CS729
Efficient Subgraph Matching

Professor: Rivero Osuna, Carlos Rafael

Team # 2
Gevaria, Harnisha (hgg5350@g.rit.edu)
Gevaria, Kushal (kgg5247@rit.edu)
Ku, Wei-Yao (wxk6489@rit.edu)

Agenda

SPath

- SPath query ordering with respect to GraphQL/VF2 Plus

CPI

- Query decomposition
- Combination of decomposition and CPI construction
- CPI construction in Neo4j
- Experimental results
- Lessons learned

SPath

- A new graph indexing technique towards resolving the graph query problem efficiently on large networks.

- It maintains for each query vertex a neighborhood signature – a structure that stores decomposed shortest path information within the vertex's vicinity.

- SPath, revolutionizes the way of graph query processing from vertex-at-a time to path-at-a-time, which proves to be more cost effective than traditional graph matching methods.

SPath

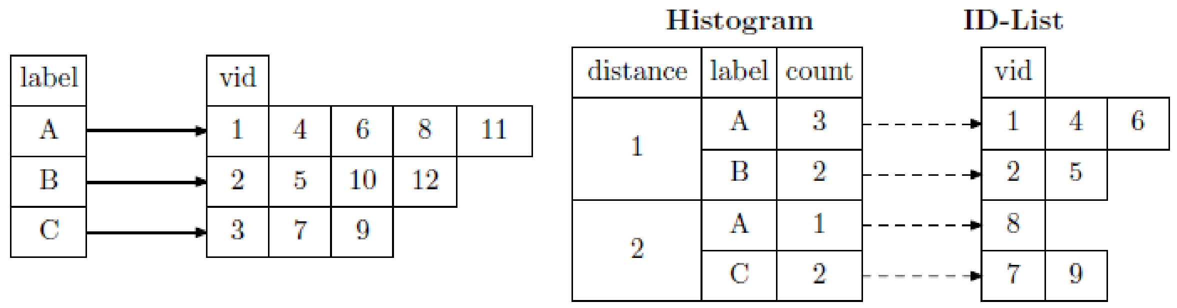
The process is as follows:

- Decompose a query graph into a set of shortest paths, among which a subset of candidate paths with high selectivity is picked by a graph query optimizer.
- The query is further processed by joining candidate shortest paths in order to reconstruct the original query graph.

The principle of SPath is to use shortest paths within the k-neighborhood subgraph of each vertex of the graph to capture the local structural information around the vertex.

SPath

The structure used to store SPath:



