Reptile for Multitask Graph Execution

Jingdong Gao, Ted Zadouri, Armaan Abraham, Yihang Guo, Alex Taylor

Outline

- Introduction and Background
  - Neural Execution of Graph Algorithms
  - Sequential Reptile
- Reptile for Multitask Graph Execution
  - Problem statement
  - Methods
  - Experimental Setting & Experiments
- Conclusion & Future Works

Introduction

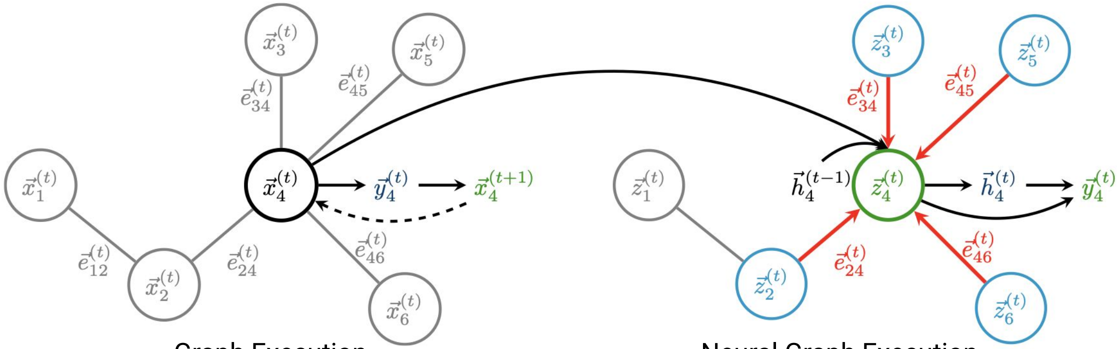
- Neural Algorithmic Reasoning incorporates the theoretical guarantees of classical computer science algorithms with the flexibility of deep learning
- Neural Reasoning typically relies only on the final state as the training signal
- Recent approaches have begun to incorporate intermediate steps to improve the accuracy of the solution

Neural Execution of Graph Algorithms

- Many classical graph algorithms (e.g. Bellman-ford, Breadth-first search, Dijkstra) share common subroutines
    - Ex. Enumerating sets of edges adjacent to a given node
- Intuition: learning multiple graph algorithms simultaneously is the best way to leverage subroutines
- The supervision signals in the learning process are the intermediate outputs of the classical algorithms we're trying to learn

Neural Execution

- The neural executor utilizes an encoder-processor-decoder architecture



Graph Execution

Neural Graph Execution

Encoder $\vec{z}_i^{(t)} = f_A(\vec{x}_i^{(t)}, \vec{h}_i^{(t-1)})$

Processor $\mathbf{H}^{(t)} = P(\mathbf{Z}^{(t)}, \mathbf{E}^{(t)})$

Decoder $\vec{y}_i^{(t)} = g_A(\vec{z}_i^{(t)}, \vec{h}_i^{(t)})$

Termination $\tau^{(t)} = \sigma(T_A(\mathbf{H}^{(t)}, \overline{\mathbf{H}^{(t)}}))$

- The loss for each task is aggregated during each training iteration
  - The parameters for the shared processor network as well as the task-specific encoder/decoder networks are all updated at once using the aggregate loss
  - Dissimilarity between task gradients makes learning more difficult

Sequential Reptile

- A learning framework that seeks to maximize the knowledge transfer when learning multiple tasks simultaneously
- Main observation: it is crucial to align gradients between tasks in order to maximize knowledge transfer while minimizing negative transfer (forgetting)
- Method: assume the model is trained on ***T*** tasks
- For each outer iteration: perform ***K*** updates (inner iterations) on a copy of the current model parameters to compute meta gradient (MG)
- For each inner iteration: sample a task $\boldsymbol{t_k}$ from a categorical distribution, then sample a batch $\boldsymbol{B_{tk}^{(k)}}$ from task $\boldsymbol{t_k}$

$$\theta^{(0)} = \phi, \qquad \theta^{(k)} = \theta^{(k-1)} - \alpha \frac{\partial \mathcal{L}(\theta^{(k-1)}; \mathcal{B}_{t_k}^{(k)})}{\partial \theta^{(k-1)}}$$

- After K inner iterations (1 outer iteration):

