{\mathrm{CS}}(u,v,l)$ is defined as

$$\bigcup_{u_n \in Nbr_q(u,l)} \{v_n \in C(u_n) | v_n \text{ is adjacent to } v \in C(u) \text{ in CS}\}.$$



Definition 4.2. Given a query graph q and a CS on q and G, we say that $v \in C(u)$ is *neighbor-safe regarding* u if for every label $l \in \Sigma$, $|Nbr_q(u,l)| \leq |Nbr_{\mathrm{CS}}(u,v,l)|$.

Extended DAG-graph DP

- We define $h(u,v)$ such that $h(u,v) = 1$ if v is **neighbor-safe** regarding u; $h(u,v) = 0$ otherwise.

$$D[u,v] = \wedge_{u_c \in Child(u)} f(D[u_c,\cdot],v)$$

Simple DAG-graph DP

$$\Downarrow$$

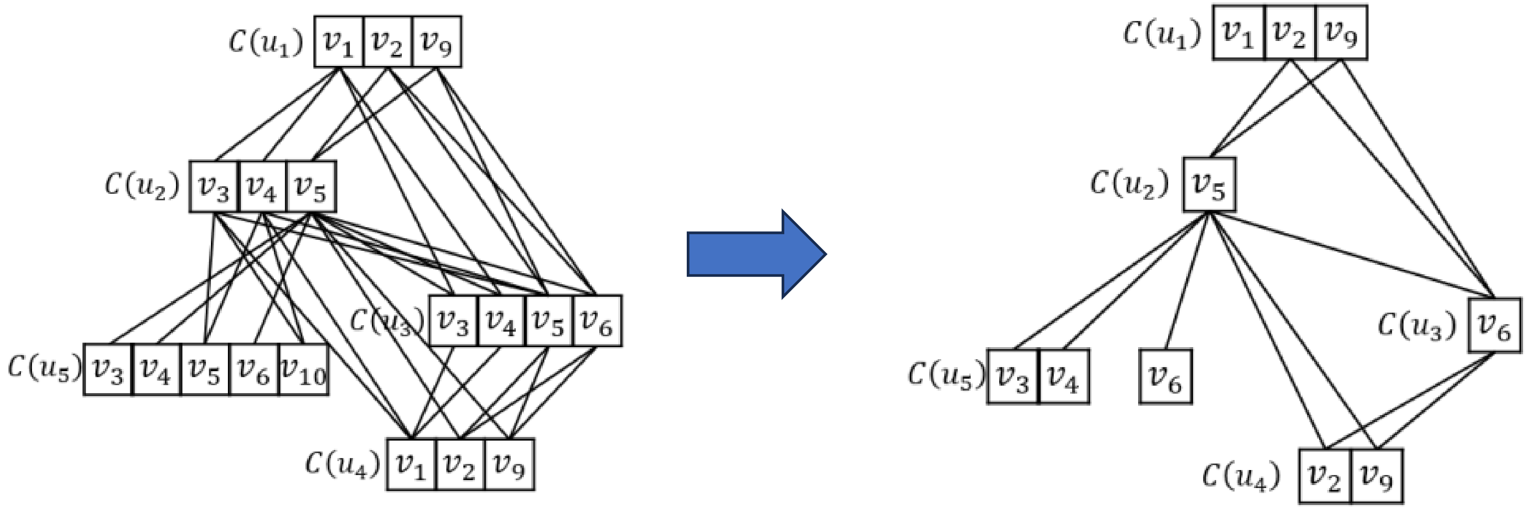
$$D[u,v] = \wedge_{u_c \in Child(u)} f(D[u_c,\cdot],v) \wedge h(u,v)$$

Extended DAG-graph DP

How to build a compact CS?

- 1. Use NLF filter to obtain the initial CS.
- 2. Run simple DAG-graph DP using q_D^{-1} to the initial CS.
- 3. Refine the CS using q_D via extended DAG-graph DP.
- 4. Perform extended DAG-graph DP using q_D^{-1}.

3 steps of DP are enough from empirical study.



How to build a compact CS?

- DAG (e.g., q_D, q_D^{-1}) = pruning order
- DAG-graph DP = pruning

- Instead of determining pruning order on the fly, define DAG before pruning.

- Perform DP three times for better performance.
- Use filtering techniques, e.g., neighbor safety.