label	vid				
A	1	4	6	8	11
B	2	5	10	12	
C	3	7	9		

(a) Global Lookup Table

Histogram

distance	label	count
1	A	3
	B	2
2	A	1
	C	2

ID-List

vid		
1	4	6
2	5	
8		
7	9	

(b) Histogram and ID-List

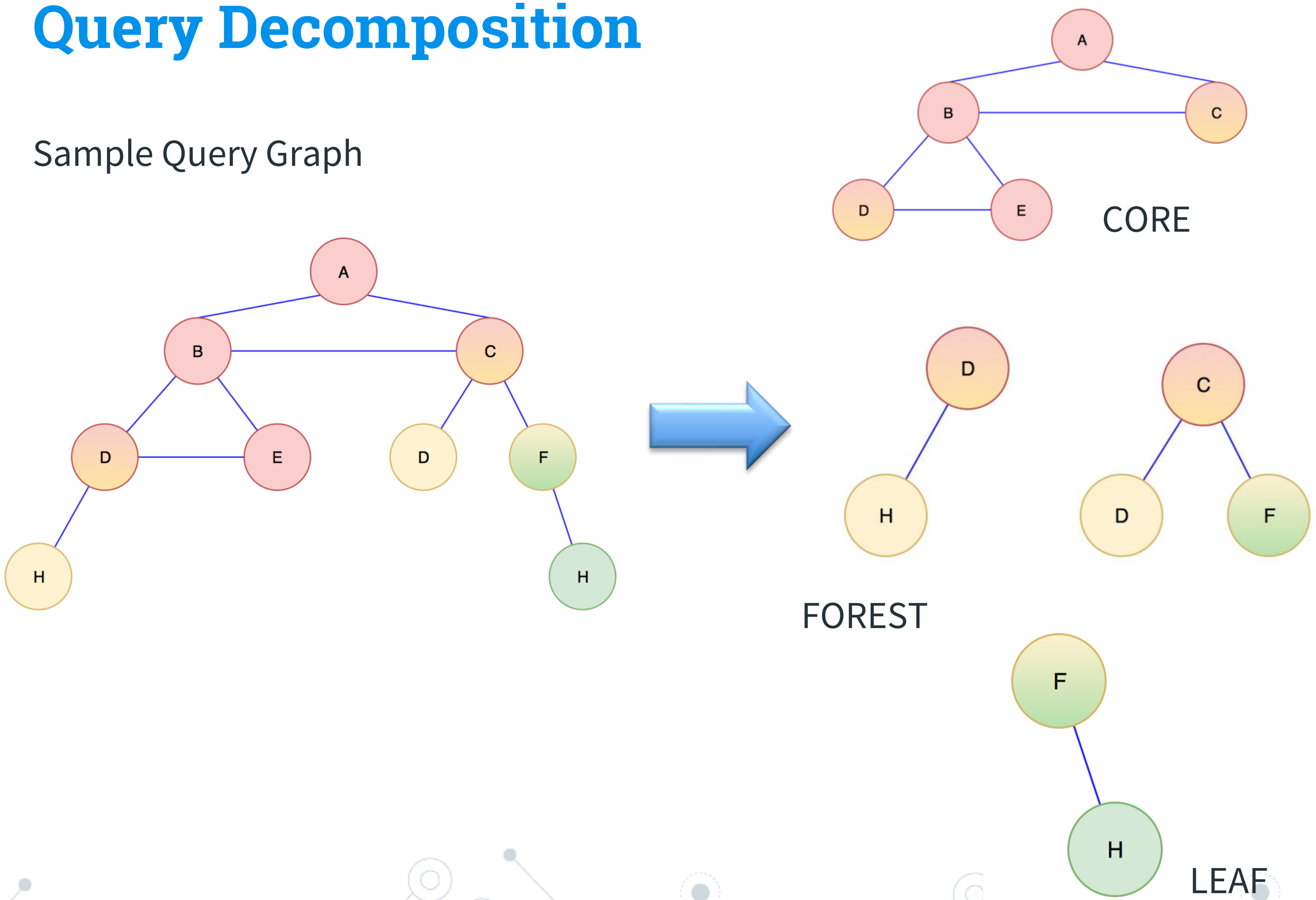
Figure 2: The Global Lookup Table $\mathcal{H}$ and the Histogram and ID-List of $NS(u_3)$, $u_3 \in V(G)$ $(k_0 = 2)$

SPath

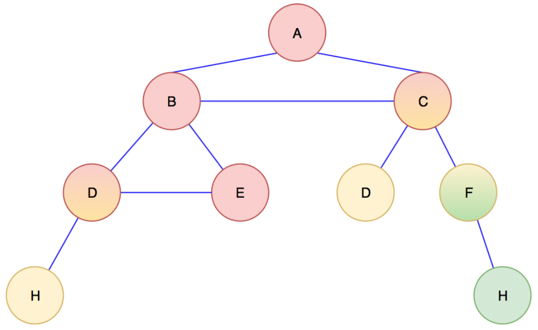
- They claim SPath is the first scalable graph indexing mechanism which supports effective path-at-a-time graph query processing on large networks, and thus achieves far better query performance, compared with other traditional vertex-at-a-time graph matching methods.

