南京大學
NANJING UNIVERSITY

PASA Lab

Paper Read：Accurate efficient and scalable graph embedding

报告人：王云攀
时间：2019.12.6

汇报内容

1. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2019). Accurate, efficient and scalable graph embedding. *IPDPS*, 462–471. https://doi.org/10.1109/IPDPS.2019.00056

Abstract

Content

- Background
  - The graph convolutional network model and its variants are powerful graph embedding tools for facilitating classification and clustering on graphs
- Challenge:
  - Reduce the complexity for layered GCNs and make them parallelizable and scalable on very large graphs – state-of the art techniques are unable to achieve scalability without losing accuracy and efficiency.
  - Propose novel parallelization techniques for graph sampling-based GCNs that achieve superior scalable performance on very large graphs without compromising accuracy.
  - Our GCN guarantees work-efficient training and produces order of magnitude savings in computation and communication.

Abstract

Content

- Key Ideas:
  - For the graph sampling step, we exploit parallelism within and across multiple sampling instances, and devise an efficient data structure for concurrent accesses that provides theoretical guarantee of near-linear speedup with number of processing units.
  - For the feature propagation step within the sampled graph, we improve cache utilization and reduce DRAM communication by data partitioning.
- Result
  - Prove that out partitioning strategy is a 2-approximation for minimizing the communication time compared to the optimal strategy.
  - Outperforms state-of-the-art methods in scalability(with respect to number of processors, graph size and GCN model size), efficiency and accuracy on several large datasets.
  - Our parallel training achieves 64X speedup in the sampling step and 25X speedup in the feature propagation step, compared to the serial implementation, resulting in a net speedup of 21X.
  - Our scalable algorithm enables deeper GCN, as demonstrated by 1306X speedup on a 3-layer GCN compared to Tensorflow implementation of state-of-the-art

I Introduction

A. Previous work[1,2,3]

- Background
  - GCN-based approaches produce accurate and robust results without the need of manual feature selection.
- Ideas
  - The typical approach is to decompose training into "mini-batches" and attempt to parallelize mini-batch training by sampling on GCN layers.
    - 1stChebNet and GraphSAGE sample the inter-layer edge connections. But they increases by a factor of node degree for every additional layer of GCN
    - FastGCN propose to sample the nodes of GCN layers instead of the edge connections. Faster, but not guarantee work-efficiency.
- Problems
  - Due to the layer sampling design philosophy, it is difficult for state-of-the-art methods to simultaneously achieve accuracy, efficiency and scalability.

I Introduction

PASA Lab

B Present work

- Solutions
  - We propose a new graph sampling-based GCN which achieves superior performance on variety of large graphs.
    - Based on parallelized graph sampling, without compromising accuracy or efficiency.
    - Achieve scalability and performance
      - Developing a novel data structure that efficient parallel subgraph sampling through fast parallel updates to degree distributions.
      - Developing an optimized parallel implementation of intra-subgraph feature propagation and weight updates—specifically a cache-efficient partitioning scheme that reduces DRAM communication, resulting in optimized memory performance.
    - Achieve work-efficiency
      - Avoiding neighbor explosion, as each layer of our complete GCN consists of only small set of nodes.
    - Achieve accuracy
      - Since our sampled subgraphs preserve connectivity characteristics of the original training graph

I Introduction

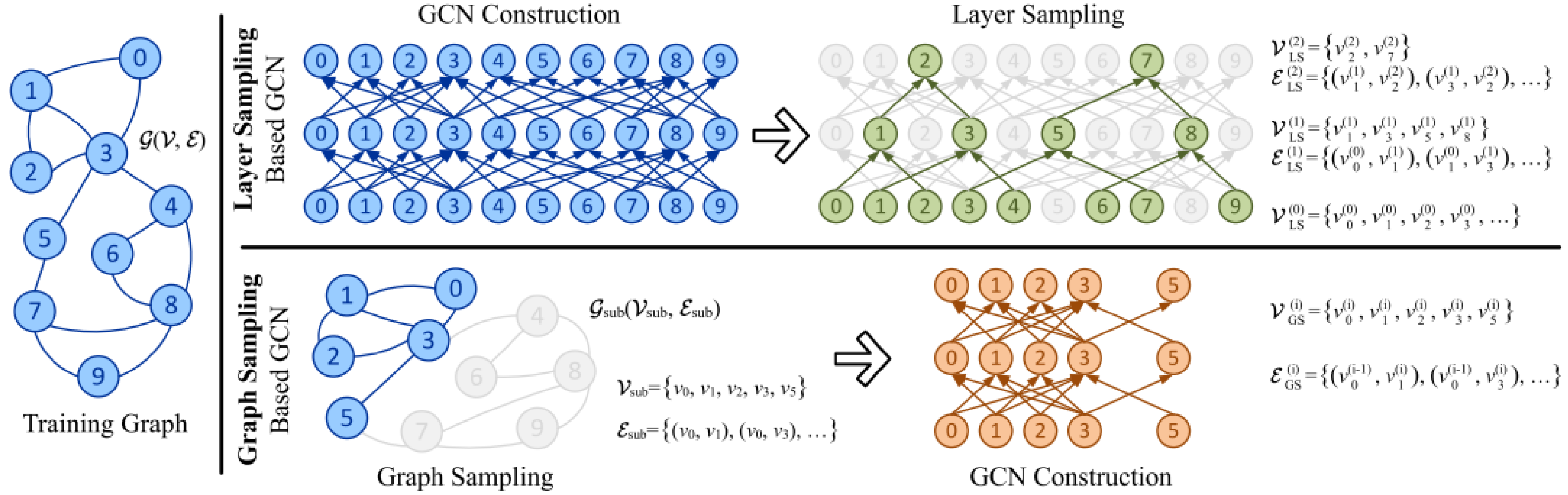
B Present work

- Contributions
  - Parallel training algorithm for a novel graph sampling-based GCN model, where:
    - Accuracy is achieved due to connectivity-preserving
    - Efficiency is optimal since “neighbor explosion” in layer sampling based GCNs is avoided
    - Scalability is achieved with respect to number of processing cores, graph size and GCN depth by extracting parallelism at various key steps
  - A novel data structure that accelerate degree distribution based sampling through fast incremental parallel updates. Near-linear scalability.
  - Parallel The key training step of subgraph feature propagation. Using intelligent partitioning on the feature dimension, we optimize DRAM and cache performance and scale training to large number of cores.
  - Perform through evaluation on a 40-core Xeon server. Compared with serial implementation, 21X overall training time speedup. Compared with other GCN based methods, our parallel implementation 37.4X training time reduction when reaching the same accuracy.
  - Demonstrate that our implementation can enable deeper GCN. We obtain 1306X speedup for 2-layer GCN compared to Tensorflow implementation of state-of-the-art on a 40-core machine

II Background and Related Work

PASA Lab

Content



GCN Construction
Layer Sampling
Layer Sampling Based GCN
Graph Sampling Based GCN
$\mathcal{G}(\mathcal{V}, \mathcal{E})$
Training Graph
$\mathcal{V}_{LS}^{(2)} = \{v_2^{(2)}, v_7^{(2)}\}$
$\mathcal{E}_{LS}^{(2)} = \{(v_1^{(1)}, v_2^{(2)}), (v_3^{(1)}, v_2^{(2)}), \ldots\}$
$\mathcal{V}_{LS}^{(1)} = \{v_1^{(1)}, v_3^{(1)}, v_5^{(1)}, v_8^{(1)}\}$
$\mathcal{E}_{LS}^{(1)} = \{(v_0^{(0)}, v_1^{(1)}), (v_1^{(0)}, v_3^{(1)}), \ldots\}$
$\mathcal{V}_{LS}^{(0)} = \{v_0^{(0)}, v_1^{(0)}, v_2^{(0)}, v_3^{(0)}, \ldots\}$
$\mathcal{G}_{sub}(\mathcal{V}_{sub}, \mathcal{E}_{sub})$
$\mathcal{V}_{sub} = \{v_0, v_1, v_2, v_3, v_5\}$
$\mathcal{E}_{sub} = \{(v_0, v_1), (v_0, v_3), \ldots\}$
Graph Sampling
GCN Construction
$\mathcal{V}_{GS}^{(i)} = \{v_0^{(i)}, v_1^{(i)}, v_2^{(i)}, v_3^{(i)}, v_5^{(i)}\}$
$\mathcal{E}_{GS}^{(i)} = \{(v_0^{(i-1)}, v_1^{(i)}), (v_0^{(i-1)}, v_3^{(i)}), \ldots\}$

Fig. 1: Illustration on layer sampling and graph sampling based GCN design.

