RapidFlow

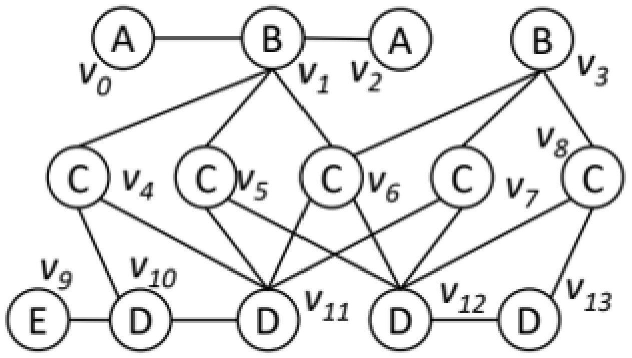
20230901

Introduction

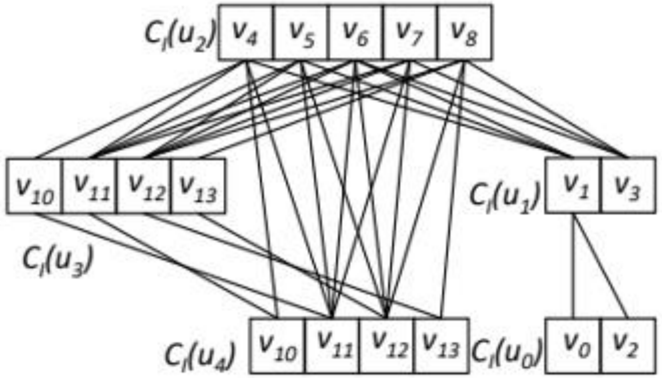
- 已有 CSM 算法的两个问题
  - 1. 匹配序列总从更新的边开始，搜索时可能遇到大量的无效结果。
  - 2. 如果 Q 包含多个自同构，将会导致很多重复计算。
- 把“枚举 $\Delta M_{e(u_a,u_b)}$”这一问题转化为“求 M_{Q_R}”。
  - $\Delta M_{e(u_a,u_b)}$ 表示把 (u_a, u_b) 映射到更新的边的新增的匹配。
  - $Q_R = Q - \{u_a, u_b\}$
- 利用两级索引，提取出更新的边会影响的 G 的子图（affected region）。

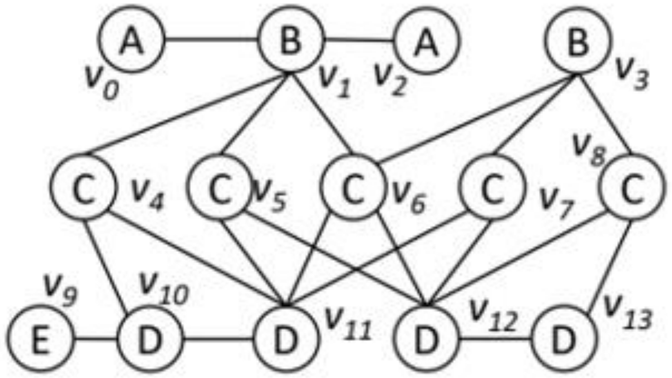
Existing CSM Framework

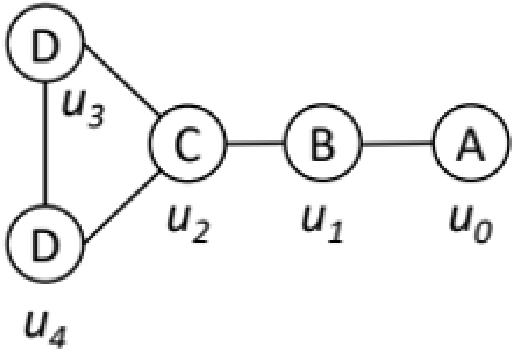
$I \leftarrow$ build an index based on Q and G;
foreach $\Delta G = (\oplus, e) \in \Delta\mathcal{G}$ **do**
  if $\oplus$ *is* $+$ **then**
    Add e to G and update I;
    $\texttt{FindIncrementalMatch}(Q, I, e)$;
  else
    $\texttt{FindIncrementalMatch}(Q, I, e)$;
    Remove e from G and update I;

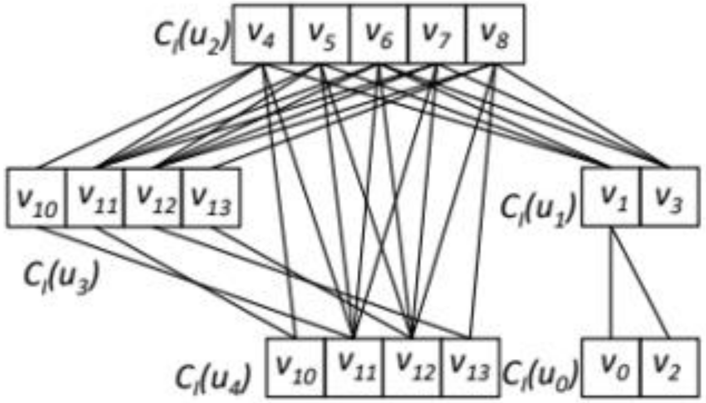


index I maintains a candidate set $C_I(u)$ for $u \in V(Q)$ and records edges between $C_I(u)$ and $C_I(u')$ if $e(u, u') \in E(Q)$.









☐ NLF

Particularly, given $u \in V(Q)$ and $v \in V(G)$, NLF requires that given $l \in L(N(u))$, $|N(u,l)| \leqslant |N(v,l)|$ where $L(N(u)) = \{L(u')|u' \in N(u)\}$ (i.e., the set of labels of u's neighbors) and $N(u,l) = \{u' \in N(u)|L(u') = l\}$ (i.e., the set of u's neighbors with label l).

foreach $u \in V(Q)$ **do**

  $C_I(u) \leftarrow \{v \in V(G)|L(u) = L(v) \wedge NLF(u,v) \text{ is true}\}$;

☐ Basic Method: Filtering $C(u)$ based on the label $L(u)$ and degree $d(u)$ of u, i.e., $C(u) = \{v \in V(G)|L(v) = L(u) \wedge d(v) \geq d(u)\}$

☐ 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 $C(u)$.