Query Decomposition

Sample Query Graph



Query Decomposition

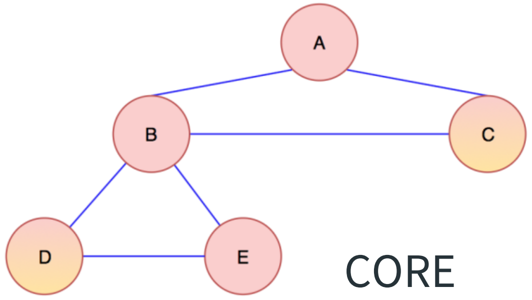


QUERY

Core-Forest Decomposition

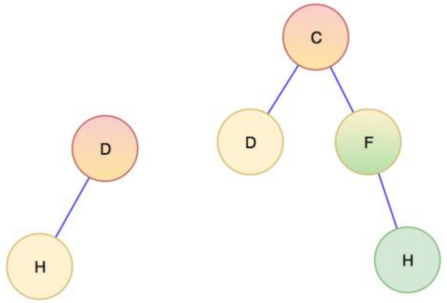
As mentioned in the paper,

1. Iteratively delete all nodes with degree 1
2. Remaining nodes represent CORE



CORE

REMAINING FOREST

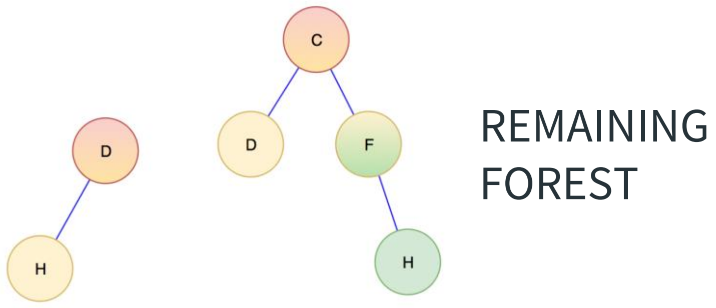


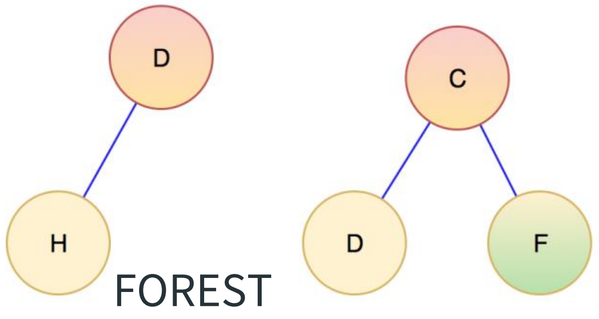
Query Decomposition

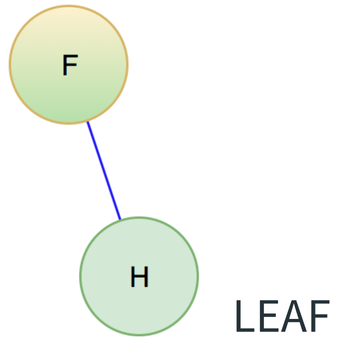
Forest-Leaf Decomposition

Different approach,

1. Leaf nodes have only one neighbor, thus degree of leaf node should be 1
2. Also, leaf node or its neighbor(parent) should not be the part of CORE. This, can be confirmed by looking at the degree of these two nodes
3. Now, if the neighbor is not a part of the Core, then its degree should be less than or equal to 2