- Layer Sampling: Previous work
- Graph Sampling: Present work

III Graph Sampling-based GCN Model

A. Design of the Model

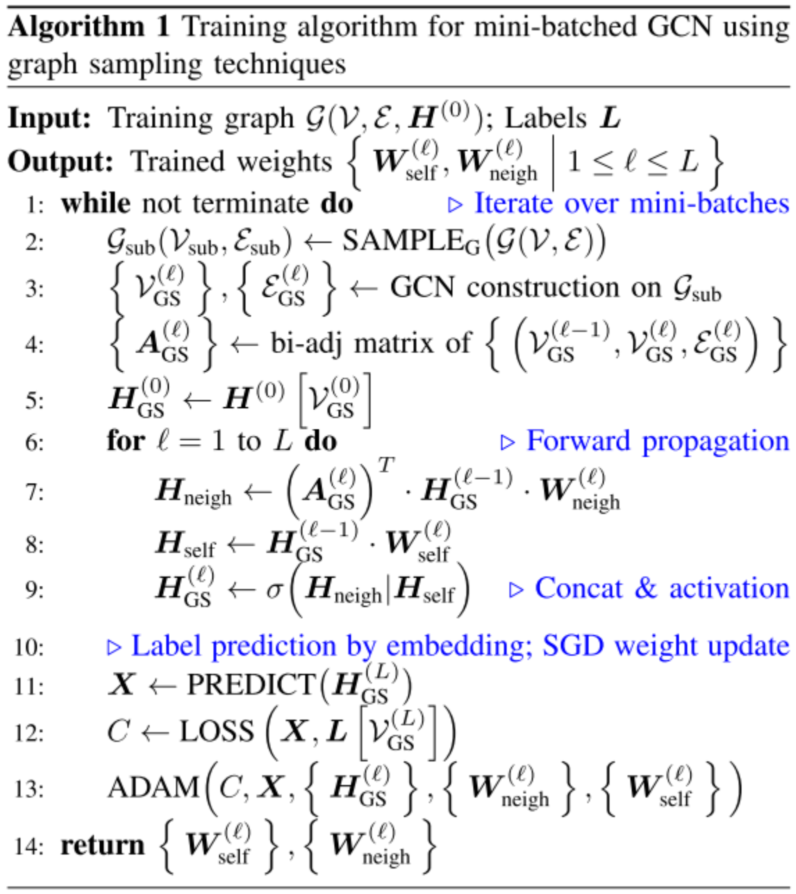
B. Complexity of Graph Sampling-Based GCN

C. Accuracy of Graph Sampling-Based GCN

III Graph Sampling-based GCN Model

PASA Lab

A. Design of the Model



Algorithm 1 Training algorithm for mini-batched GCN using graph sampling techniques

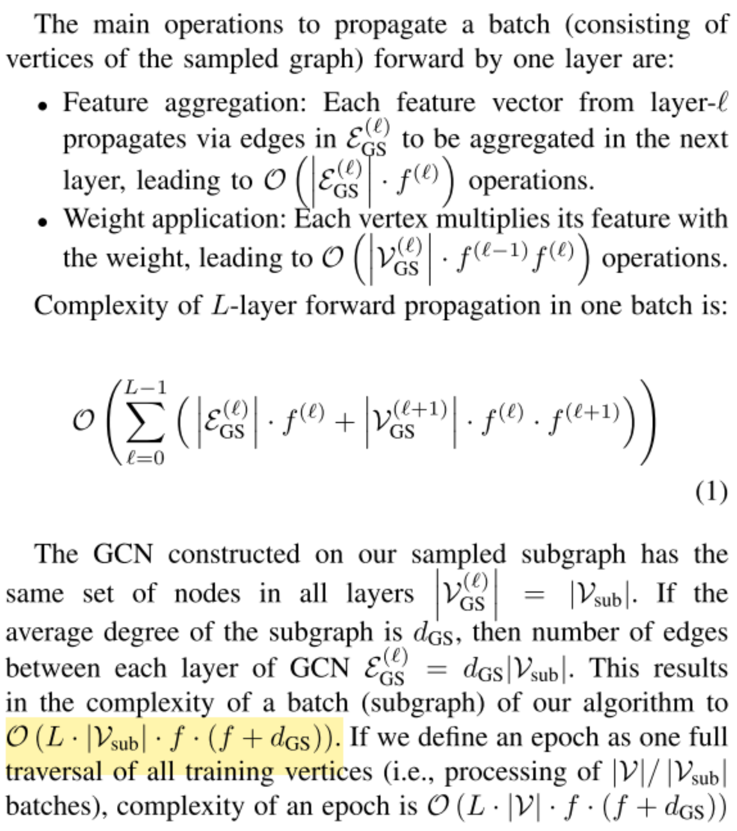
```
Input: Training graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{H}^{(0)})$; Labels $\boldsymbol{L}$
Output: Trained weights $\left\{ \boldsymbol{W}_{\text{self}}^{(\ell)}, \boldsymbol{W}_{\text{neigh}}^{(\ell)} \;\middle|\; 1 \leq \ell \leq L \right\}$
1: while not terminate do                                  ▷ Iterate over mini-batches
2:     $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}}) \leftarrow \text{SAMPLE}_{\text{G}}(\mathcal{G}(\mathcal{V}, \mathcal{E}))$
3:     $\left\{ \mathcal{V}_{\text{GS}}^{(\ell)} \right\}, \left\{ \mathcal{E}_{\text{GS}}^{(\ell)} \right\} \leftarrow$ GCN construction on $\mathcal{G}_{\text{sub}}$
4:     $\left\{ \boldsymbol{A}_{\text{GS}}^{(\ell)} \right\} \leftarrow$ bi-adj matrix of $\left\{ \left( \mathcal{V}_{\text{GS}}^{(\ell-1)}, \mathcal{V}_{\text{GS}}^{(\ell)}, \mathcal{E}_{\text{GS}}^{(\ell)} \right) \right\}$
5:     $\boldsymbol{H}_{\text{GS}}^{(0)} \leftarrow \boldsymbol{H}^{(0)} \left[ \mathcal{V}_{\text{GS}}^{(0)} \right]$
6:     for $\ell = 1$ to $L$ do                             ▷ Forward propagation
7:         $\boldsymbol{H}_{\text{neigh}} \leftarrow \left( \boldsymbol{A}_{\text{GS}}^{(\ell)} \right)^{T} \cdot \boldsymbol{H}_{\text{GS}}^{(\ell-1)} \cdot \boldsymbol{W}_{\text{neigh}}^{(\ell)}$
8:         $\boldsymbol{H}_{\text{self}} \leftarrow \boldsymbol{H}_{\text{GS}}^{(\ell-1)} \cdot \boldsymbol{W}_{\text{self}}^{(\ell)}$
9:         $\boldsymbol{H}_{\text{GS}}^{(\ell)} \leftarrow \sigma \left( \boldsymbol{H}_{\text{neigh}} | \boldsymbol{H}_{\text{self}} \right)$          ▷ Concat & activation
10:    ▷ Label prediction by embedding; SGD weight update
11:    $\boldsymbol{X} \leftarrow \text{PREDICT}(\boldsymbol{H}_{\text{GS}}^{(L)})$
12:    $C \leftarrow \text{LOSS} \left( \boldsymbol{X}, \boldsymbol{L} \left[ \mathcal{V}_{\text{GS}}^{(L)} \right] \right)$
13:    $\text{ADAM} \left( C, \boldsymbol{X}, \left\{ \boldsymbol{H}_{\text{GS}}^{(\ell)} \right\}, \left\{ \boldsymbol{W}_{\text{neigh}}^{(\ell)} \right\}, \left\{ \boldsymbol{W}_{\text{self}}^{(\ell)} \right\} \right)$
14: return $\left\{ \boldsymbol{W}_{\text{self}}^{(\ell)} \right\}, \left\{ \boldsymbol{W}_{\text{neigh}}^{(\ell)} \right\}$
```

III Graph Sampling-based GCN Model

PASA Lab

B. Complexity of Graph Sampling-Based GCN

- Graph Sampling
  - $f^{(l)}$: the length of feature vector in layer l
  - L: GCN layers
- GraphSAGE