Subgraph Matching

Graph Exploration based Approaches

- **General Idea:**
  Input: a query graph q and a data graph G
  Output: all subgraph isomorphisms from q to G
  1. Generate a matching order π;
  2. Obtain a complete candidate set $u.C$ for every vertex $u \in V(q)$;
  3. Recursively enumerate all solutions by extending partial subgraph isomorphisms iteratively along π.

现有 CSM 问题的解法使用了以上 SM 算法，即，
枚举 (u_a, u_b)，对于每一个 (u_a, u_b)，使用 SM 算法求出 $\Delta M_{e(u_a,u_b)}$，
这些结果形成了 ΔM。

Generic Subgraph Matching

Algorithm 1: Generic Subgraph Matching

Input: a query graph q and a data graph G;
Output: all matches from q to G;

`/* The filtering method. */`
$C, \mathcal{A} \leftarrow$ generate candidate vertex sets and build auxiliary data structure;
`/* The ordering method. */`
$\varphi \leftarrow$ generate a matching order;
`/* The enumeration method. */`
Enumerate$(q, G, C, \mathcal{A}, \varphi, \{\}, 1)$;
Procedure Enumerate$(q, G, C, \mathcal{A}, \varphi, M, i)$
  if $i = |\varphi| + 1$ **then** Output M, **return**;
  $u \leftarrow$ select an extendable vertex given φ and M;
  `/* The local candidate computation method. */`
  $LC(u, M) \leftarrow$ComputeLC$(q, G, C, \mathcal{A}, \varphi, M, u, i)$;
  foreach $v \in LC(u, M)$ **do**
    if $v \notin M$ **then**
      Add (u, v) to M;
      Enumerate$(q, G, C, \mathcal{A}, \varphi, M, i + 1)$;
      Remove (u, v) from M;

Procedure FindIncrementalMatch$(Q, I, e(v_a, v_b))$ // conduct the whole algorithm on I instead of G

$\Delta M \leftarrow \{\}$;

foreach $e(u_a, u_b) \in E(Q)$ **do**

if $L(u_a) = L(v_a)$ *and* $L(u_b) = L(v_b)$ **then**

$\varphi \leftarrow$ generate a matching order beginning with u_a, u_b;

$M \leftarrow \{(u_a, v_a), (u_b, v_b)\}$; // Initialize M. M is now a partial result

$\Delta M_{e(u_a,u_b)} \leftarrow$Enumerate$(\varphi, I, M, 3)$; // Initial recursive depth is 3

$\Delta M \leftarrow \Delta M \cup \Delta M_{e(u_a,u_b)}$; // ΔM_e is incremental matches mapping e to the updated edge

Output ΔM;

Procedure Enumerate(φ, I, M, i) // The integer i is the recursive depth. Index of φ starts from 1.

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 $v \in C_M(u)$ **do**

if v *is not visited* **then** // not being mapped to

Add (u, v) to M;

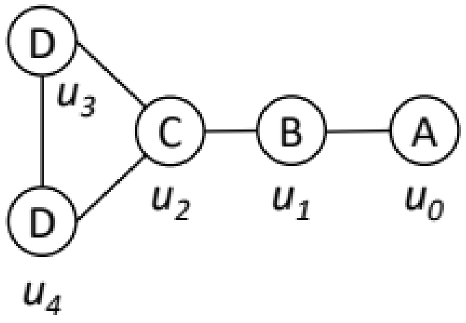
Enumerate$(\varphi, I, M, i+1)$;

Remove (u, v) from M;

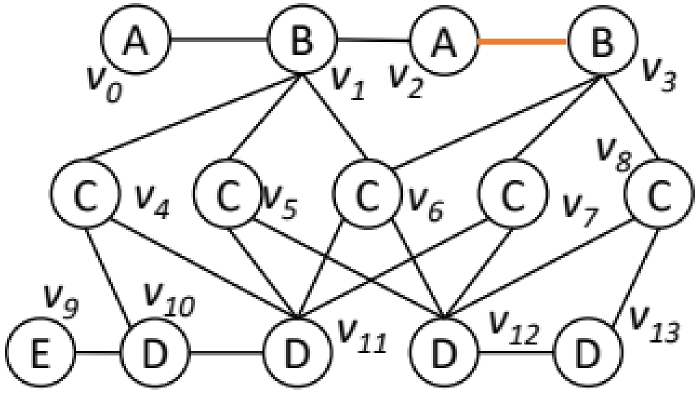
如果 u' 映射到 $M(u')$ 的话，u 的候选集合为 $I_u^{u'}(M(u'))$

//set $C_M(u)$ to the set of common neighbors of candidates who are mapped to query vertices $u' \in N_+^{\varphi}(u)$ where $N_+^{\varphi}(u)$ is the set of u' neighbors before u in φ.