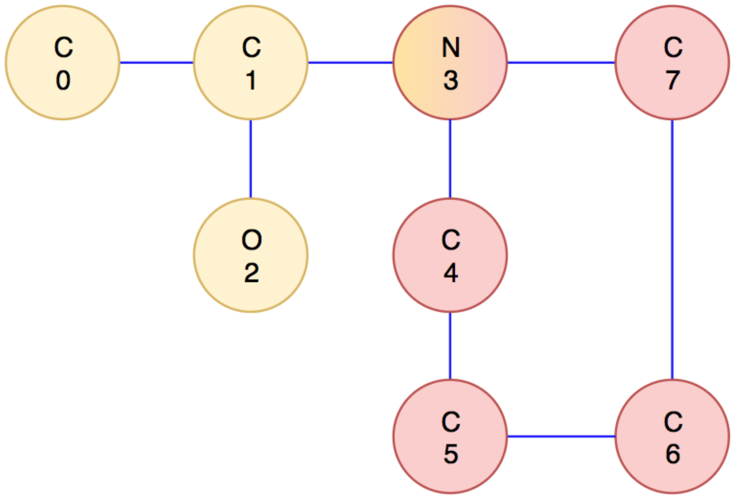


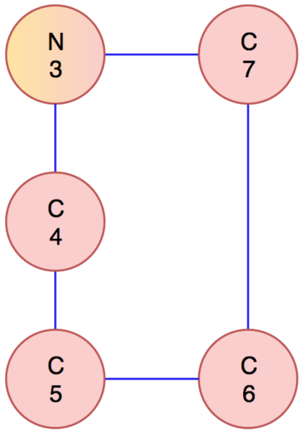




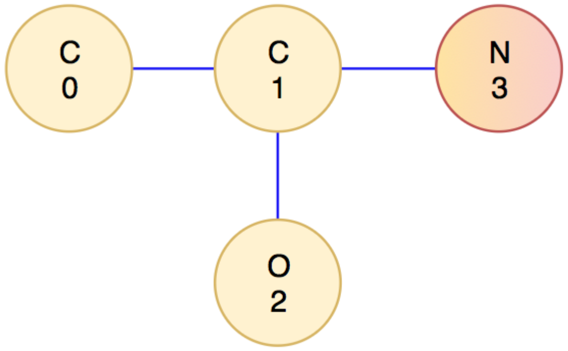
Query Decomposition

Query graph from proteins database - "ecoli_1RF8.8.sub.grf"