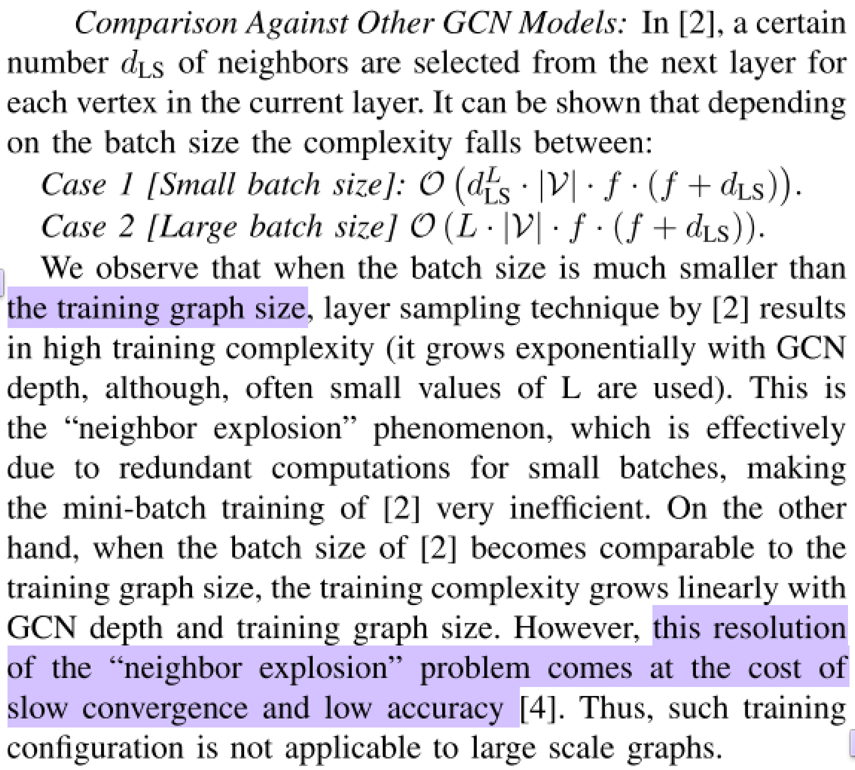
The main operations to propagate a batch (consisting of vertices of the sampled graph) forward by one layer are:

- Feature aggregation: Each feature vector from layer-ℓ propagates via edges in $\mathcal{E}_{\mathrm{GS}}^{(\ell)}$ to be aggregated in the next layer, leading to $\mathcal{O}\left(\left|\mathcal{E}_{\mathrm{GS}}^{(\ell)}\right| \cdot f^{(\ell)}\right)$ operations.
- Weight application: Each vertex multiplies its feature with the weight, leading to $\mathcal{O}\left(\left|\mathcal{V}_{\mathrm{GS}}^{(\ell)}\right| \cdot f^{(\ell-1)} f^{(\ell)}\right)$ operations.

Complexity of L-layer forward propagation in one batch is:

$$\mathcal{O}\left(\sum_{\ell=0}^{L-1}\left(\left|\mathcal{E}_{\mathrm{GS}}^{(\ell)}\right| \cdot f^{(\ell)} + \left|\mathcal{V}_{\mathrm{GS}}^{(\ell+1)}\right| \cdot f^{(\ell)} \cdot f^{(\ell+1)}\right)\right) \tag{1}$$

The GCN constructed on our sampled subgraph has the same set of nodes in all layers $\left|\mathcal{V}_{\mathrm{GS}}^{(\ell)}\right| = |\mathcal{V}_{\mathrm{sub}}|$. If the average degree of the subgraph is d_{GS}, then number of edges between each layer of GCN $\mathcal{E}_{\mathrm{GS}}^{(\ell)} = d_{\mathrm{GS}}|\mathcal{V}_{\mathrm{sub}}|$. This results in the complexity of a batch (subgraph) of our algorithm to $\mathcal{O}\left(L \cdot |\mathcal{V}_{\mathrm{sub}}| \cdot f \cdot (f + d_{\mathrm{GS}})\right)$. If we define an epoch as one full traversal of all training vertices (i.e., processing of $|\mathcal{V}| / |\mathcal{V}_{\mathrm{sub}}|$ batches), complexity of an epoch is $\mathcal{O}\left(L \cdot |\mathcal{V}| \cdot f \cdot (f + d_{\mathrm{GS}})\right)$



Comparison Against Other GCN Models: In [2], a certain number d_{LS} of neighbors are selected from the next layer for each vertex in the current layer. It can be shown that depending on the batch size the complexity falls between:

Case 1 [Small batch size]: $\mathcal{O}\left(d_{\mathrm{LS}}^{L} \cdot |\mathcal{V}| \cdot f \cdot (f + d_{\mathrm{LS}})\right)$.

Case 2 [Large batch size] $\mathcal{O}\left(L \cdot |\mathcal{V}| \cdot f \cdot (f + d_{\mathrm{LS}})\right)$.

We observe that when the batch size is much smaller than the training graph size, layer sampling technique by [2] results in high training complexity (it grows exponentially with GCN depth, although, often small values of L are used). This is the "neighbor explosion" phenomenon, which is effectively due to redundant computations for small batches, making the mini-batch training of [2] very inefficient. On the other hand, when the batch size of [2] becomes comparable to the training graph size, the training complexity grows linearly with GCN depth and training graph size. However, this resolution of the "neighbor explosion" problem comes at the cost of slow convergence and low accuracy [4]. Thus, such training configuration is not applicable to large scale graphs.

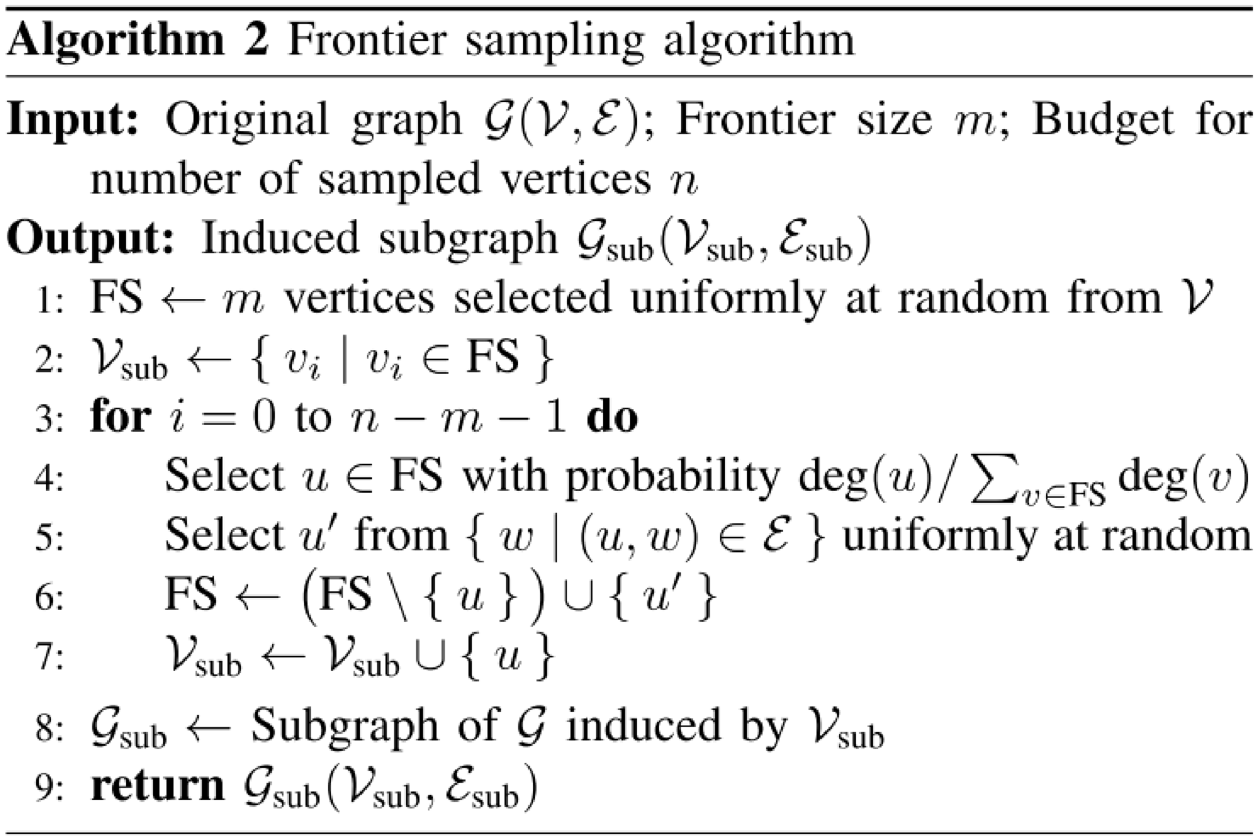
C. Accuracy of Graph Sample-Based GCN

- Targets
  - If the sampling algorithm constructs enough number of representative subgraphs, graph sampling-based GCN model would absorb all the information in G, and generate accurate embedding.
- Key Points
  - A good graph sampling algorithm, should guarantee:
    - Sampled subgraphs preserve the connectivity characteristics in the training graph.
    - Each vertex in the training graph has some non-negligible probability to be sampled
  - Frontier sampling algorithm[5]
    - It’ s output approximate the original graph with respect to multiple connectivity measures.
    - At the beginning of the sampling phase, the sampler picks some initial root vertices uniformly at random from the original graph. These root vertices constitute a significant portion of the sampled graph vertices. Thus, over large enough number of sampling iterations, all input vertex attributes of the training vertices will be covered by the sampler.

