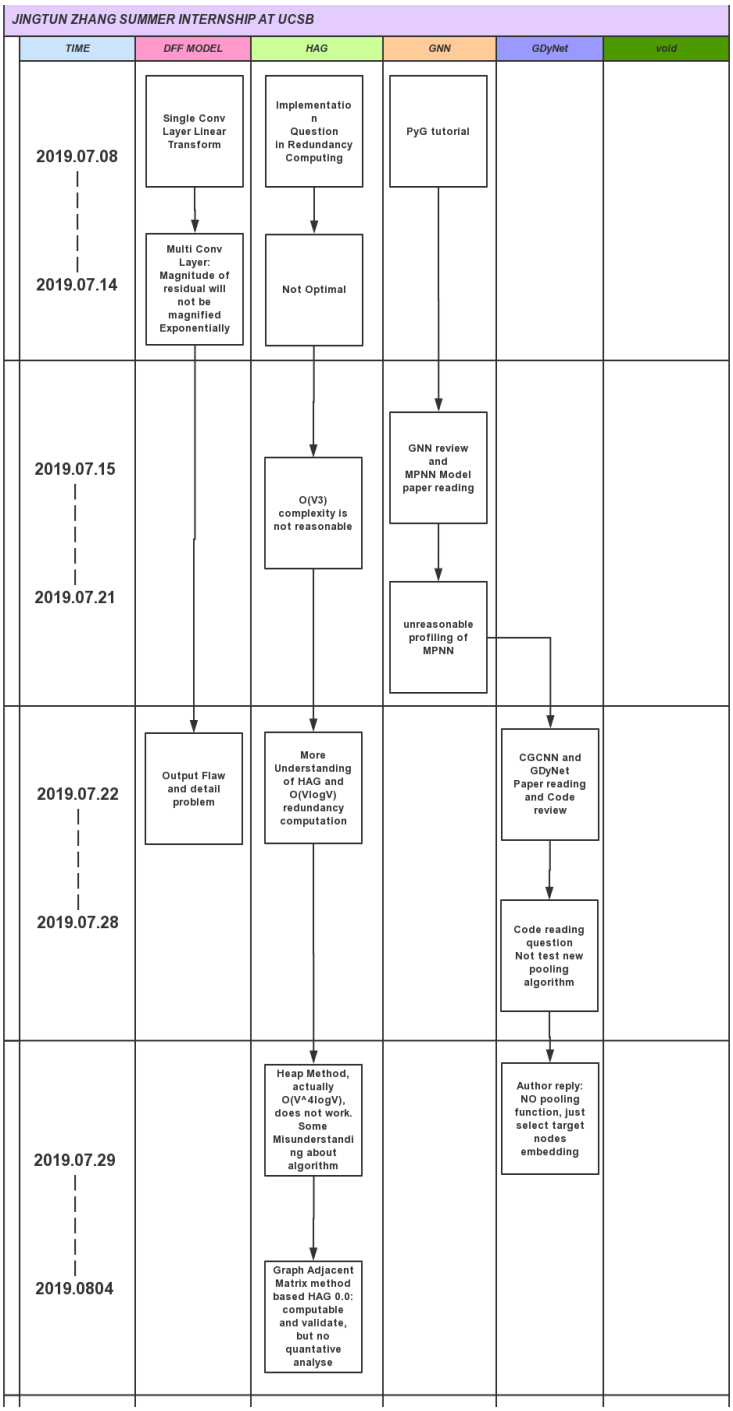
Weekly Report 2019.07.22-2019.07.28

> Jingtun ZHANG

WHERE WE ARE:



Work and Progress

1. Reimplementation of HAG:
    1. release HAG model Version 0.0

system configuration:

- operating system: Ubuntu 19.04
- python: Python 3.7.3 [GCC 8.3.0] on linux
- packets: **numpy 1.17.0rc2**, **scipy 1.3.0**, **torch 1.1.0**

model:

class HAG(x, edge_index, ha_proportion=0.25, redundancy_threshold=1, aggr='add', flow='source_to_target'):

attributes:

- **h [torch.tensor shape (V + V_A, d)]**: hag nodes (original nodes and aggregated nodes) embedding
- **x [torch.tensor shape (V, d)]** : original nodes embedding
- **V [scalar]**: number of original nodes
- **edge_index [torch.tensor COO format shape (2 * E, 2)]**: coo format edges of hag
- **capacity [scalar]**: max number of aggregated nodes
- **redundancy_threshold [scalar]**: min redundancy to eliminate
- **aggr [str 'add' or 'mean' or 'max']**: aggregation scheme to use
- **flow [str 'source_to_target' or 'target_to_source']**: flow direction of message passing

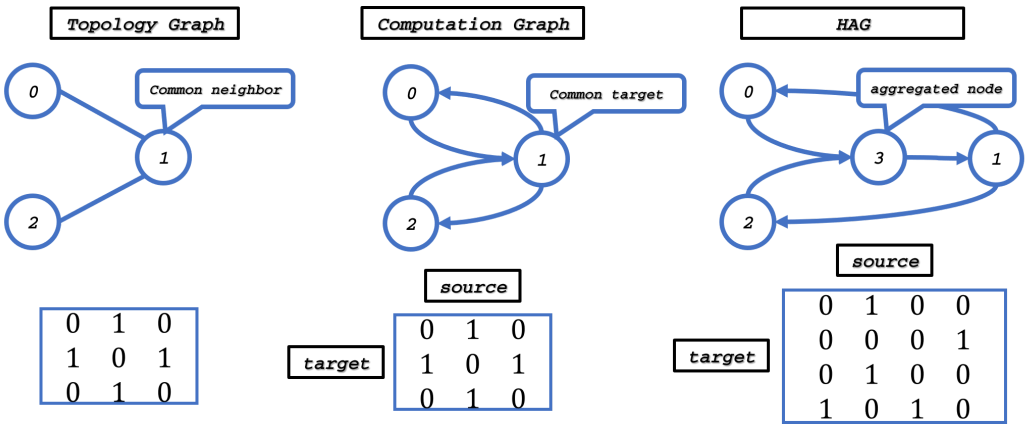
methods:

- **graph_to_hag()**: build HAG at preprocessing stage
- **hag_aggregate()**: compute embedding of aggregated nodes every iteration(layer)
- **max_redundancy()**: find max redundancy node pair and return it

2. detailed description about max redundancy computation:

- object:

***KEY POINT**: computation graph (directed graph) is different from topology graph (directed or undirected graph) and our object is to eliminate redundancy in computation graph*



- every iteration new node-pair heap building:
  - one node pair redundancy computation: O(V)
  - all node pairs redundancy computation: O(V^2) * O (V)
  - heap building: O(logV)
  - capacity (iterations): O(V)

        - overall: O(V) * (O(V^3) + O(logV)) = O(V^4)
        - Not executable
    - graph adjacent matrix method (implemented by scipy.sparse API, have not profiling now) (*can get CUDA parallization in future*)
        - common targets number = number of two-step roads with the first step source-to-target and the second step target-to-source = AT.dot(A) ~ O(V^2.7)
    - update of aggregated nodes
        - aggregate only on aggregated nodes for log2(capacity) = log2(O(V)) times thus all aggregated nodes get new embedding of this layer (not need for the first layer)

This week plan

1. paper reading for idea:
    1. Graph data Processing
    2. redundancy elimination of computation graph
    3. GNN models
2. optimization of HAG Code