CORE

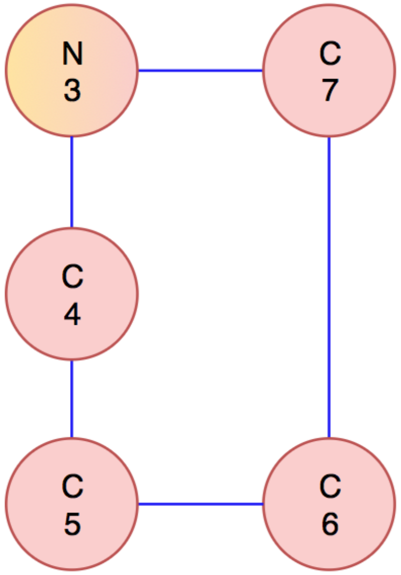


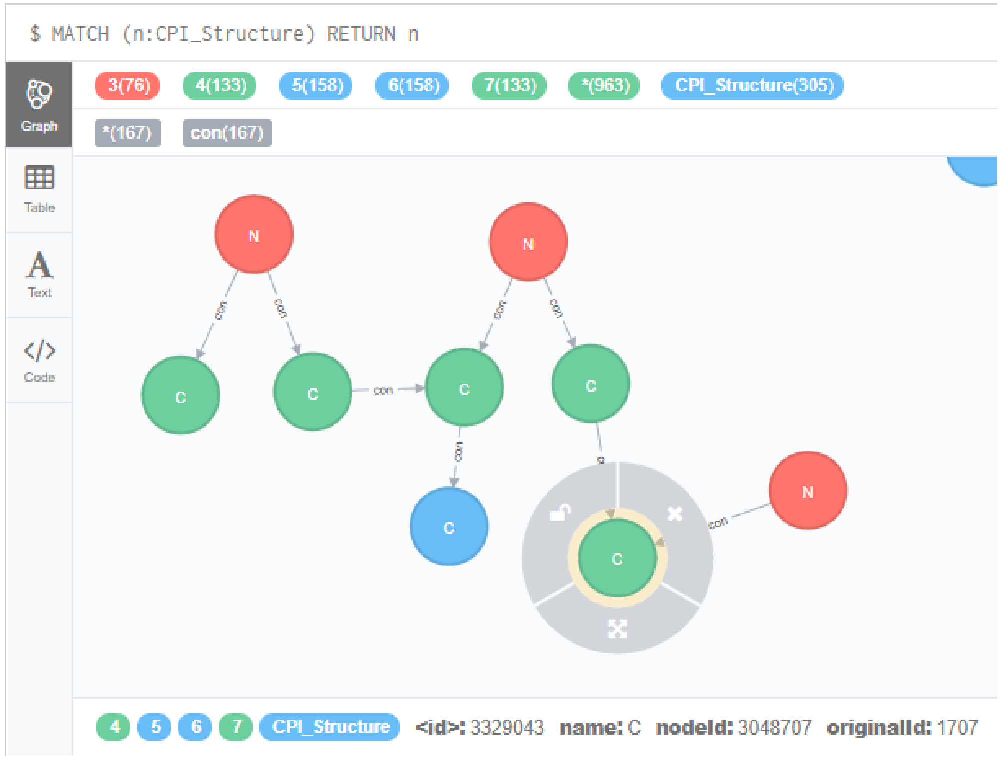
FOREST

NO LEAF

Combination of Decomposition and CPI Construction

- CPI for CORE

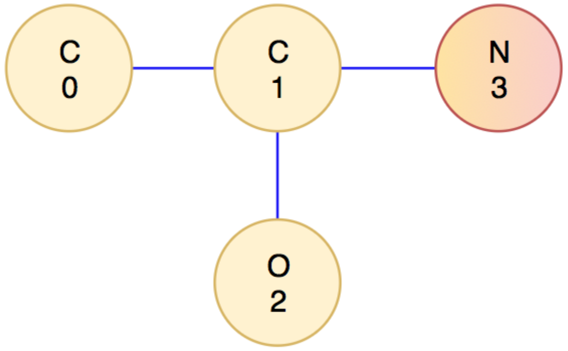


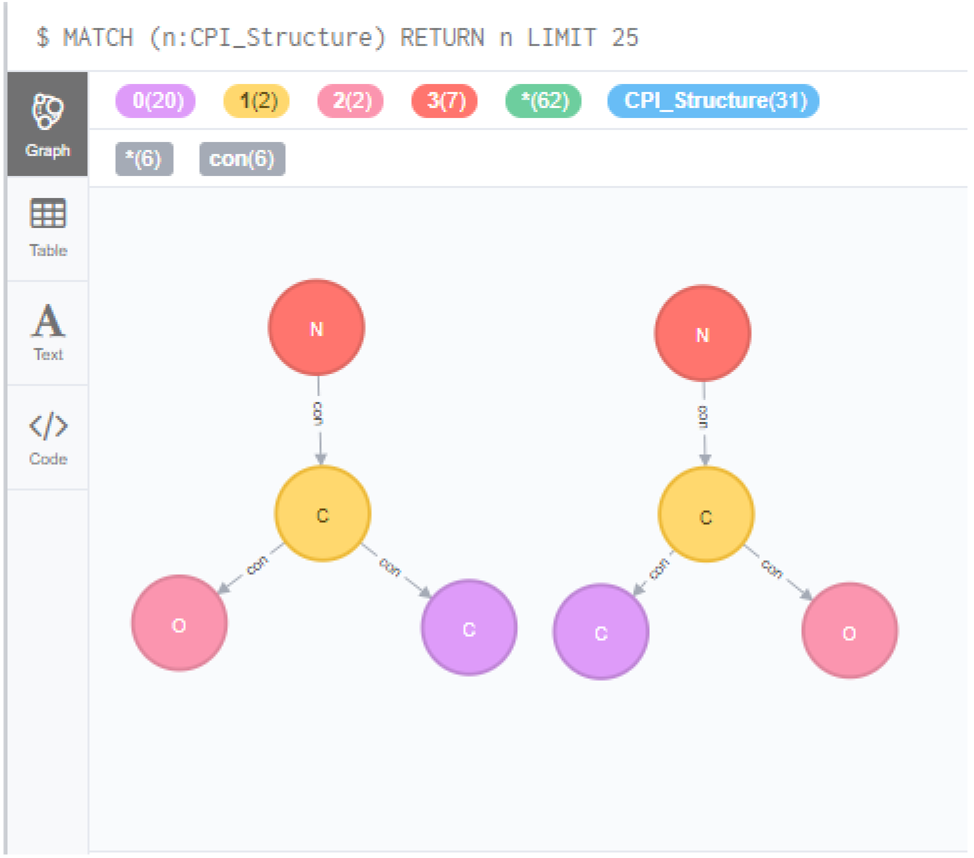

```
$ MATCH (n:CPI_Structure) RETURN n
```