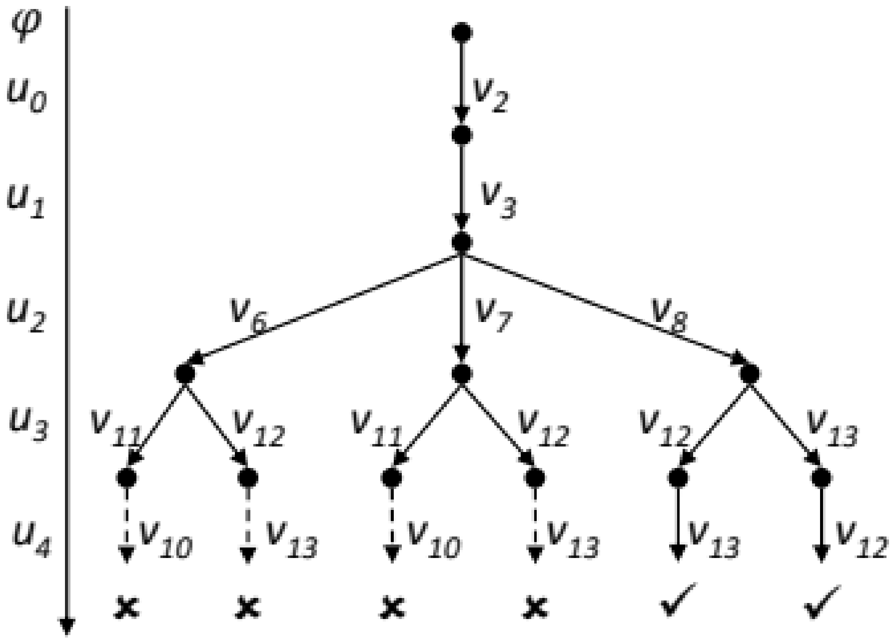
Given $e(u, u') \in E(Q)$ and $v \in C_I(u)$, $I_{u'}^{u}(v) = N(v) \cap C_I(u')$ (i.e., the neighbors of v who are in the candidate set of u').



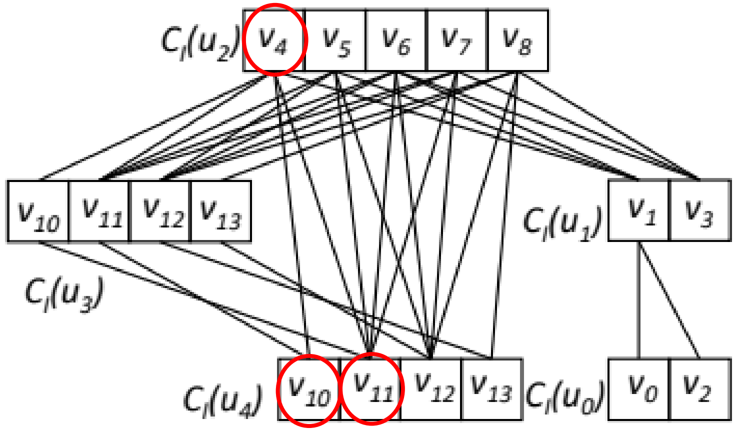
(a) Query graph Q.



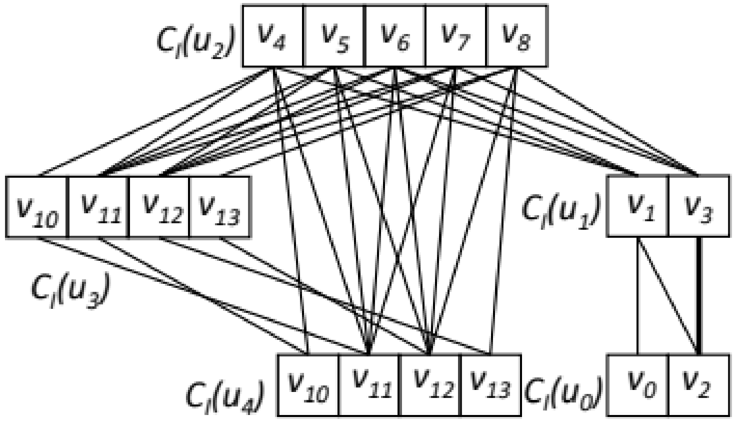
(b) Data graph G.



(c) Search tree.



(a) Index on G.



(b) Index on G'.

G': Insert $e(v_2, v_3)$ to G

Problems in Existing CSM Framework

- 1. The matching order is required to begin with query edges mapped to the updated edge, which may lead to many invalid partial results.
- 2. Existing approaches may perform redundant computation if the query graph has more than one automorphism.

RapidFlow Overview

/* The offline stage.

$I \leftarrow$ `BuildGlobalIndex` (Q, G); // the global index I is consistent with G where we can find all matches of Q in G.

$\mathcal{X} \leftarrow$ `GenerateAutoSet` (Q); // disjoint sets based on automorphisms of Q

/* The online stage.

foreach $\Delta G = (\oplus, e) \in \Delta\mathcal{G}$ **do**

  if $\oplus$ *is* $+$ **then**

    $G \leftarrow G \oplus \Delta G$;

    `UpdateGlobalIndex` $(Q, G, I, \oplus, e)$;

    `FindIncrementalMatch` $(Q, I, e, \mathcal{X})$;

  else

    `FindIncrementalMatch` $(Q, I, e, \mathcal{X})$;

    $G \leftarrow G \oplus \Delta G$;

    `UpdateGlobalIndex` $(Q, G, I, \oplus, e)$;

Procedure `FindIncrementalMatch` $(Q, I, e(v_a, v_b), \mathcal{X})$

  $\Delta\mathcal{M} \leftarrow \{\}$;

  foreach $X \in \mathcal{X}$ **do**

    $e(u_a, u_b) \leftarrow$ an arbitrary edge in X;

    $Q_R \leftarrow Q - \{u_a, u_b\}$;

    $A \leftarrow$ `BuildLocalIndex` $(Q_R, I, e(u_a, u_b), e(v_a, v_b))$;

    if *there are empty candidate sets in A* **then** Continue;

    $\varphi \leftarrow$ generate a matching order of Q_R;

    $\mathcal{M}_{Q_R} \leftarrow$ `Enumerate` $(\varphi, A, \{\}, 1)$;

    $\Delta\mathcal{M}_{e(u_a,u_b)} \leftarrow \{\{(u_a, v_a), (u_b, v_b)\} \cup M \mid M \in \mathcal{M}_{Q_R}\}$;