III Graph Sampling-based GCN Model

PASA Lab

C. Accuracy of Graph Sample-Based GCN



Algorithm 2 Frontier sampling algorithm

```
Input: Original graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Frontier size $m$; Budget for
    number of sampled vertices $n$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$
1: FS $\leftarrow m$ vertices selected uniformly at random from $\mathcal{V}$
2: $\mathcal{V}_{\text{sub}} \leftarrow \{ v_i \mid v_i \in \text{FS} \}$
3: for $i = 0$ to $n - m - 1$ do
4:     Select $u \in \text{FS}$ with probability $\deg(u) / \sum_{v \in \text{FS}} \deg(v)$
5:     Select $u'$ from $\{ w \mid (u, w) \in \mathcal{E} \}$ uniformly at random
6:     FS $\leftarrow (\text{FS} \setminus \{ u \}) \cup \{ u' \}$
7:     $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ u \}$
8: $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
9: return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$
```

IV Parallel Graph Sampling Algorithm

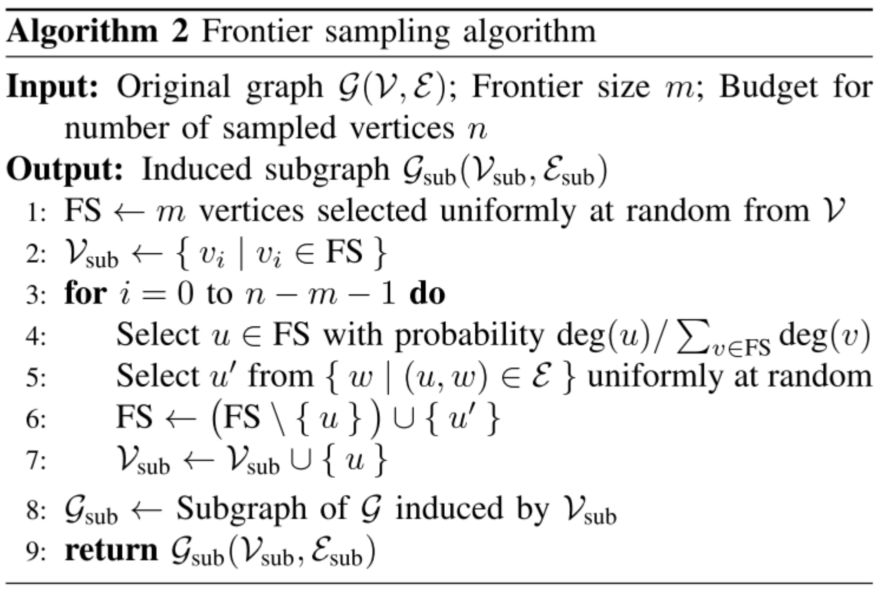
A. Graph Sampling Algorithm

B. Dashboard Based Implementation

C. Intra- and Inter-Subgraph Parallelization

IV Parallel Graph Sampling Algorithm

A. Graph Sampling Algorithm



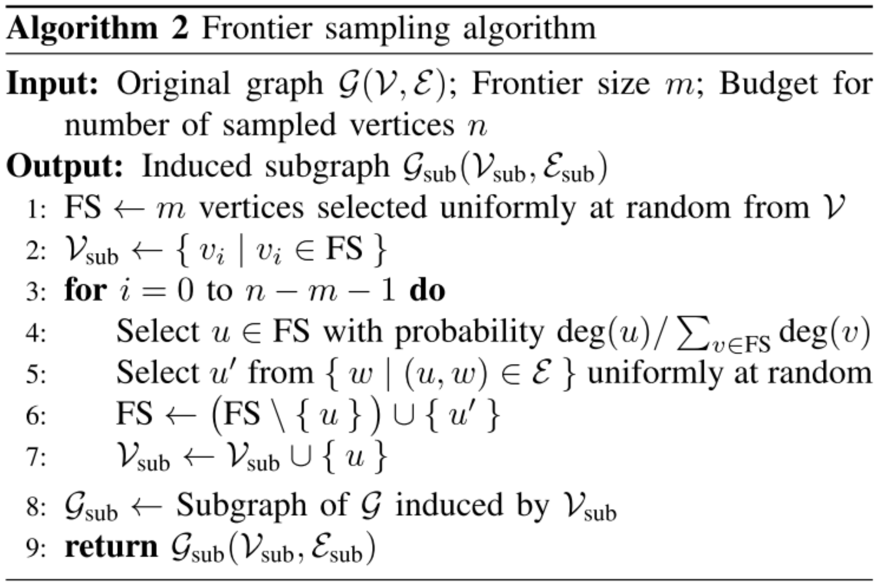
Algorithm 2 Frontier sampling algorithm

```
Input: Original graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Frontier size $m$; Budget for
    number of sampled vertices $n$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$
1: FS $\leftarrow m$ vertices selected uniformly at random from $\mathcal{V}$
2: $\mathcal{V}_{\text{sub}} \leftarrow \{ v_i \mid v_i \in \text{FS} \}$
3: for $i = 0$ to $n - m - 1$ do
4:     Select $u \in \text{FS}$ with probability $\deg(u) / \sum_{v \in \text{FS}} \deg(v)$
5:     Select $u'$ from $\{ w \mid (u, w) \in \mathcal{E} \}$ uniformly at random
6:     FS $\leftarrow (\text{FS} \setminus \{ u \}) \cup \{ u' \}$
7:     $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ u \}$
8: $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
9: return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$
```

- In our experiments sequential implementation, we notice that half of the time is spent in sampling phase. Therefore, there is a need to parallelize the sampling phase.
- Challenges
  - While sampling from a discrete distribution is a well-researched problem, we are focused on fast parallel sampling of a dynamic degree distribution. (changes with addition/deletion of new vertex to the frontier). O(mn)
  - The sampler is inherently sequential. The vertices in the frontier set should be popped out one at a time to guarantee the quality of the sampled graph.

IV Parallel Graph Sampling Algorithm

B. Dashboard Based Implementation



Algorithm 2 Frontier sampling algorithm

```
Input: Original graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; Frontier size $m$; Budget for
    number of sampled vertices $n$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
1: FS $\leftarrow m$ vertices selected uniformly at random from $\mathcal{V}$
2: $\mathcal{V}_{\text{sub}} \leftarrow \{ v_i \mid v_i \in \text{FS} \}$
3: for $i = 0$ to $n - m - 1$ do
4:     Select $u \in \text{FS}$ with probability $\deg(u)/\sum_{v\in\text{FS}} \deg(v)$
5:     Select $u'$ from $\{ w \mid (u,w) \in \mathcal{E} \}$ uniformly at random
6:     FS $\leftarrow (\text{FS} \setminus \{ u \}) \cup \{ u' \}$
7:     $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ u \}$
8: $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
9: return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
```

- Dashboard DB: $R^{3*(\eta*m*d)}$. For every v_i:
  - i, the position of G
  - Offset value, to check if it need to be popped out
  - k, the $k-th$ vertex added into DB.
- Index array IA: $R^{2*(\eta*m*d+1)}$, the last entry contains the current number of used DB entries. for every the $j-th$ entry in IA, the $j-th$ vertex added into DB:
  - Offset， the starting index of the DB entries corresponding to v.
  - Flag, true when v is a current frontier vertex, and False when v is a historical frontier vertex