4 5 6 7 CPI_Structure <id>: 3329043 name: C nodeId: 3048707 originalId: 1707

Combination of Decomposition and CPI Construction

- CPI for FOREST





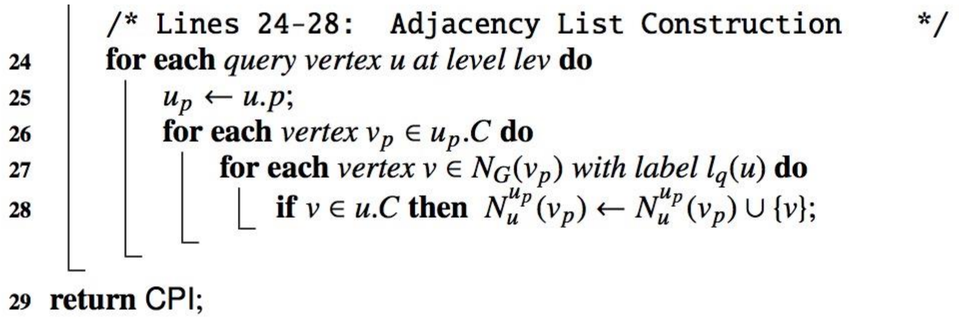
```
$ MATCH (n:CPI_Structure) RETURN n LIMIT 25
```

CPI construction in Neo4j

Top Down CPI construction:

- Directly generate node in neo4j with following properties:
  Label: CPI_Structure, 0
  <id>: 3290308,
  Property: name:N, nodeId: 3047906, originalId: 906

- Check as line 26-28, add edge between node if not present



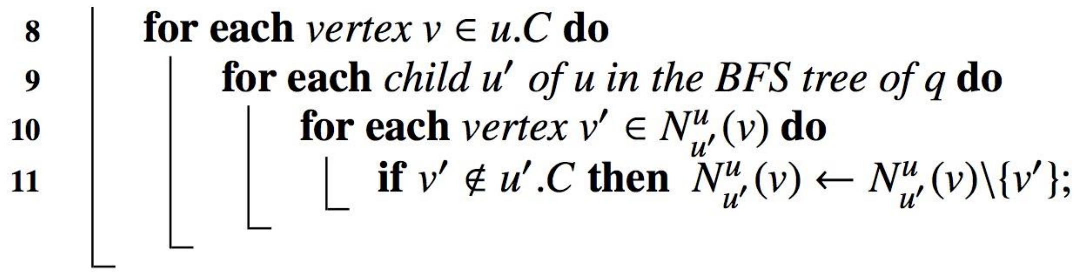
```
   /* Lines 24-28:  Adjacency List Construction    */
24 for each query vertex $u$ at level $lev$ do
25     $u_p \leftarrow u.p$;
26     for each vertex $v_p \in u_p.C$ do
27         for each vertex $v \in N_G(v_p)$ with label $l_q(u)$ do
28             if $v \in u.C$ then $N_u^{u_p}(v_p) \leftarrow N_u^{u_p}(v_p) \cup \{v\}$;

29 return CPI;
```

CPI construction in Neo4j

Bottom-Up Refinement :