    $\Delta\mathcal{M}_X \leftarrow$ `DualMatch` $(\Delta\mathcal{M}_{e(u_a,u_b)}, X)$;

    $\Delta\mathcal{M} \leftarrow \Delta\mathcal{M} \cup \Delta\mathcal{M}_X$;

  Output $\Delta\mathcal{M}$;

Reduce CSM to BSM

generate candidate set for every vertices in Q_R

- $M(u_a) = v_a, M(u_b) = v_b$
- $u \in N(u_a) \Rightarrow M(u) \in N(v_a)$
- $\Rightarrow C(u) \subseteq N(v_a)$

- $u' \in N(u) \Rightarrow C(u') \subseteq N\big(M(u)\big)$
- $\Rightarrow C(u') \subseteq \bigcup_{v \in \mathrm{C}(u)} N(v)$

$$\Delta\mathcal{M}_{e(u_a,u_b)} \leftarrow \{\{(u_a, v_a), (u_b, v_b)\} \cup M | M \in \mathcal{M}_{Q_R}\};$$

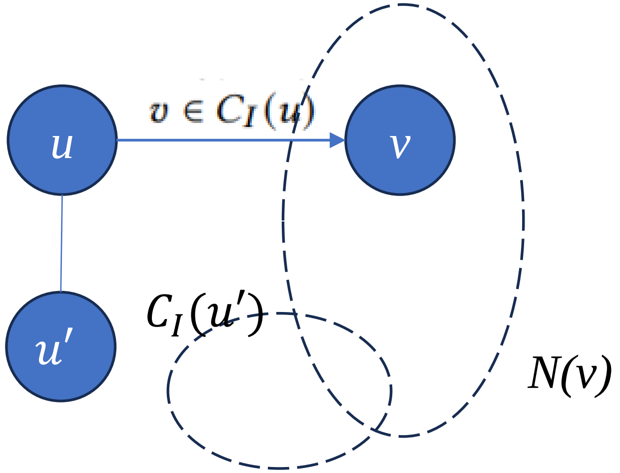
Two-Level Indexing Mechanism

- Global index (the first-level index) is **query-dependent**.
- Local index (the second-level index) is **update-dependent**.
- Local index is built on top of the global index for each update and immediately destroyed after the search procedure.

Global Index

to obtain the affected region

Particularly, given $u \in V(Q)$ and $v \in V(G)$, NLF requires that given $l \in L(N(u))$, $|N(u,l)| \leqslant |N(v,l)|$ where $L(N(u)) = \{L(u')|u' \in N(u)\}$ (i.e., the set of labels of u's neighbors) and $N(u,l) = \{u' \in N(u)|L(u') = l\}$ (i.e., the set of u's neighbors with label l).



对于边 (u,u')，如果 u 映射到 v，那么 u' 可能映射到的点属于 $I^u_{u'}(v)$.

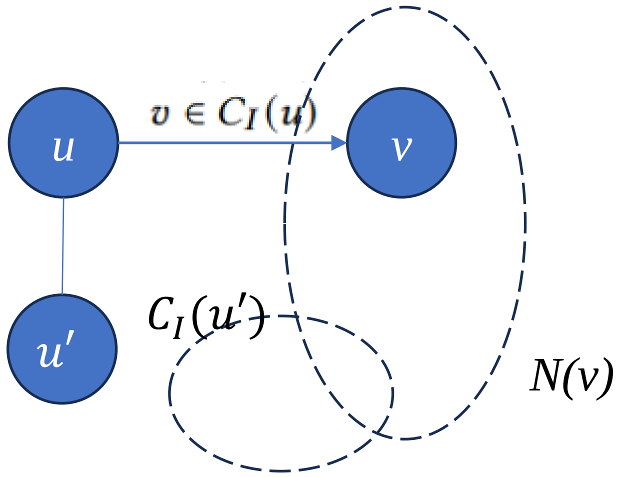
Algorithm 3: Global Index

Procedure `BuildGlobalIndex`(Q, G)
- **foreach** $u \in V(Q)$ **do**
  - $C_I(u) \leftarrow \{v \in V(G)|L(u) = L(v) \wedge NLF(u,v)$ is true$\}$; // neighbor label frequency (NLF) filter
- **foreach** $e(u,u') \in E(Q)$ **do**
  - **foreach** $v \in C_I(u)$ **do**
    - $I^u_{u'}(v) \leftarrow N(v) \cap C_I(u')$; $I^u_{uu}(v) \cap C_I(uu)$
- **return** I;

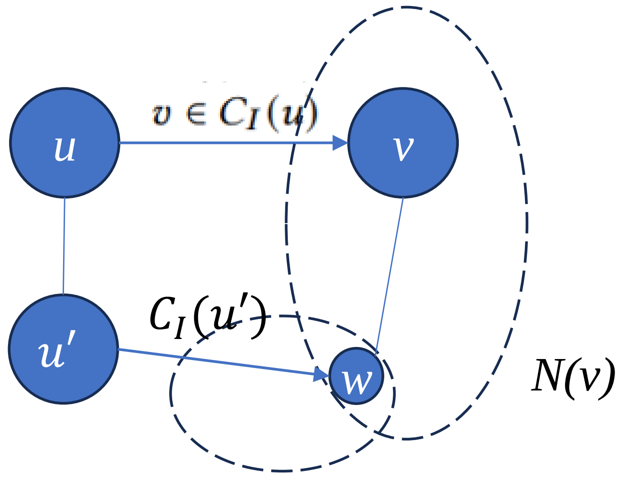
/* Maintain the index given an update. */