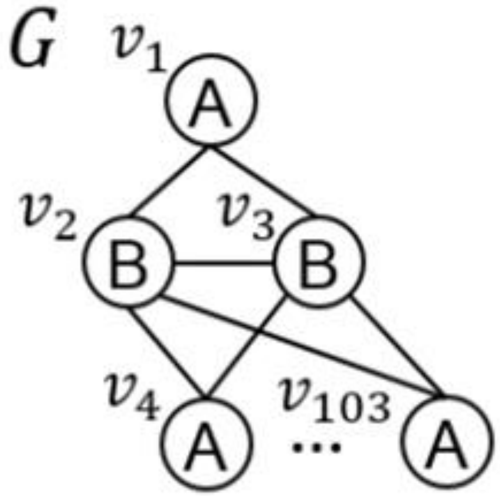
Adaptive matching order

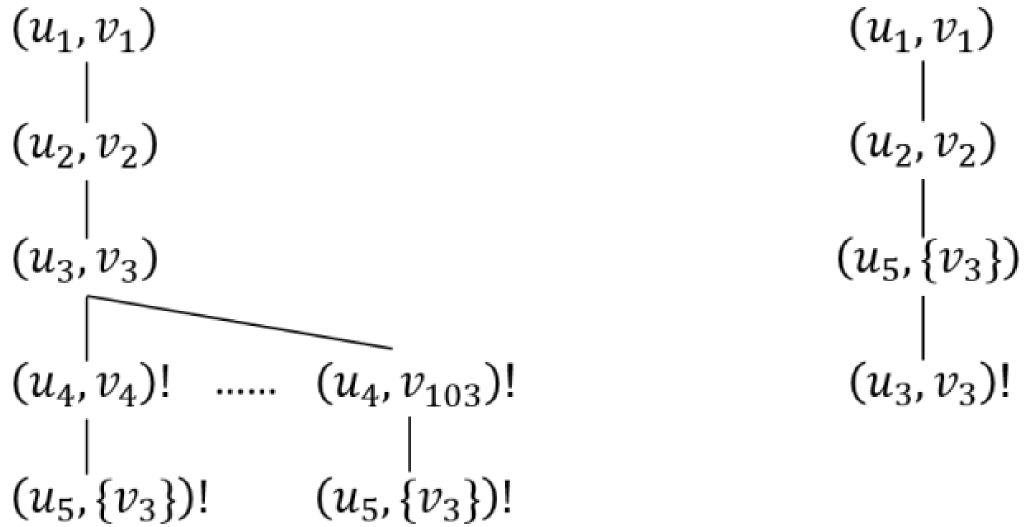
- State-of-the-art SM algorithms adopt leaf decomposition strategy in which the degree-one vertices are matched after the non-degree-one vertices.

- This method generally helps postponing redundant Cartesian product.

- Sometimes inefficient.

Adaptive matching order





(a) Search tree of the existing algorithms with leaf decomposition

(b) Search tree of the matching order based on static equivalence

Figure 4: Search trees of two different adaptive matching orders where $(u, v)!$ means a mapping conflict (i.e, v is already matched therefore u cannot be mapped to v)

Adaptive matching order

- Adaptively select next query vertex according to
  - $|NEC(u)|$ (if u is a leaf vertex)
  - $|C_M(u)|$ (i.e., # of unmapped extendable candidates)

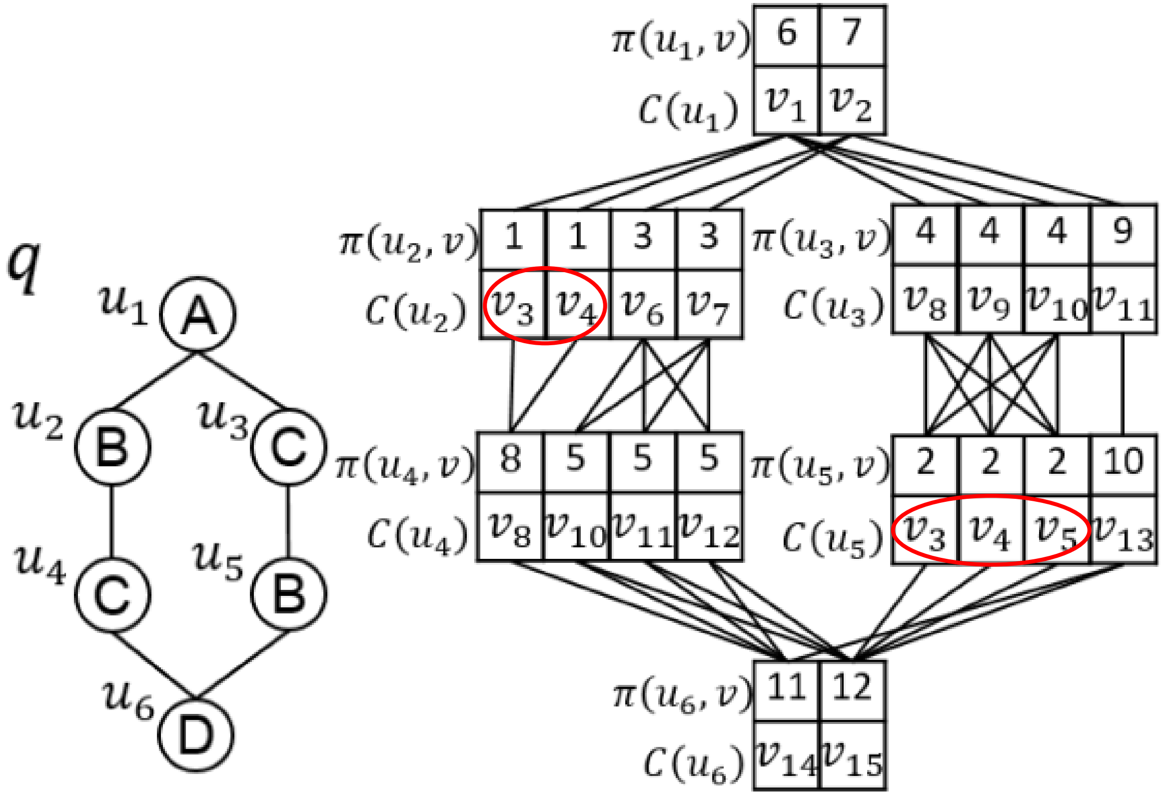
- If there is a degree-one extendable vertex u such that $|\mathrm{NEC}(u)| \geq |U_M(u)|$ where $U_M(u)$ denotes the set of unmapped extendable candidates of u in $C_M(u)$,
  - If $|\mathrm{NEC}(u)| > |U_M(u)|$, backtrack.
  - Otherwise (i.e., if $|\mathrm{NEC}(u)| = |U_M(u)|$), select u as the next vertex.
- Otherwise,
  - If there are only degree-one extendable vertices, select one of them as the next vertex.
  - Otherwise, select an extendable vertex u such that $|C_M(u)|$ is the minimum among non-degree-one vertices.