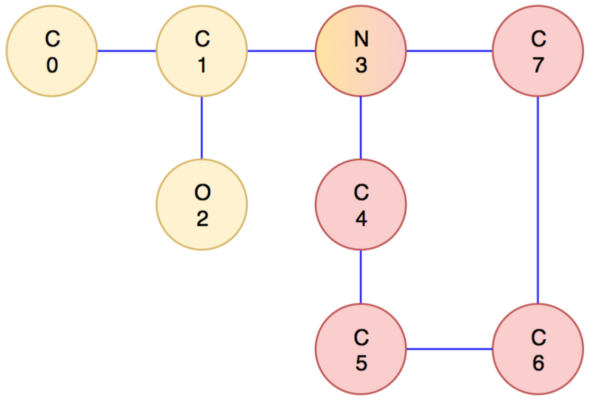
- Remove/update graph in Neo4J (Remove edge, Remove node)

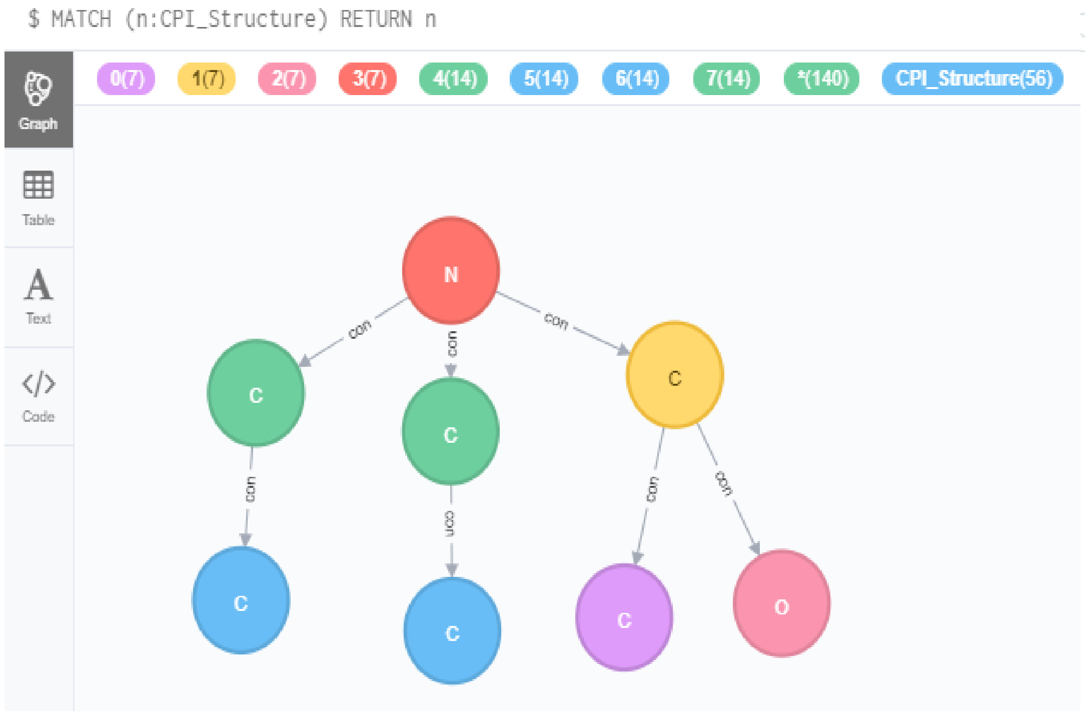


```
for each vertex $v \in u.C$ do
    for each child $u'$ of $u$ in the BFS tree of $q$ do
        for each vertex $v' \in N^{u}_{u'}(v)$ do
            if $v' \notin u'.C$ then $N^{u}_{u'}(v) \leftarrow N^{u}_{u'}(v) \setminus \{v'\}$;
```

CPI construction in Neo4j

Query graph from proteins database "ecoli_1RF8.8.sub.grf":





```
$ MATCH (n:CPI_Structure) RETURN n
```