Procedure `UpdateGlobalIndex`$(Q, G, I, \oplus, e(v_a, v_b))$
- **foreach** $\{(u,u'),(v,v')\} \in E(Q) \times \{(v_a,v_b),(v_b,v_a)\}$ **do**
  - **if** $v \in C_I(u)$ *and* $v' \in C_I(u')$ **then**
    - Add v' to $I^u_{u'}(v)$ and add v to $I^{u'}_u(v')$;
- $\Delta C_I \leftarrow \{\}$;
- **foreach** $(u,v) \in V(Q) \times \{v_a, v_b\}$ **do** // 尚未加入索引
  - **if** $L(u) = L(v)$ *and* $\boxed{v \notin C_I(u)}$ *and NLF(u, v) is true* **then**
    - Add v to $C_I(u)$ and add (u,v) to ΔC_I;
- **foreach** $u' \in N(u)$ *where* $(u,v) \in \Delta C_I$ **do**
  - $I^u_{u'}(v) \leftarrow N(v) \cap C_I(u')$;
  - Add v to $I^{u'}_u(v')$ given $v' \in I^u_{u'}(v)$;

关于 $I_{u'}^{u}$ 和 $I_{u}^{u'}$



对于边 (u, u')，如果 u 映射到 v，那么 u' 可能映射到的点属于 $I_{u'}^{u}(v)$.



对于边 (u, u')，如果 u' 映射到 w，那么 u 可能映射到的点属于 $I_{u}^{u'}(w)$.

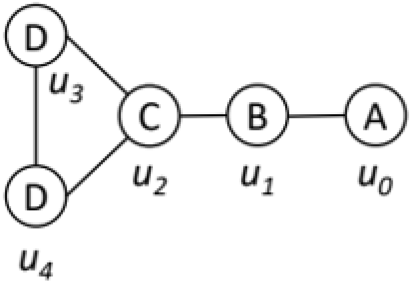
I 维护了 $C(u)$ 和 $C(u')$ 之间的边，且 $I_{u'}^{u}$ 和 $I_{u}^{u'}$ 是对称的。

Given $e(u, u') \in E(Q)$, if $v \in C_I(u)$ and $w \in C_I(u')$, then we will always add $e(v, w)$ to the global index I.

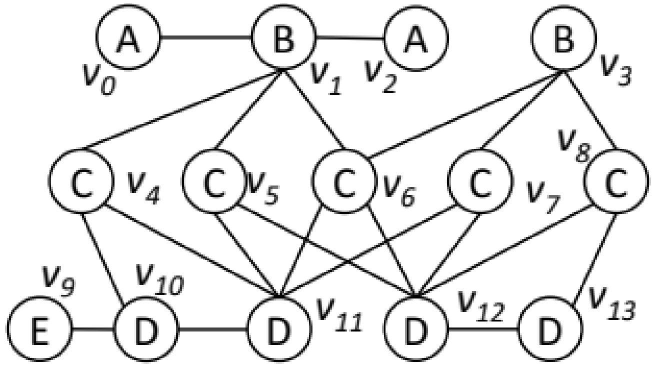
Global Index

Example 4.2. Figure 4a demonstrates I given Q and G in Figure 1. $I_{u_0}^{u_1}(v_1) = \{v_0, v_2\}$. Although $L(v_3) = L(u_1)$ in Figure 1b, $v_3 \notin C_I(u_1)$ because $|N(v_3, A)| = 0$, which is less than $|N(u_1, A)| = 1$. Given insertion of $e(v_2, v_3)$ in Figure 1c, $NLF(u_1, v_3)$ is true. Therefore, we add v_3 to $C_I(u_1)$ and update edges between candidates in Figure 4b.

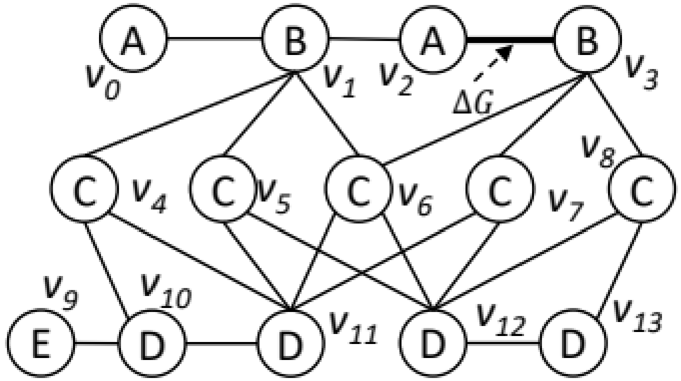
We only check whether v_a and v_b can be inserted into certain candidate sets based on NLF. So here v_3 is inserted into $C_I(u_1)$.



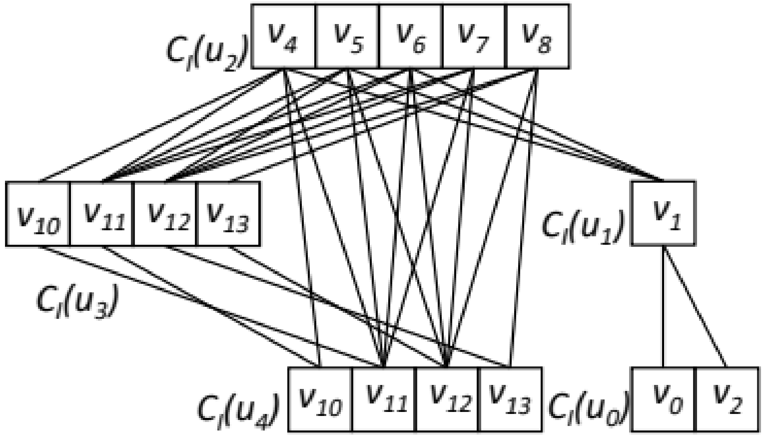
(a) Query graph Q.