IV Parallel Graph Sampling Algorithm

C. Intra- and Inter-Subgraph Parallelization

Algorithm 3 Parallel Dashboard based frontier sampling

```
Input: Original graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; Frontier size $m$; Budget $n$;
    Enlargement factor $\eta$; Number of processors $p$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
 1: $d \leftarrow |\mathcal{E}|/|\mathcal{V}|$
 2: DB $\leftarrow$ Array of $\mathbb{R}^{3\times(\eta\cdot m\cdot d)}$ with value INV        ▷ INValid
 3: IA $\leftarrow$ Array of $\mathbb{R}^{2\times(\eta\cdot m\cdot d+1)}$ with value INV
 4: FS $\leftarrow$ $m$ vertices selected uniformly at random from $\mathcal{V}$
 5: $\mathcal{V}_{\text{sub}} \leftarrow \{ v \mid v \in \text{FS} \}$
 6: FS $\leftarrow$ Indexable list of vertices converted from set FS
 7: IA$[0,0] \leftarrow 0$;     IA$[1,0] \leftarrow$True;
 8: for $i = 1$ to $m$ do                                  ▷ Initialize IA from FS
 9:     IA$[0,i] \leftarrow$ IA$[0,i-1] + \deg(\text{FS}[i-1])$
10:     IA$[1,i] \leftarrow (i \neq m)$?True : False
11: for $i = 0$ to $m-1$ pardo                             ▷ Initialize DB from FS
12:     for $k =$ IA$[i]$ to IA$[i+1]-1$ do
13:         DB$[0,k] \leftarrow$ FS$[i]$
14:         DB$[1,k] \leftarrow (k \neq \text{IA}[i])?(k - \text{IA}[i]) : -\deg(\text{FS}[i])$
15:         DB$[2,k] \leftarrow i$
16: $s \leftarrow m$
17: for $i = m$ to $n-1$ do                                ▷ Sampling main loop
18:     $v_{\text{pop}} \leftarrow$ pardo_POP_FRONTIER(DB,$p$)
19:     $v_{\text{new}} \leftarrow$ Vertex sampled from $v_{\text{pop}}$'s neighbors
20:     if $\deg(v_{\text{new}}) > \eta \cdot m \cdot d - \text{IA}[0,s] + 1$ then
21:         DB $\leftarrow$ pardo_CLEANUP(DB, IA, $p$)
22:         $s \leftarrow m - 1$
23:     pardo_ADD_TO_FRONTIER $(v_{\text{new}}, s, \text{DB}, \text{IA}, p)$
24:     $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ v_{\text{new}} \}$
25:     $s \leftarrow s + 1$
26: $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
27: return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
```

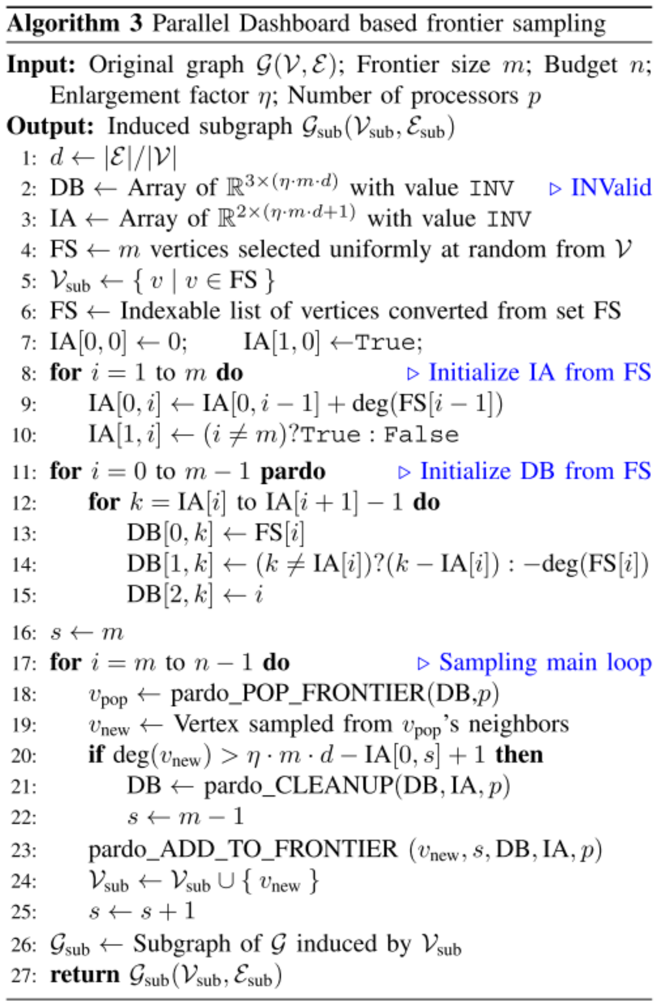
Algorithm 2 Frontier sampling algorithm

```
Input: Original graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; Frontier size $m$; Budget for
    number of sampled vertices $n$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
 1: FS $\leftarrow$ $m$ vertices selected uniformly at random from $\mathcal{V}$
 2: $\mathcal{V}_{\text{sub}} \leftarrow \{ v_i \mid v_i \in \text{FS} \}$
 3: for $i = 0$ to $n - m - 1$ do
 4:     Select $u \in$ FS with probability $\deg(u)/\sum_{v\in\text{FS}} \deg(v)$
 5:     Select $u'$ from $\{ w \mid (u,w) \in \mathcal{E} \}$ uniformly at random
 6:     FS $\leftarrow (\text{FS} \setminus \{ u \}) \cup \{ u' \}$
 7:     $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ u \}$
 8: $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
 9: return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
```

- Dashboard DB: $R^{3*(\eta*m*d)}$. For every v_i:
  - i, the position of G
  - Offset value, to check if it need to be popped out
  - k, the $k-th$ vertex added into DB.
- Index array IA: $R^{2*(\eta*m*d+1)}$, the last entry contains the current number of used DB entries. for every the $j-th$ entry in IA, the $j-th$ vertex added into DB:
  - Offset, the starting index of the DB entries corresponding to v.
  - Flag, true when v is a current frontier vertex, and False when v is a historical frontier vertex

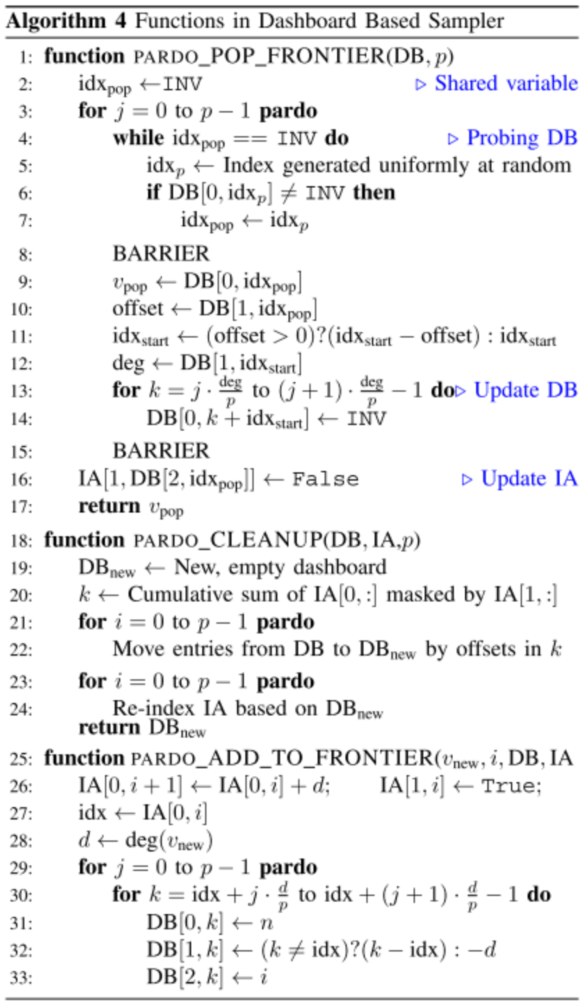
IV Parallel Graph Sampling Algorithm

C. Intra- and Inter-Subgraph Parallelization



Algorithm 3 Parallel Dashboard based frontier sampling

```
Input: Original graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; Frontier size $m$; Budget $n$;
    Enlargement factor $\eta$; Number of processors $p$
Output: Induced subgraph $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
$d \leftarrow |\mathcal{E}|/|\mathcal{V}|$
DB $\leftarrow$ Array of $\mathbb{R}^{3\times(\eta\cdot m\cdot d)}$ with value INV        ▷ INValid
IA $\leftarrow$ Array of $\mathbb{R}^{2\times(\eta\cdot m\cdot d+1)}$ with value INV
FS $\leftarrow$ $m$ vertices selected uniformly at random from $\mathcal{V}$
$\mathcal{V}_{\text{sub}} \leftarrow \{ v \mid v \in \text{FS} \}$
FS $\leftarrow$ Indexable list of vertices converted from set FS
IA$[0,0] \leftarrow 0$;      IA$[1,0] \leftarrow$ True;
for $i = 1$ to $m$ do                                ▷ Initialize IA from FS
    IA$[0,i] \leftarrow$ IA$[0,i-1] + \deg(\text{FS}[i-1])$
    IA$[1,i] \leftarrow (i \neq m)$?True : False
for $i = 0$ to $m-1$ pardo                           ▷ Initialize DB from FS
    for $k =$ IA$[i]$ to IA$[i+1]-1$ do
        DB$[0,k] \leftarrow$ FS$[i]$
        DB$[1,k] \leftarrow (k \neq \text{IA}[i])?(k - \text{IA}[i]) : -\deg(\text{FS}[i])$
        DB$[2,k] \leftarrow i$
$s \leftarrow m$
for $i = m$ to $n-1$ do                              ▷ Sampling main loop
    $v_{\text{pop}} \leftarrow$ pardo_POP_FRONTIER(DB,$p$)
    $v_{\text{new}} \leftarrow$ Vertex sampled from $v_{\text{pop}}$'s neighbors
    if $\deg(v_{\text{new}}) > \eta\cdot m\cdot d - \text{IA}[0,s] + 1$ then
        DB $\leftarrow$ pardo_CLEANUP(DB, IA, $p$)
        $s \leftarrow m-1$
    pardo_ADD_TO_FRONTIER ($v_{\text{new}}, s$, DB, IA, $p$)
    $\mathcal{V}_{\text{sub}} \leftarrow \mathcal{V}_{\text{sub}} \cup \{ v_{\text{new}} \}$
    $s \leftarrow s+1$
$\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph of $\mathcal{G}$ induced by $\mathcal{V}_{\text{sub}}$
return $\mathcal{G}_{\text{sub}}(\mathcal{V}_{\text{sub}},\mathcal{E}_{\text{sub}})$
```