From RapidFlow

Procedure `Enumerate` (φ, I, M, i)
  if $i = |\varphi| + 1$ **then** Output M, **return**;
  else if $i = 1$ **then** $u \leftarrow \varphi[i]$, $C_M(u) \leftarrow C_I(u)$;
  else $u \leftarrow \varphi[i]$, $C_M(u) \leftarrow \bigcap_{u' \in N_+^{\varphi}(u)} I_u^{u'}(M(u'))$;
  foreach $v \in \boxed{C_M(u)}$ **do**
    if *v is not visited* **then**
      Add (u, v) to M;
      `Enumerate` $(\varphi, I, M, i+1)$;
      Remove (u, v) from M;

Runtime Pruning

Use the knowledge gained from the exploration of the subtree rooted at M.



ID	Cell
1	$\{v_3, v_4\}$
2	$\{v_3, v_4, v_5\}$
3	$\{v_6, v_7\}$
4	$\{v_8, v_9, v_{10}\}$
5	$\{v_{10}, v_{11}, v_{12}\}$
6	$\{v_1\}$
7	$\{v_2\}$
8	$\{v_8\}$
⋮	
12	$\{v_{15}\}$

Figure 5: A query graph q and CS. Every cell $\pi(u, v)$ is represented as a unique ID according to a table above

Runtime Pruning

- Basic idea: avoid equivalent subtrees $M \cup \{(u, v_i)\}, M \cup \{(u, v_j)\}$
  - Case 1: both subtrees lead to failure, or
  - Case 2: these subtrees have *symmetric* full matchings

- It's easy to obtain matching results from a *symmetric* subtree,
- but not easy to find equivalent subtrees.

- This paper provide a method to detect equivalent subtree during runtime.

Summary

- Compact candidate space (with DP and neighbor-safety filtering)
- Adaptive matching order
- Runtime pruning by dynamic equivalence

SUFF: Accelerating Subgraph Matching with Historical Data

Xun Jian
Hong Kong University of Science and Technology
Hong Kong, China
xjian@connect.ust.hk

Zhiyuan Li
Hong Kong University of Science and Technology
Hong Kong, China
zlicw@cse.ust.hk

Lei Chen
Hong Kong University of Science and Technology
Hong Kong, China
leichen@cse.ust.hk

Historical Data?

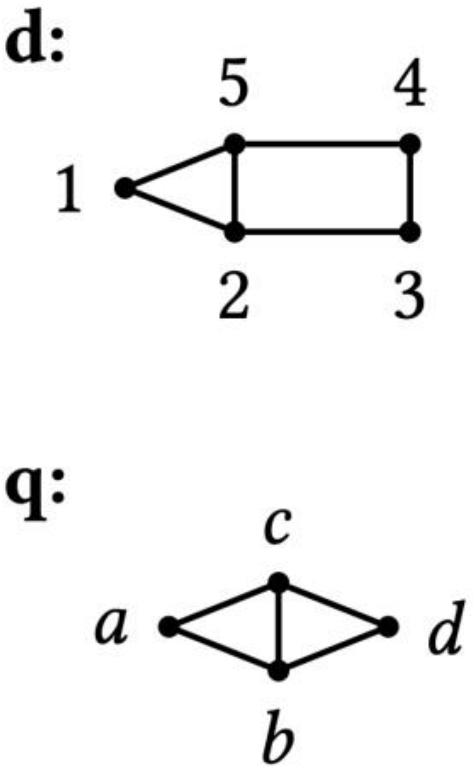
- This paper studies SM, not CSM.

- Data graph is static.
- Consider a subgraph matching system that continually accepts queries from users.

Bloom filter (or Cuckoo filter)

- This paper designs a framework which builds a **filter database** using matching results of **past queries**, and uses them to prune the search space for **future queries**.

Example



(a) Data graph and query graph.