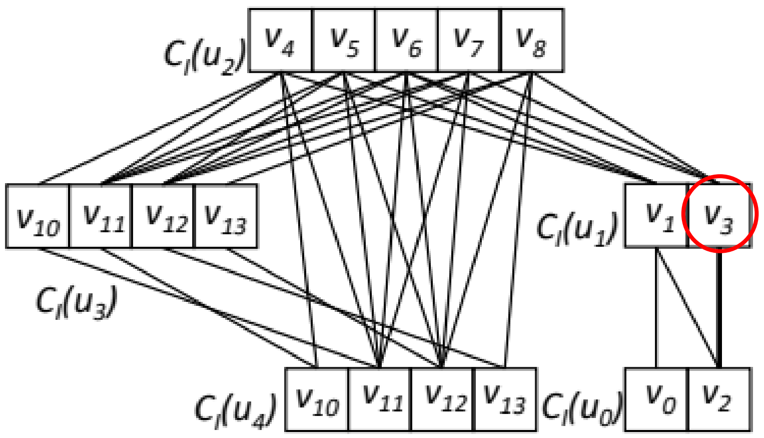
(b) Data graph G.



(c) G': Insert $e(v_2, v_3)$ to G.



(a) Global index on G.



(b) Global index on G'.

Local Index

Algorithm 4: Local Index

1 **Procedure** `BuildLocalIndex` $(Q_R, I, e(u_a, u_b), e(v_a, v_b))$

2   **if** $v_a \notin C_I(u_a)$ *or* $v_b \notin C_I(u_b)$ **then return**;

3   $M \leftarrow \{(u_a, v_a), (u_b, v_b)\}$; // initial mapping

4   $\Phi \leftarrow V(Q_R) \cap (N_Q(u_a) \cup N_Q(u_b))$; // query vertices adjacent to u_a or u_b

5   **foreach** $u \in \Phi$ **do**

6     $C_A(u) \leftarrow \bigcap_{u' \in N_Q(u) \cap \{u_a, u_b\}} I_u^{u'}(M(u')) - \{v_a, v_b\}$;

7   $\delta \leftarrow$ sort vertices $u \in \Phi$ in the ascending order of $|C_A(u)|$;

8   **foreach** $u \in \Phi$ *along the order of* δ **do**

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

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

Lines 7-10 prune candidate sets $C_A(u)$ for $u \in \Phi$ based on the filtering rule: we can remove v from $C_A(u)$ without breaking its completeness if there exists $u' \in N_+^{\delta}(u)$ such that v has no neighbor in $C_A(u')$ where $N_+^{\delta}(u)$ is the set of vertices (which are adjacent to u) positioned before u in a sequence δ of Φ. In particular, δ prioritizes query vertices with fewer candidates to utilize small candidate sets to prune large ones.

u 不可能映射到 v, 如果在 u 之前的某个 u 的邻居 u’ 不能映射到任何 v 的邻居

Local Index

Algorithm 4: Local Index

1 **Procedure** BuildLocalIndex$(Q_R, I, e(u_a, u_b), e(v_a, v_b))$

2 **if** $v_a \notin C_I(u_a)$ *or* $v_b \notin C_I(u_b)$ **then return**;

3 $M \leftarrow \{(u_a, v_a), (u_b, v_b)\}$; // initial mapping

4 $\Phi \leftarrow V(Q_R) \cap (N_Q(u_a) \cup N_Q(u_b))$; // query vertices adjacent to u_a or u_b

5 **foreach** $u \in \Phi$ **do**

6 $C_A(u) \leftarrow \bigcap_{u' \in N_Q(u) \cap \{u_a, u_b\}} I_u^{u'}(M(u')) - \{v_a, v_b\}$;

7 $\delta \leftarrow$ sort vertices $u \in \Phi$ in the ascending order of $|C_A(u)|$;

8 **foreach** $u \in \Phi$ *along the order of* δ **do**

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

10 $C_A(u) \leftarrow C_A(u) \bigcap (\bigcup_{v \in C_A(u')} I_u^{u'}(v))$;

prune candidate set C_A

11 $\overline{\Phi} \leftarrow V(Q_R) - \Phi$; // query vertices NOT adjacent to u_a, u_b

12 **while** $\overline{\Phi} \neq \emptyset$ **do**

13 $u \leftarrow arg\max_{u' \in \overline{\Phi}} |N(u) - \overline{\Phi}|$; // u who has the largest num of neighbors that have candidate sets generated

14 $C_A(u) \leftarrow C_I(u) - \{v_a, v_b\}$;

15 **foreach** $u' \in N(u) - \overline{\Phi}$ **do**

16 Do the same operation as Line 10;

17 Remove u from $\overline{\Phi}$;

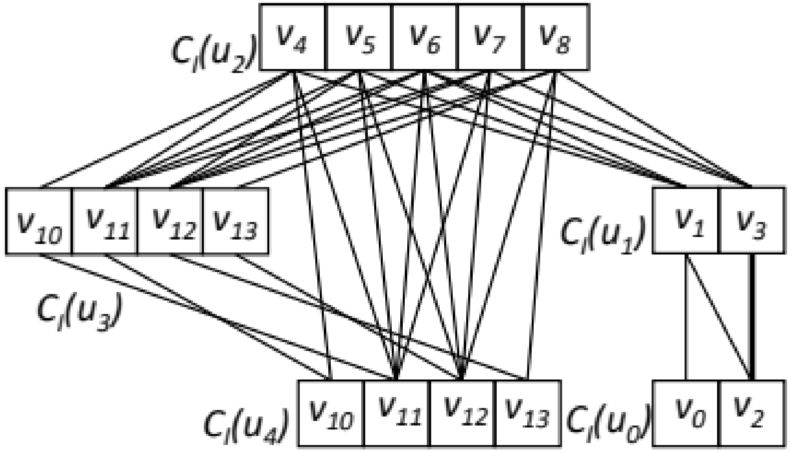
18 **foreach** $e(u, u') \in E(Q_R)$ **do**