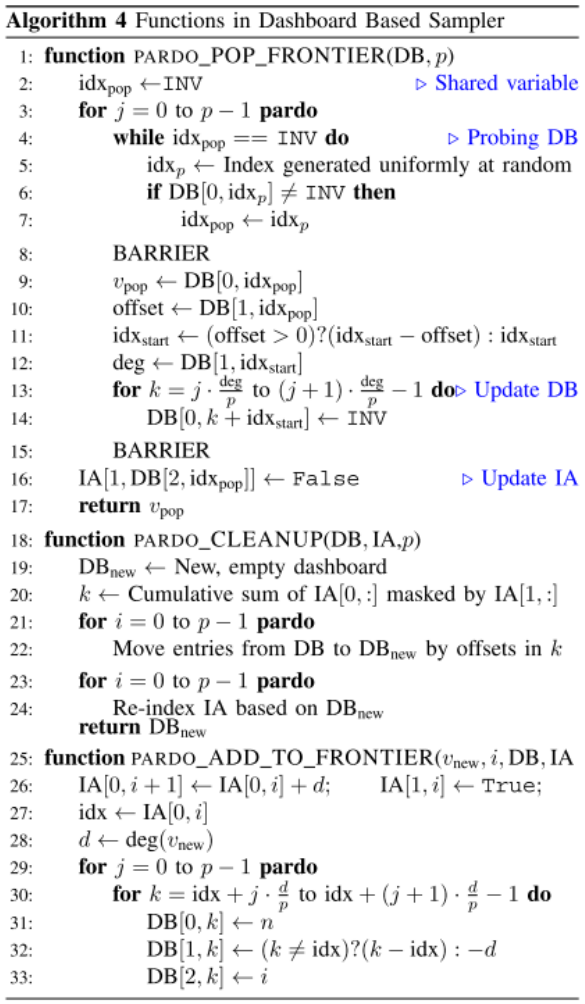


Algorithm 4 Functions in Dashboard Based Sampler

```
function PARDO_POP_FRONTIER(DB, $p$)
    idx$_{\text{pop}}$ $\leftarrow$INV                              ▷ Shared variable
    for $j = 0$ to $p-1$ pardo
        while idx$_{\text{pop}}$ == INV do                          ▷ Probing DB
            idx$_p$ $\leftarrow$ Index generated uniformly at random
            if DB$[0,\text{idx}_p] \neq$ INV then
                idx$_{\text{pop}} \leftarrow$ idx$_p$
        BARRIER
        $v_{\text{pop}} \leftarrow$ DB$[0,\text{idx}_{\text{pop}}]$
        offset $\leftarrow$ DB$[1,\text{idx}_{\text{pop}}]$
        idx$_{\text{start}} \leftarrow$ (offset $> 0$)?(idx$_{\text{start}}$ $-$ offset) : idx$_{\text{start}}$
        deg $\leftarrow$ DB$[1,\text{idx}_{\text{start}}]$
        for $k = j\cdot\frac{\deg}{p}$ to $(j+1)\cdot\frac{\deg}{p} - 1$ do▷ Update DB
            DB$[0, k + \text{idx}_{\text{start}}] \leftarrow$ INV
        BARRIER
    IA$[1,\text{DB}[2,\text{idx}_{\text{pop}}]] \leftarrow$ False                ▷ Update IA
    return $v_{\text{pop}}$
function PARDO_CLEANUP(DB, IA,$p$)
    DB$_{\text{new}}$ $\leftarrow$ New, empty dashboard
    $k \leftarrow$ Cumulative sum of IA$[0,:]$ masked by IA$[1,:]$
    for $i = 0$ to $p-1$ pardo
        Move entries from DB to DB$_{\text{new}}$ by offsets in $k$
    for $i = 0$ to $p-1$ pardo
        Re-index IA based on DB$_{\text{new}}$
    return DB$_{\text{new}}$
function PARDO_ADD_TO_FRONTIER($v_{\text{new}}, i$, DB, IA
    IA$[0,i+1] \leftarrow$ IA$[0,i] + d$;      IA$[1,i] \leftarrow$ True;
    idx $\leftarrow$ IA$[0,i]$
    $d \leftarrow \deg(v_{\text{new}})$
    for $j = 0$ to $p-1$ pardo
        for $k = \text{idx} + j\cdot\frac{d}{p}$ to $\text{idx} + (j+1)\cdot\frac{d}{p} - 1$ do
            DB$[0,k] \leftarrow n$
            DB$[1,k] \leftarrow (k \neq \text{idx})?(k - \text{idx}) : -d$
            DB$[2,k] \leftarrow i$
```

IV Parallel Graph Sampling Algorithm

C. Intra- and Inter-Subgraph Parallelization

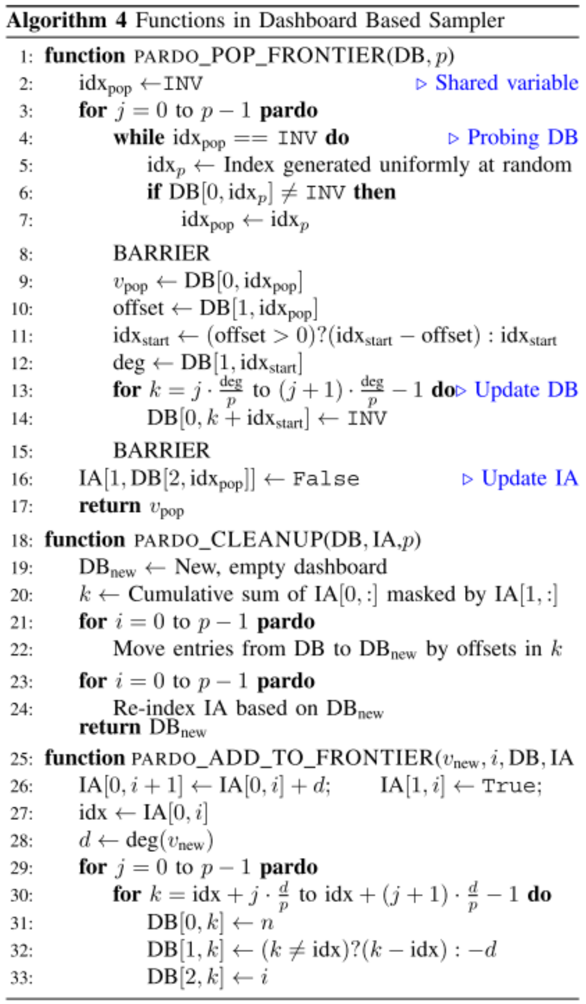


Algorithm 4 Functions in Dashboard Based Sampler

```
function PARDO_POP_FRONTIER(DB, $p$)
    $\text{idx}_{\text{pop}} \leftarrow$ INV                                ▷ Shared variable
    for $j = 0$ to $p-1$ pardo
        while $\text{idx}_{\text{pop}} ==$ INV do                        ▷ Probing DB
            $\text{idx}_p \leftarrow$ Index generated uniformly at random
            if $\text{DB}[0, \text{idx}_p] \neq$ INV then
                $\text{idx}_{\text{pop}} \leftarrow \text{idx}_p$
        BARRIER
        $v_{\text{pop}} \leftarrow \text{DB}[0, \text{idx}_{\text{pop}}]$
        offset $\leftarrow \text{DB}[1, \text{idx}_{\text{pop}}]$
        $\text{idx}_{\text{start}} \leftarrow (\text{offset} > 0)?(\text{idx}_{\text{start}} - \text{offset}) : \text{idx}_{\text{start}}$
        deg $\leftarrow \text{DB}[1, \text{idx}_{\text{start}}]$
        for $k = j \cdot \frac{\text{deg}}{p}$ to $(j+1) \cdot \frac{\text{deg}}{p} - 1$ do▷ Update DB
            $\text{DB}[0, k + \text{idx}_{\text{start}}] \leftarrow$ INV
        BARRIER
    $\text{IA}[1, \text{DB}[2, \text{idx}_{\text{pop}}]] \leftarrow$ False              ▷ Update IA
    return $v_{\text{pop}}$
function PARDO_CLEANUP(DB, IA, $p$)
    $\text{DB}_{\text{new}} \leftarrow$ New, empty dashboard
    $k \leftarrow$ Cumulative sum of $\text{IA}[0,:]$ masked by $\text{IA}[1,:]$
    for $i = 0$ to $p-1$ pardo
        Move entries from DB to $\text{DB}_{\text{new}}$ by offsets in $k$
    for $i = 0$ to $p-1$ pardo
        Re-index IA based on $\text{DB}_{\text{new}}$
    return $\text{DB}_{\text{new}}$
function PARDO_ADD_TO_FRONTIER($v_{\text{new}}$, $i$, DB, IA
    $\text{IA}[0, i+1] \leftarrow \text{IA}[0, i] + d$;      $\text{IA}[1, i] \leftarrow$ True;
    idx $\leftarrow \text{IA}[0, i]$
    $d \leftarrow \text{deg}(v_{\text{new}})$
    for $j = 0$ to $p-1$ pardo
        for $k = \text{idx} + j \cdot \frac{d}{p}$ to $\text{idx} + (j+1) \cdot \frac{d}{p} - 1$ do
            $\text{DB}[0, k] \leftarrow n$
            $\text{DB}[1, k] \leftarrow (k \neq \text{idx})?(k - \text{idx}) : -d$
            $\text{DB}[2, k] \leftarrow i$
```