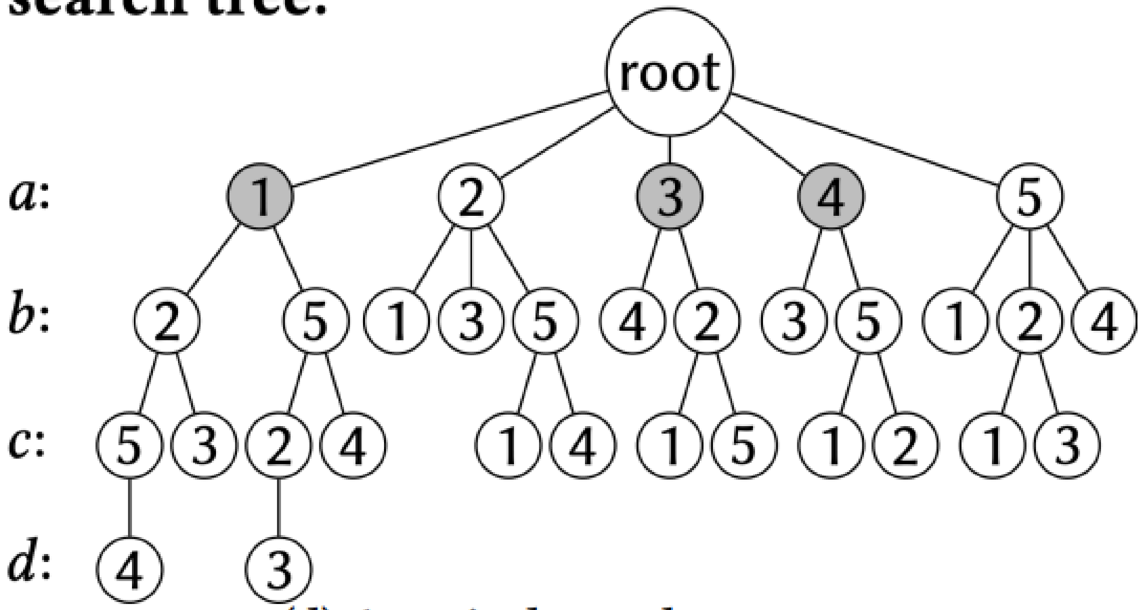
$\triangle(a, b, c)$ **in d:**

$(1,2,5)$ $(1,5,2)$
$(2,1,5)$ $(2,5,1)$
$(5,2,1)$ $(5,1,2)$

$\square(a, b, c, d)$ **in d:**

$(2,3,4,5)$ $(2,5,4,3)$
$(3,4,5,2)$ $(3,2,5,4)$
$(4,5,2,3)$ $(4,3,2,5)$
$(5,2,3,4)$ $(5,4,3,2)$

search tree:



(d) A typical search tree.

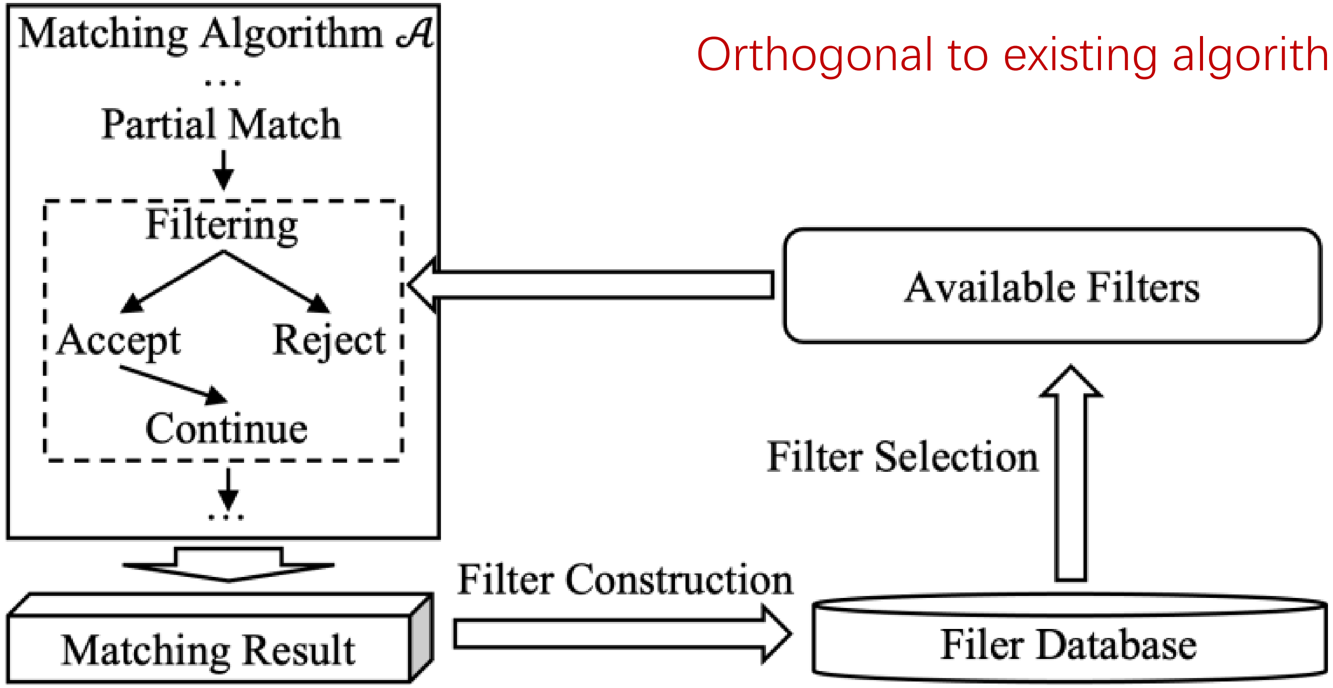
Utilize partial matchings of past queries

- Assume that q' is a subgraph of q.
- **Lemma**: Given a full match $f \in M(q,d)$, let $f_p[V]$ be a partial match of f, where $V \subset V(q') \cap V(q)$, then there exists a full match $h \in M(q',d)$, s.t. $\forall v \in V, f_p(v) = h(v)$.