Experimental Results

Target Graph (5):		Query Graph (60):
backbones_140L.grf backbones_1NOD.grf backbones_2X3T.grf human_1B9G human_2KSA.grf	X	Protein 8 Protein 16 Protein 32 Protein 64 Protein 128 Protein 256

- 50 query per protein size
- Total 300 query
- Get the average time (Min)

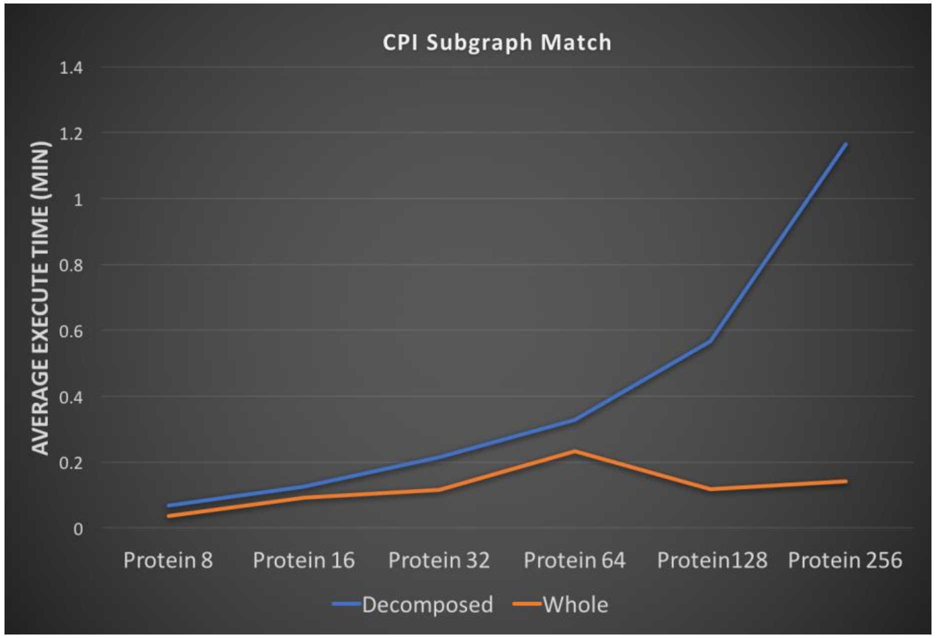
Experimental Results

Decomposed	Protein 8	Protein 16	Protein 32	Protein 64	Protein 128	Protein 256
Max (Min)	0.2547	0.3771	0.6458	0.7971	1.9941	3.7188
Min (Min)	0.1130	0.0198	0.0240	0.0293	0.0301	0.0768

Whole	Protein 8	Protein 16	Protein 32	Protein 64	Protein 128	Protein 256
Max (Min)	0.1384	0.2758	0.39365	0.6909	1.0577	1.7942
Min (Min)	0.0143	0.0170	0.0126	0.0167	0.0107	0.0227

Experimental Results

All subset match between with the ground truth success !!



Averager time (Min)	Protein 8	Protein 16	Protein 32	Protein 64	Protein 128	Protein 256
Decomposed	0.0675	0.1269	0.2161	0.3284	0.567	1.1651
Whole	0.0362	0.0924	0.1159	0.2340	0.1178	0.1413

Lessons Learned

- **Different way to improving subgraph match**. But we can't really compare CPI performant with the previous algorithm like VF2 plus because we didn't perform the framework matching and the best search ordering .
- **Great team work.** Great teamwork experiment during this final project. Help each other to focus and avoid confusing by the complex algorithm.
- **Paper digest and implementation.** Learn how to digest paper and implement the algorithm step by step.

References

- Bi F, Chang L, Lin X, Qin L and Zhang W. Efficient Subgraph Matching by Postponing Cartesian Products in International Conference on Management of Data, 1199-1214, July 2016.
- P. Zhao and J. Han. On graph query optimization in large networks. PVLDB, 3(1), 2010.

Thanks You!

Any questions?