19 **foreach** $v \in C_A(u)$ **do**

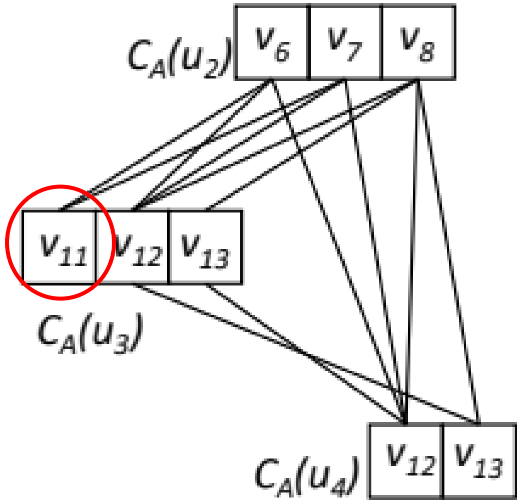
20 $A_{u'}^{u}(v) \leftarrow I_{u'}^{u}(v) \cap C_A(u')$;

21 **return** A; // Finally, we record edges between candidates in $C_A(u)$ and $C_A(u')$ if $e(u, u') \in E(Q_R)$.

Local Index

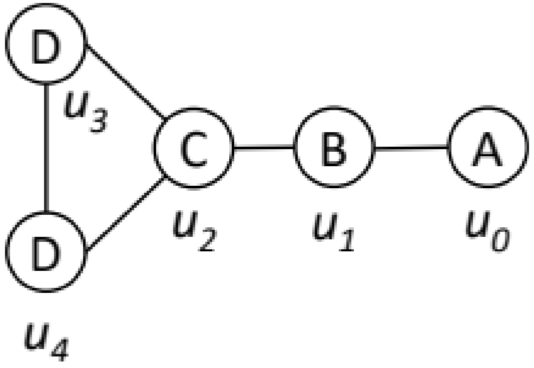


(b) Global index on G'.

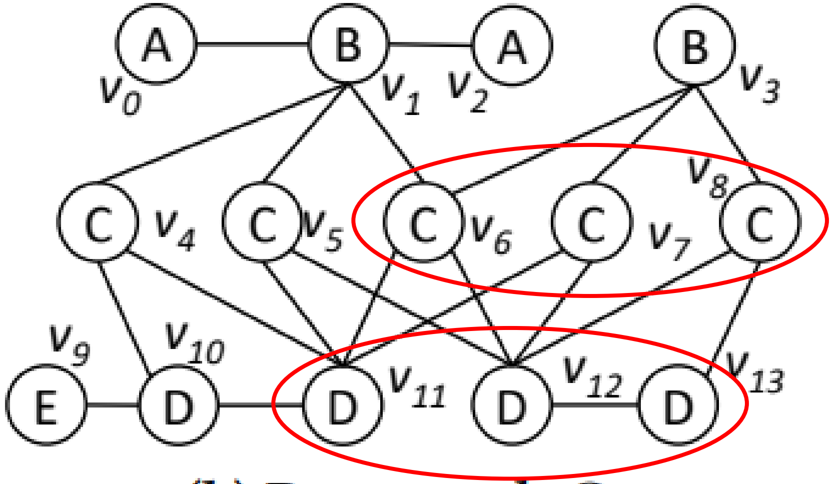


folder

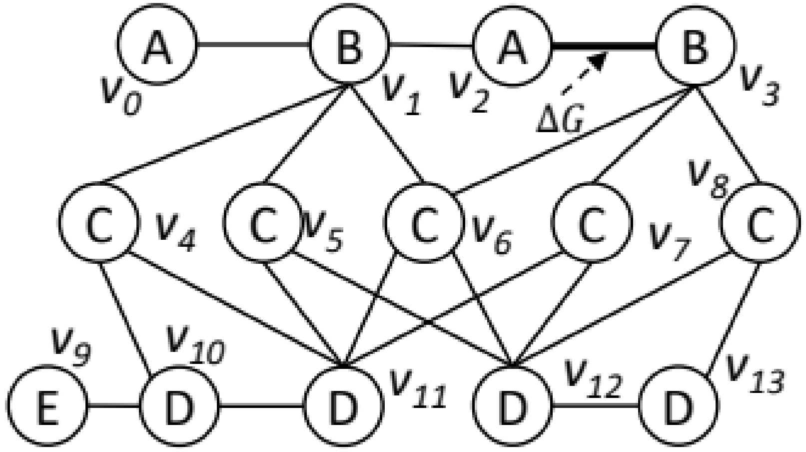
(c) Local index.



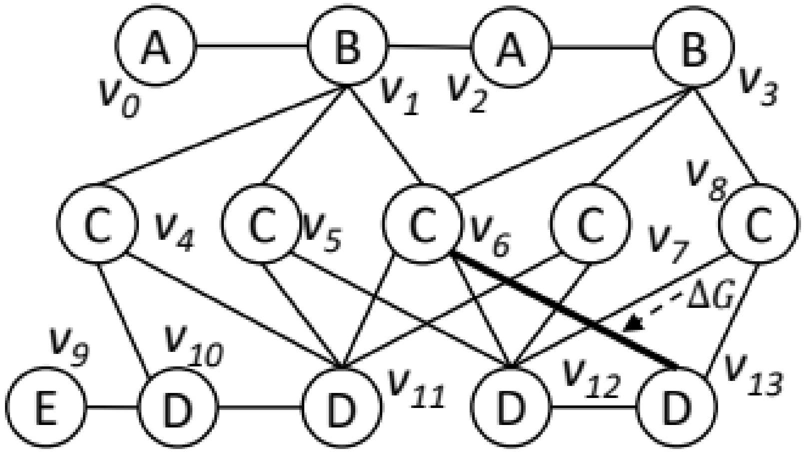
(a) Query graph Q.



(b) Data graph G.



(c) G': Insert $e(v_2, v_3)$ to G.



(d) G'': Insert $e(v_6, v_{13})$ to G'.