- Each time the output of query q is produced, filters are constructed for different subsets of $V(q)$, which correspond to partial matches. (In the phase of **Filter Construction**)

Framework Overview

Orthogonal to existing algorithms



Algorithm 1: A Typical Modified Matching Algorithm

Input : Data graph d, query graph q, filter database Φ.
Output : All matches of q in d.

$\mathcal{D} \leftarrow$ generate auxiliary data structure;
$F \leftarrow$ selected filters from Φ;
`Enumerate`$(d, q, \mathcal{D}, \{\}, 1)$;
Procedure `Enumerate`$(d, q, \mathcal{D}, f, i)$:
  if $i = |V(q)| + 1$ **then**
    Output f;
    return;
  if *any filter in F rejects f* **then return**;
  $u \leftarrow$ the query vertex to be matched in this level;
  $C \leftarrow$ generate candidates for u;
  foreach $v \in C$ **do**
    if $v \notin f$ **then**
      Add $\{u \mapsto v\}$ to f;
      `Enumerate`$(d, q, \mathcal{D}, f, i+1)$;
      Remove $\{u \mapsto v\}$ from f;

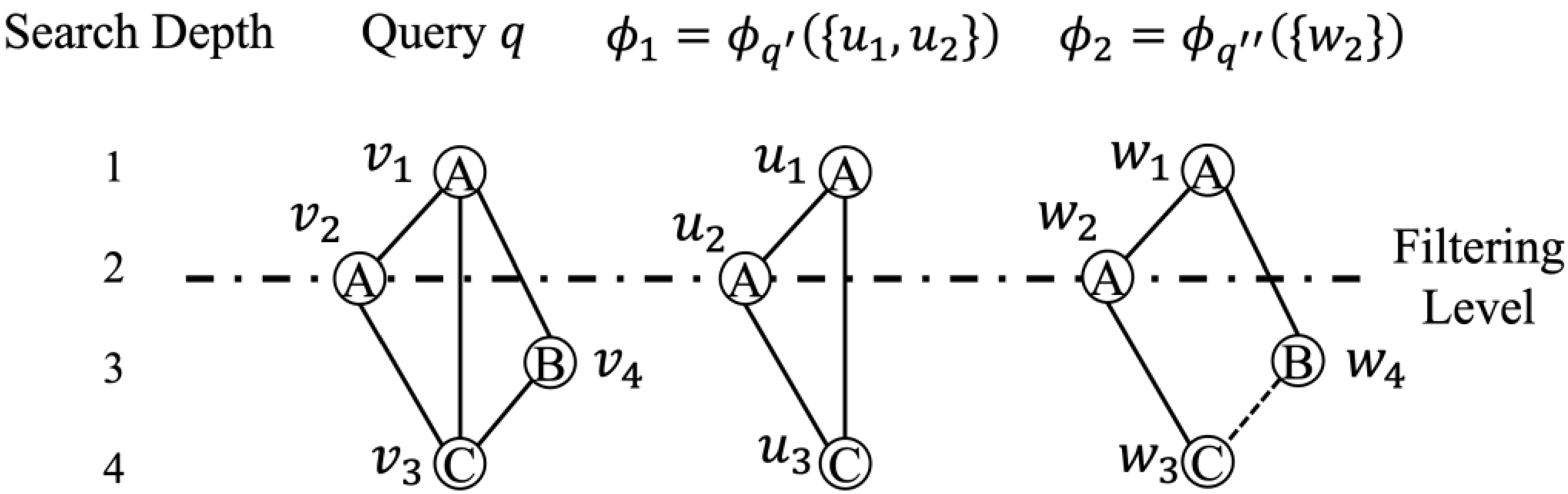
Utilizing Filters

- Query graph q. Data graph d.
- Given a match set $M(q,d)$, we can build a filter $\phi_q(V)$ for each $V \subset V(q)$.
- Each filter $\phi_q(V)$ stores all the partial matches in $\{h_p[V], \forall h \in M(q,d)\}$.

Utilizing Filters

- $V = \{u_1\}, \{u_1, u_2\}, \dots, \{u_1, u_2, \dots, u_a\}, \{u_2\}, \{u_3\}, \dots, \{u_a\}$;
- $2a - 1$ filters in total, $a = |V(q)|$.