- The parallel Dashboard based frontier sampler:
  - Leads to low serial complexity by incremental update on probability distribution
  - Scales well up to p = O(d).

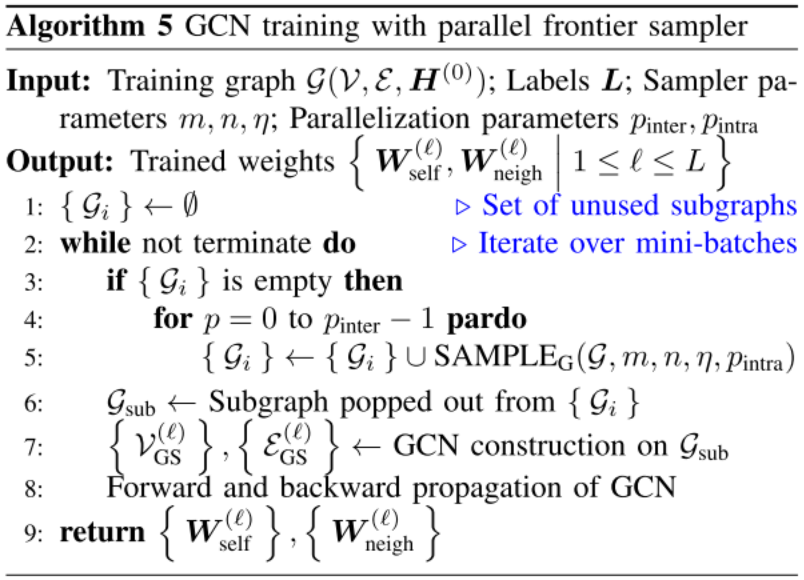
C. Intra- and Inter-Subgraph Parallelization



Algorithm 4 Functions in Dashboard Based Sampler

```
1: function PARDO_POP_FRONTIER(DB, $p$)
2:     $\text{idx}_{\text{pop}} \leftarrow$ INV                              ▷ Shared variable
3:     for $j = 0$ to $p - 1$ pardo
4:         while $\text{idx}_{\text{pop}} ==$ INV do                     ▷ Probing DB
5:             $\text{idx}_p \leftarrow$ Index generated uniformly at random
6:             if $\text{DB}[0, \text{idx}_p] \neq$ INV then
7:                 $\text{idx}_{\text{pop}} \leftarrow \text{idx}_p$
8:         BARRIER
9:         $v_{\text{pop}} \leftarrow \text{DB}[0, \text{idx}_{\text{pop}}]$
10:        $\text{offset} \leftarrow \text{DB}[1, \text{idx}_{\text{pop}}]$
11:        $\text{idx}_{\text{start}} \leftarrow (\text{offset} > 0)?(\text{idx}_{\text{start}} - \text{offset}) : \text{idx}_{\text{start}}$
12:        $\text{deg} \leftarrow \text{DB}[1, \text{idx}_{\text{start}}]$
13:        for $k = j \cdot \frac{\text{deg}}{p}$ to $(j+1) \cdot \frac{\text{deg}}{p} - 1$ do ▷ Update DB
14:            $\text{DB}[0, k + \text{idx}_{\text{start}}] \leftarrow$ INV
15:        BARRIER
16:    $\text{IA}[1, \text{DB}[2, \text{idx}_{\text{pop}}]] \leftarrow$ False                ▷ Update IA
17:    return $v_{\text{pop}}$
18: function PARDO_CLEANUP(DB, IA, $p$)
19:    $\text{DB}_{\text{new}} \leftarrow$ New, empty dashboard
20:    $k \leftarrow$ Cumulative sum of $\text{IA}[0, :]$ masked by $\text{IA}[1, :]$
21:    for $i = 0$ to $p - 1$ pardo
22:        Move entries from DB to $\text{DB}_{\text{new}}$ by offsets in $k$
23:    for $i = 0$ to $p - 1$ pardo
24:        Re-index IA based on $\text{DB}_{\text{new}}$
    return $\text{DB}_{\text{new}}$
25: function PARDO_ADD_TO_FRONTIER($v_{\text{new}}, i$, DB, IA
26:    $\text{IA}[0, i+1] \leftarrow \text{IA}[0, i] + d$;      $\text{IA}[1, i] \leftarrow$ True;
27:    $\text{idx} \leftarrow \text{IA}[0, i]$
28:    $d \leftarrow \text{deg}(v_{\text{new}})$
29:    for $j = 0$ to $p - 1$ pardo
30:        for $k = \text{idx} + j \cdot \frac{d}{p}$ to $\text{idx} + (j+1) \cdot \frac{d}{p} - 1$ do
31:            $\text{DB}[0, k] \leftarrow n$
32:            $\text{DB}[1, k] \leftarrow (k \neq \text{idx})?(k - \text{idx}) : -d$
33:            $\text{DB}[2, k] \leftarrow i$
```

- Inter- and Intra-
    - Each of the p-inter sampler instances runs on p-intra processing units.



Algorithm 5 GCN training with parallel frontier sampler

Input: Training graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{H}^{(0)})$; Labels $\boldsymbol{L}$; Sampler parameters m, n, η; Parallelization parameters $p_{\text{inter}}, p_{\text{intra}}$

Output: Trained weights $\left\{ \boldsymbol{W}_{\text{self}}^{(\ell)}, \boldsymbol{W}_{\text{neigh}}^{(\ell)} \mid 1 \leq \ell \leq L \right\}$

```
1: $\{ \mathcal{G}_i \} \leftarrow \emptyset$                                  ▷ Set of unused subgraphs
2: while not terminate do                              ▷ Iterate over mini-batches
3:     if $\{ \mathcal{G}_i \}$ is empty then
4:         for $p = 0$ to $p_{\text{inter}} - 1$ pardo
5:             $\{ \mathcal{G}_i \} \leftarrow \{ \mathcal{G}_i \} \cup \text{SAMPLE}_{\text{G}}(\mathcal{G}, m, n, \eta, p_{\text{intra}})$
6:     $\mathcal{G}_{\text{sub}} \leftarrow$ Subgraph popped out from $\{ \mathcal{G}_i \}$
7:     $\left\{ \mathcal{V}_{\text{GS}}^{(\ell)} \right\}, \left\{ \mathcal{E}_{\text{GS}}^{(\ell)} \right\} \leftarrow$ GCN construction on $\mathcal{G}_{\text{sub}}$
8:     Forward and backward propagation of GCN
9: return $\left\{ \boldsymbol{W}_{\text{self}}^{(\ell)} \right\}, \left\{ \boldsymbol{W}_{\text{neigh}}^{(\ell)} \right\}$
```

V Parallel Training Algorithm

PASA Lab

Content

- A. Two kernels:
  - Feature propagation in subgraph
  - Dense matrix multiplication.
- B. Parallel Feature Propagation within Subgraph
  - Related Work:
    - vertex-centric,
    - edge-centric
    - partition-centric paradigms perform vertex partitioning on graphs so that processors can work independently in parallel.
  - Our Work
    - Two dimensional partitioning along the graph. Q_v
    - Feature dimensions. Q_f

$$\begin{aligned} \underset{Q_v, Q_f}{\text{minimize}} \quad & g_{\text{comm}}(Q_v, Q_f) = 2Q_f \cdot nd + 8Q_v \cdot nf\gamma_v \\ \text{subject to} \quad & Q_v Q_f \geq p; \quad \frac{8nf\gamma_v}{Q_f} \leq S_{\text{cache}}; \quad Q_v, Q_f \in \mathbb{Z}^+ \end{aligned}$$