Dual Matching

- 假如 Q 存在自同构将边 e 映射到 e'，那么存在 ΔM_e 到 $\Delta M_{e'}$ 的一一对应。

PROPOSITION 5.1. *Given an automorphism M_Q of Q, e denotes $e(u_a, u_b) \in E(Q)$ and e' denotes $e(M_Q(u_a), M_Q(u_b)) \in E(Q)$. Then, $\Delta\mathcal{M}_{e'}$ is equal to $\{M \circ M_Q | M \in \Delta\mathcal{M}_e\}$ where $\circ$ is the function composition operation.*

Dual Matching

Algorithm 5: Dual Matching

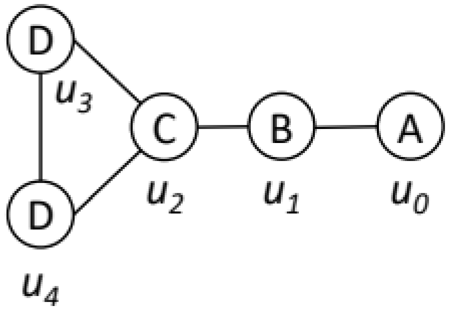
Procedure `GenerateAutoSet` (Q)
    $\mathcal{M}_Q \leftarrow$ find matches of Q in Q;
    $\mathcal{X} \leftarrow \emptyset$;
    foreach $e(u, u') \in E(Q)$ **do**
        if $e(u, u')$ *is not selected* **then**
            $X \leftarrow \emptyset$;
            foreach $M_Q \in \mathcal{M}_Q$ **do**
                if $e(M_Q(u), M_Q(u'))$ *is not selected* **then**
                    $X \leftarrow X \cup \{(e(M_Q(u), M_Q(u')), M_Q)\}$;
                    Mark $e(M_Q(u), M_Q(u'))$ as selected;
            $\mathcal{X} \leftarrow \mathcal{X} \cup \{X\}$;
    return $\mathcal{X}$;

Procedure `DualMatch` ($\Delta\mathcal{M}_e, X$)
    foreach $(e', M_Q) \in X$ **do**
        if $e' \neq e$ **then** $\Delta\mathcal{M}_{e'} \leftarrow \{M \circ M_Q | M \in \Delta\mathcal{M}_e\}$;
    $\Delta\mathcal{M}_X \leftarrow \bigcup_{e \in X} \Delta\mathcal{M}_e$;
    return $\Delta\mathcal{M}_X$;

Dual Matching

Optimization. To further improve the performance, we optimize the procedure of generating matches for auto-sets X. In practice, $\Delta\mathcal{M}_e$ is stored as a table where the header is a sequence of query vertices $(..., u_i, ...)$ and each tuple is a sequence of data vertices. Figure 3 presents an example. Given $e' \in X$ and the corresponding automorphism M_Q, we can generate $\Delta\mathcal{M}_{e'}$ by simply adding another header $(..., M_Q(u_i), ...)$ instead of iterating each match in $\Delta\mathcal{M}_e$. Therefore, the set $\Delta\mathcal{M}_X$ is stored as a table with $|X|$ headers each of which is a sequence of query vertices based on automorphisms of Q.

Example 5.3. Given Q in Figure 1a, $\mathcal{M}_Q = \{M_1 = \{(u_0, u_0), (u_1, u_1), (u_2, u_2), (u_3, u_3), (u_4, u_4)\}, M_2 = \{(u_0, u_0), (u_1, u_1), (u_2, u_2), (u_3, u_4), (u_4, u_3)\}\}$. The set $\mathcal{X}$ of auto-sets is $\{X_1 = \{(e(u_0, u_1), M_1)\}, X_2 = \{(e(u_1, u_2), M_1)\}, X_3 = \{(e(u_3, u_4), M_1)\}, X_4 = \{(e(u_2, u_3), M_1), (e(u_2, u_4), M_2)\}\}$. Given insertion of $e(v_6, v_{13})$, suppose that we obtain $\Delta\mathcal{M}_{e(u_2,u_3)}$ in Figure 3. As $e(u_2, u_3) \in X_4$, the dual matching technique generates $\Delta\mathcal{M}_{X_4}$ by adding a header based on M_2. The results are shown in Figure 5. Thus, we do not need to execute a search procedure for $e(u_2, u_4)$.



u_0	u_1	u_2	u_3	u_4
v_0	v_1	v_6	v_{13}	v_{12}
v_2	v_1	v_6	v_{13}	v_{12}
v_2	v_3	v_6	v_{13}	v_{12}

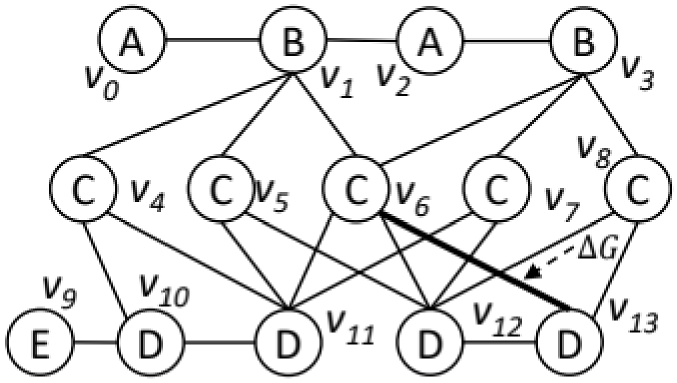
$\Delta\mathcal{M}_{e(u_2,u_3)}$

The same table content

u_0	u_1	u_2	u_4	u_3
v_0	v_1	v_6	v_{13}	v_{12}
v_2	v_1	v_6	v_{13}	v_{12}
v_2	v_3	v_6	v_{13}	v_{12}

$\Delta\mathcal{M}_{e(u_2,u_4)}$

Figure 3: Incremental matches generated by existing CSM methods given insertion of $e(v_6, v_{13})$ to G' in Figure 1d.



(d) G'': Insert $e(v_6, v_{13})$ to G'.

How to generate a good matching order

- RI