- ~~q' is usable for q only if q' is a subgraph of q.~~

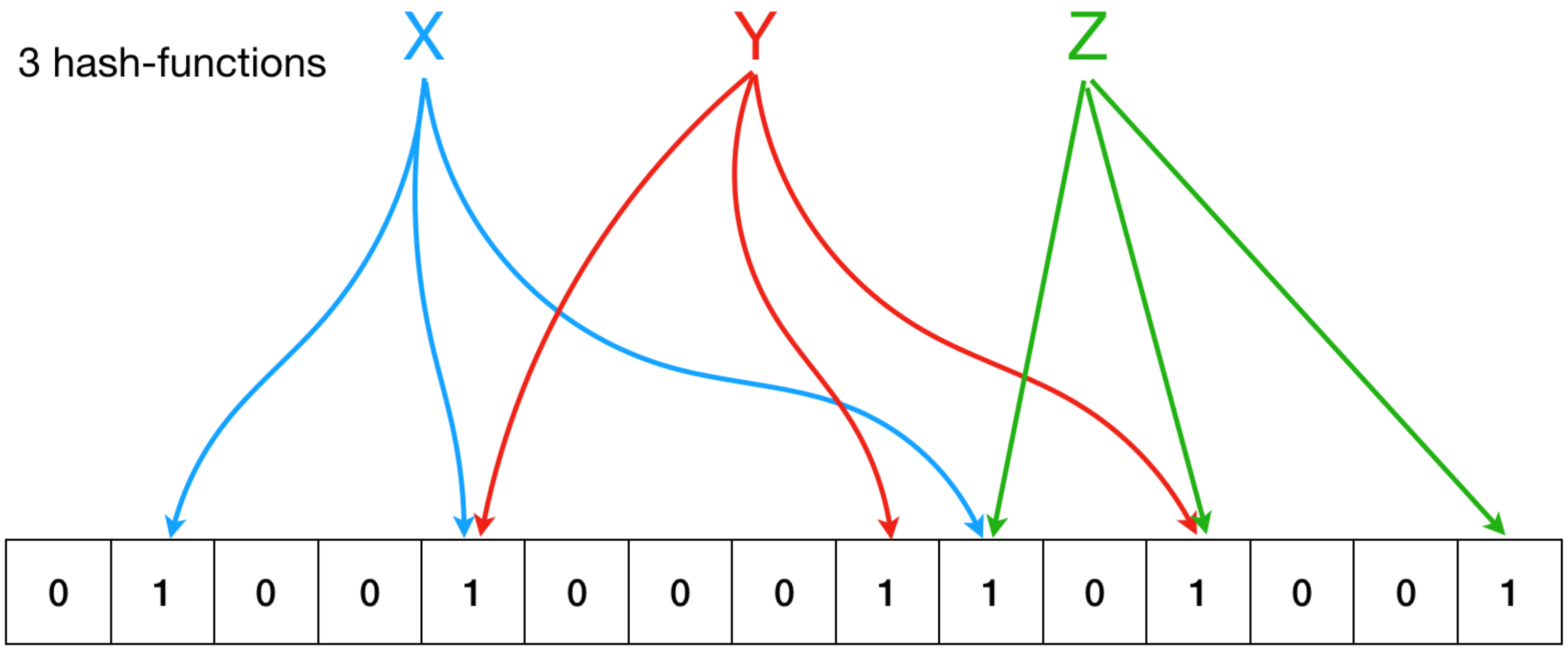


Previous query q' and q'', they are both subgraph of q.

Utilizing Filters

- Bloom filter, a space-efficient representation of a set S
- A Bloom filter is a m-bit array
- Small false-positive rate (e.g., 0.1)

- $\phi_q(V)$?
- $\phi_q(\{v_1, v_2\})$



The Maximum Utility Problem

- Pick a set of filters such that the overall performance is maximized.
- Goal: maximize utility score.
- NP-hard
- This paper uses a greedy approximate algorithm .

Summary

- It can prune the search space not only on a **single vertex** but also on **a set of vertices**, which potentially provides more pruning power.
- SUFF can achieve up to **15X speedup** with small overheads.

Efficient GPU-Accelerated Subgraph Matching

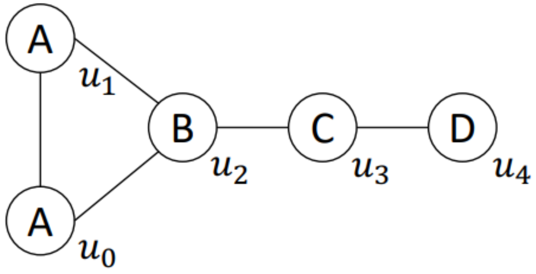
XIBO SUN, The Hong Kong University of Science and Technology, Hong Kong SAR
QIONG LUO, The Hong Kong University of Science and Technology, Hong Kong SAR and The Hong Kong University of Science and Technology (Guangzhou), China

Subgraph Matching on CPU and GPU

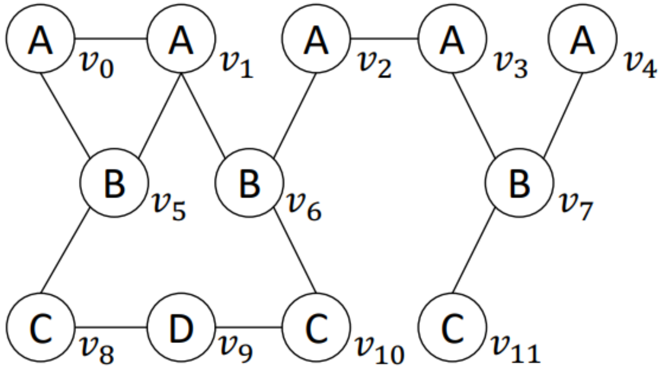
- CPU-based algorithms are highly optimized
  - Matching on some graphs is still time-consuming