Algorithm 6 Feature propagation within sampled graph

Input: Sampled graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Vertex features $\{ \boldsymbol{h}_v \}$; Cache size S_{cache}; Number of processors p

Output: Updated features $\{ \boldsymbol{h}_v \}$

```
1: $n \leftarrow |\mathcal{V}|$;    $f \leftarrow |\boldsymbol{h}_v|$;
2: $Q_f \leftarrow \max \left\{ p, \frac{8nf}{S_{\text{cache}}} \right\}$
3: Equal partition each $\boldsymbol{h}_v$ into $\left\{ \boldsymbol{h}_v^{(i)} \,\middle|\, 0 \leq i \leq Q_f - 1 \right\}$
4: for $r = 0$ to $Q_f/p - 1$ do
5:     for $i = 0$ to $p - 1$ pardo
6:         Propagation of $\left\{ \boldsymbol{h}_v^{(i+r\cdot p)} \,\middle|\, v \in \mathcal{V} \right\}$ into $\mathcal{V}$
7: return $\{ \boldsymbol{h}_v \}$
```

VI Experiment

A. Experiment Setup

- 4 large scale graph datasets:
  - PPI: a protein-protein interaction graph.
  - Reddit: a post-post graph
  - Yelp: a social network graph. Vertex attributes are user comments converted from text using Word2Vec
  - Amazon: An item-item graph
- Two implementations:
  - VI-B: Tensorflow
  - VI-C: c++: since Python and Tensorflow are not flexible enough for parallel computing experiments.(eg., AVX and thread binding are not explicit in Python
- Environment
  - Dual 20-Core Intel Xeon E5-2698 v4@2.2GHz machine with 512 GB of DDR4 RAM.
  - Python: 3.6.5, Tensorflow 1.10.0
  - C++: -O3 flag

VI Experiment

B. Evaluation on Accuracy and Efficiency

- 2-layer GCN model
- Threshold: a0 – 0.0025
- 1.9x, 7.8x, 4.7x and 2.1x for PPI, Reddit, Yelp and Amazon

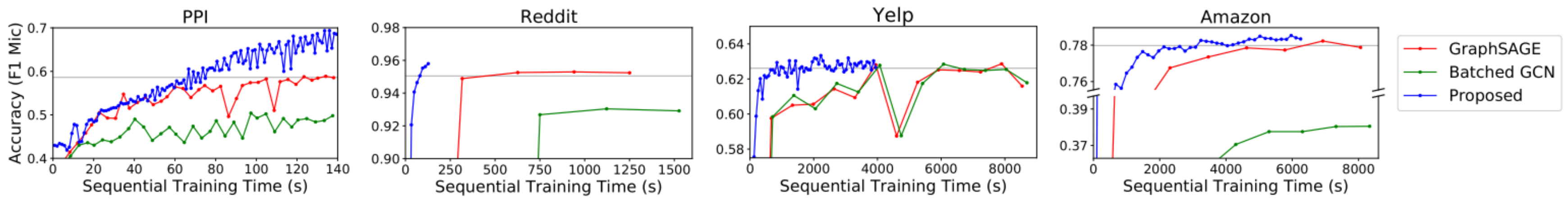


Fig. 2: Time-Accuracy plot for sequential execution

VI Experiment

C. Evaluation on Scalability

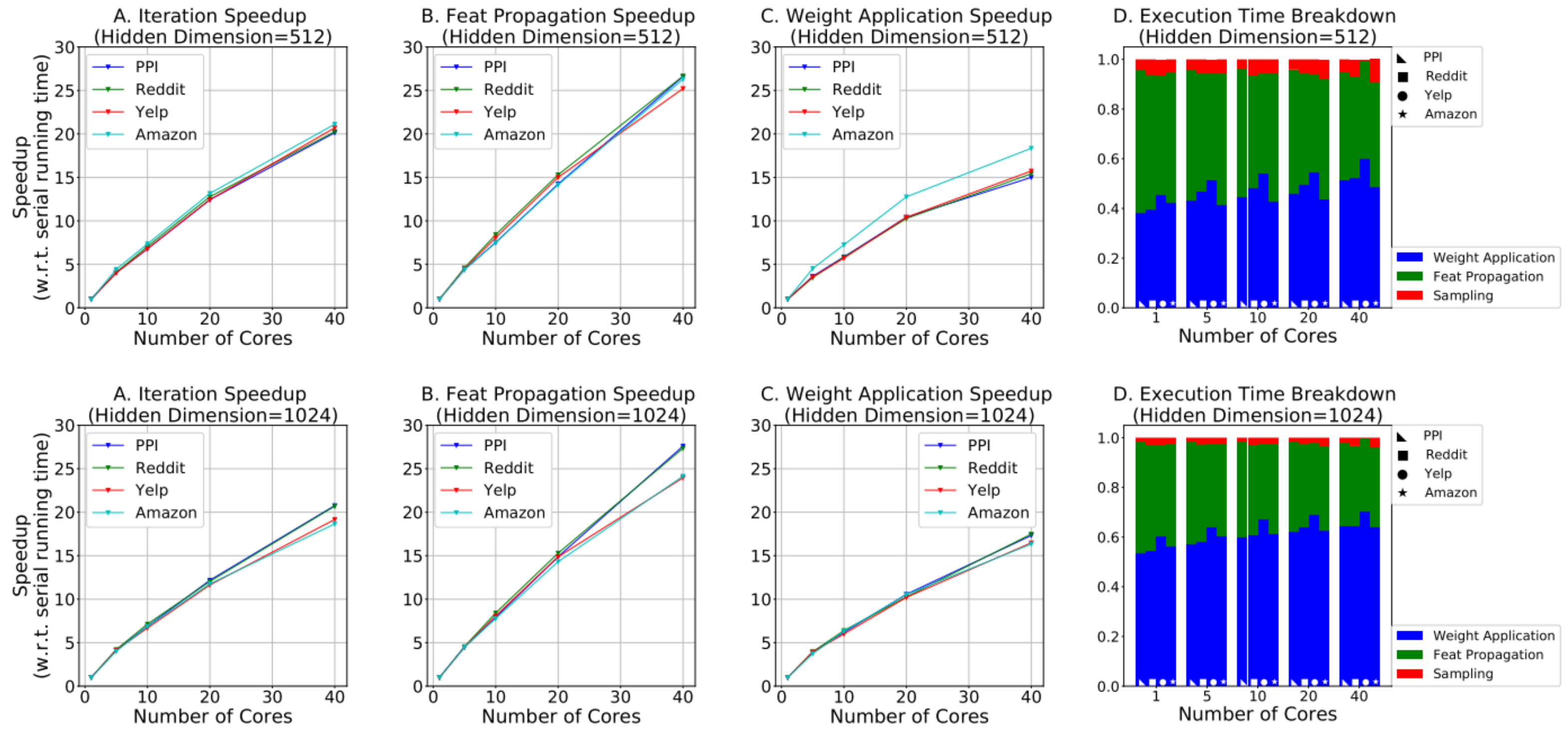


Fig. 3: Scaling evaluation with hidden feature dimensions: 512 (Upper), 1024 (Lower)

VI Experiment

PASA Lab

C. Evaluation on Scalability

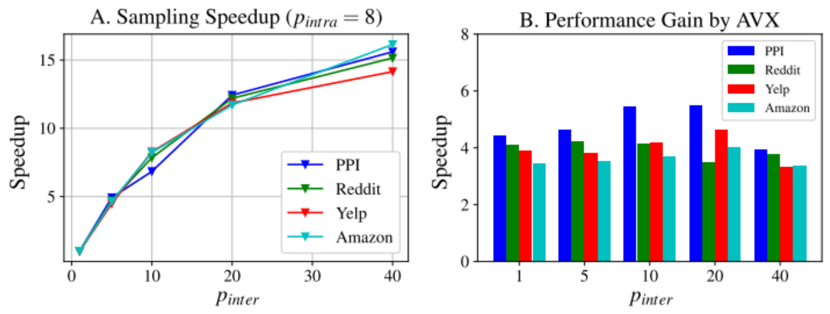


Fig. 4: Sampling speedup (inter- & intra-subgraph parallelism)

TABLE II: Speedup Comparison with Parallelized [2] (Reddit)

	1-core	5-core	10-core	20-core	40-core
1-layer	2.03×	4.77×	9.34×	17.25×	23.93×
2-layer	7.74×	12.95×	18.50×	28.43×	37.44×
3-layer	335.36×	568.93×	828.25×	1164.45×	1306.21×

PASA Lab

谢 谢