- Some researchers start to adopt modern hardware, including GPU
  - 1. Ineffective filtering and ordering
  - 2. Memory-inefficient BFS enumeration
  - This paper addresses above two problems

Trie (CSR)

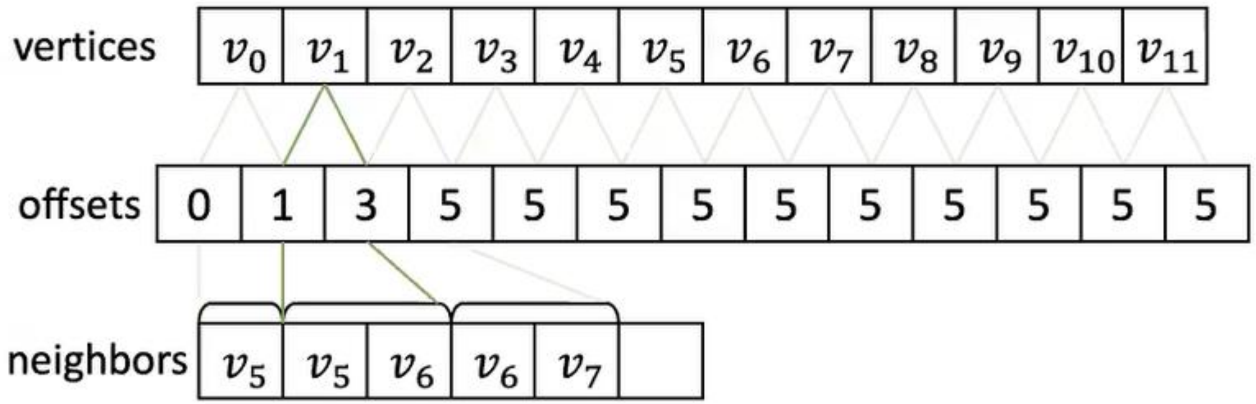


(a) Query graph Q.



(b) Data graph G.

- Trie (CSR) is commonly used to store a relation,
  - Good for retrieving neighbors of a vertex
  - **Many vertices and offsets for big relations**
  - **Expensive to update**



u_0	u_1
v_0	v_1
v_1	v_0
v_2	v_3
v_3	v_2

u_0	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

u_1	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

Index

u_2	u_3
v_5	v_8
v_6	v_{10}

u_3	u_4
v_8	v_9
v_{10}	v_9

Cuckoo Hashing m=2

Cuckoo Hashing

procedure insert(x)
  if lookup(x) **then return**
  loop MaxLoop **times**
    if $T_1[h_1(x)] = \perp$ **then** { $T_1[h_1(x)] \leftarrow x$; **return** }
    $x \leftrightarrow T_1[h_1(x)]$
    if $T_2[h_2(x)] = \perp$ **then** { $T_2[h_2(x)] \leftarrow x$; **return** }
    $x \leftrightarrow T_2[h_2(x)]$
  end loop
  rehash(); insert(x)
end

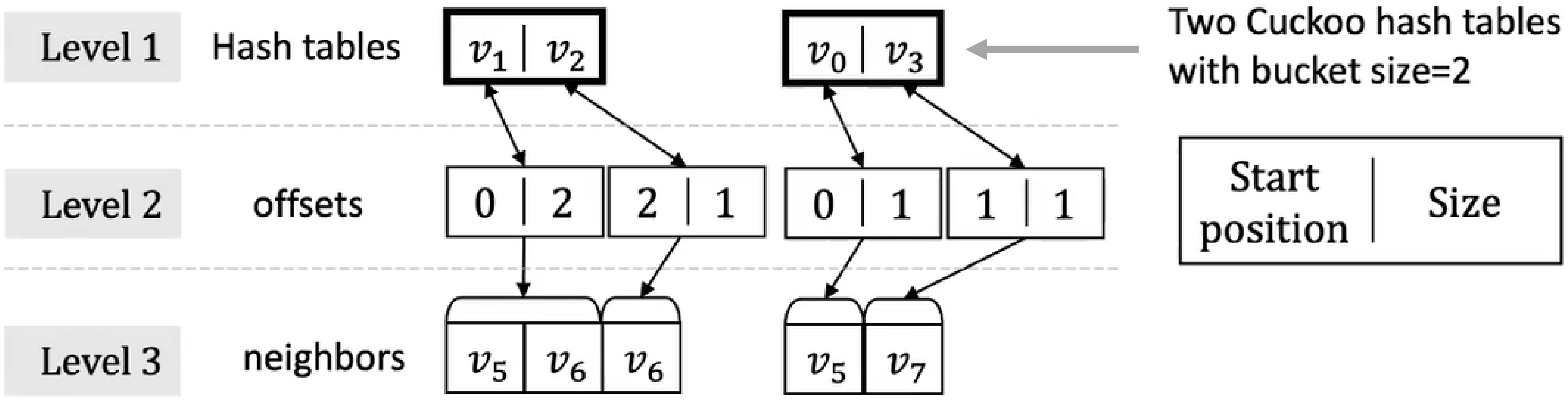
A group of m independent hash functions, each corresponding to a separate hash table.

Cuckoo Tries

To support dynamic maintenance of candidates for filtering.
It is difficult to maintain dynamic data structures on the GPU.

- Each relation is implemented as a set of Cuckoo tries
  - Level 1 corresponds to Cuckoo hash tables
    - **Multiple** hash tables, **Worst case $O(1)$** access time, no warp divergence
    - **Bucketize elements** to fully utilize the high memory bandwidth
  - Level 2 indicates the position of each neighbor set
    - On filtering, the modification of one neighbor set does not affect the others
  - Level 3 stores the neighbor sets consecutively
- Efficient batch-insertion, deletion, and search process

u_0	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

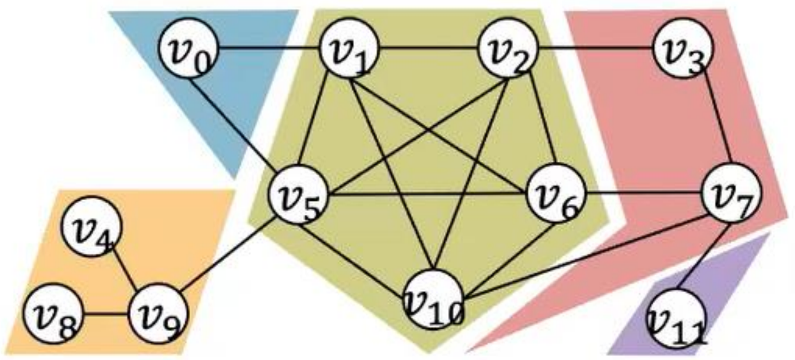


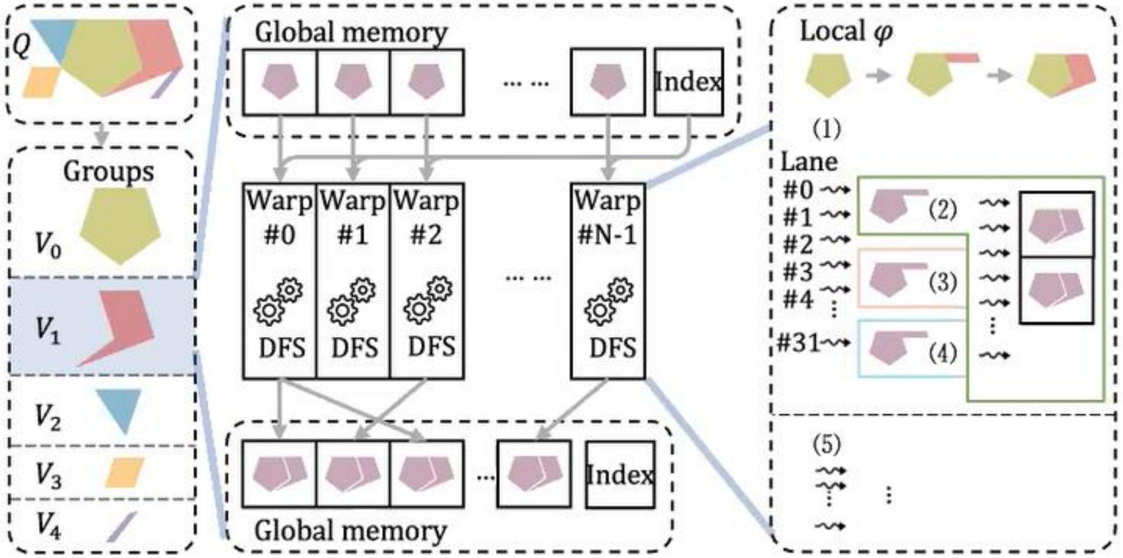
Enumeration: Parallel BFS and DFS

- Parallel-BFS enumeration
  - Utilized by most GPU-based algorithms
  - In each step, all partial results are extended by one vertex
  - A warp extends a partial match of size m to all partial matches of size $m + 1$
  - Large memory consumption and many memory accesses
- Parallel-DFS enumeration
  - A warp obtains an edge and get all complete matches containing the edge
  - Alleviate the memory issues
  - Severe load imbalance due to the search space

Hybrid BFS-DFS Enumeration

- Hybrid parallel BFS-DFS extension method
  - Organize vertices in Q into groups $(V_0, V_1, \ldots, V_n)$ based on the structure of Q
    - Dense vertex, then sparse vertices, and finally tree vertices
  - Extend vertices within the same group in DFS
  - Write all partial results when a group is finished (BFS)





VINCENT: Towards Efficient Exploratory Subgraph Search in Graph Databases

Kai Huang [§,†], Qingqing Ye [†], Jing Zhao [§], Xi Zhao [§], Haibo Hu [†], Xiaofang Zhou [§]
[§]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
[†]Department of Electronic and Information Engineering, Hong Kong Polytechnic University
ustkhuang|xizhao|zxf@ust.hk,qqing.ye|haibo.hu@polyu.edu.hk,jzhaobq@connect.ust.hk

Background

- Mary wants to query a substructure q, but she does not have precise knowledge of the subgraph structure due to the topological complexity of data graphs.
- If there is an efficient tool that supports exploratory search, Mary can formulate an initial query graph (e.g., a subgraph of q) and then iteratively formulate the query and explore the query results, and finally identify the exact query q.

- 结束