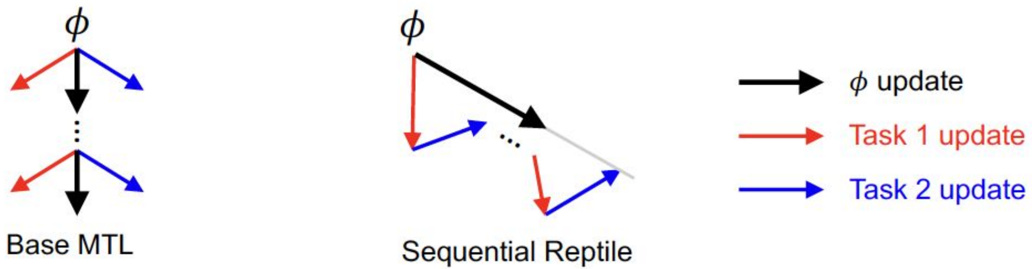
$$\phi \leftarrow \phi - \eta \cdot \mathrm{MG}(\phi), \quad \text{where} \quad \mathrm{MG}(\phi) = \phi - \theta^{(K)}$$

- Advantage:

$$\mathbb{E}\left[\mathrm{MG}(\phi)\right] \approx \frac{\partial}{\partial \phi}\mathbb{E}\left[\sum_{k=1}^{K} \mathcal{L}(\phi; \mathcal{B}_{t_k}^{(k)}) - \frac{\alpha}{2}\sum_{k=1}^{K}\sum_{j=1}^{k-1}\left\langle \frac{\partial \mathcal{L}(\phi; \mathcal{B}_{t_k}^{(k)})}{\partial \phi}, \frac{\partial \mathcal{L}(\phi; \mathcal{B}_{t_j}^{(j)})}{\partial \phi} \right\rangle\right]$$

Problem Statement

- We want to align the gradients from each algorithm learning task to leverage inter-algorithm features in a multi-task learning setting to improve performance
    - Ex. common subroutines: shortest path computation via Bellman-Ford and breadth-first search both enumerate sets of edges adjacent to a particular node



Training Procedure

- Incorporate sequential reptile with neural execution baseline for multi-task learning
- For each iteration: perform K updates on a copy of the current model parameters
  - For each update: sample a task t_k (BFS or Bellman-Ford) from a categorical distribution / specified order, then sample a batch $B_{tk}^{(k)}$ from task t_k and taking gradient steps with them (learning rate = 5e-4)
  - After K updates
    - we meta-update Φ with learning rate = 1
- Also, the percentage of graph numbers w.r.t. different algorithms varies (imbalanced data settings)

Experiment Settings

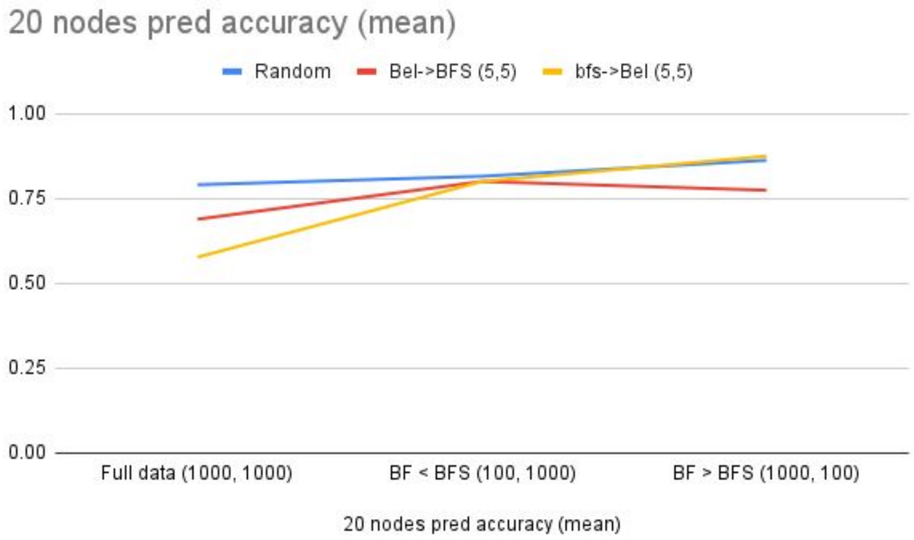
- Algorithms: We are currently focusing on two parallel algorithms, **BELLMAN-FORD** and **BREADTH-FIRST SEARCH**
- Our datasets:
  - Graph Types: Most experiments are done on **Erdós Rényi** graphs
  - Train Set: number of graph for each algorithm = 1000 (or 100 imbalance case); number of nodes = 20
  - Validation Set: number of graph for each algorithm = 100; number of nodes = 20
  - Test Set: number of graph for each algorithm = 200; number of nodes = 20, 50
- Performance metrics:
  - BFS: reachability mean step accuracy; reachability last step accuracy; termination accuracy
  - Bellman-Ford: mean squared error, last step mean squared error; predecessors mean step accuracy: predecessors last step accuracy; termination accuracy

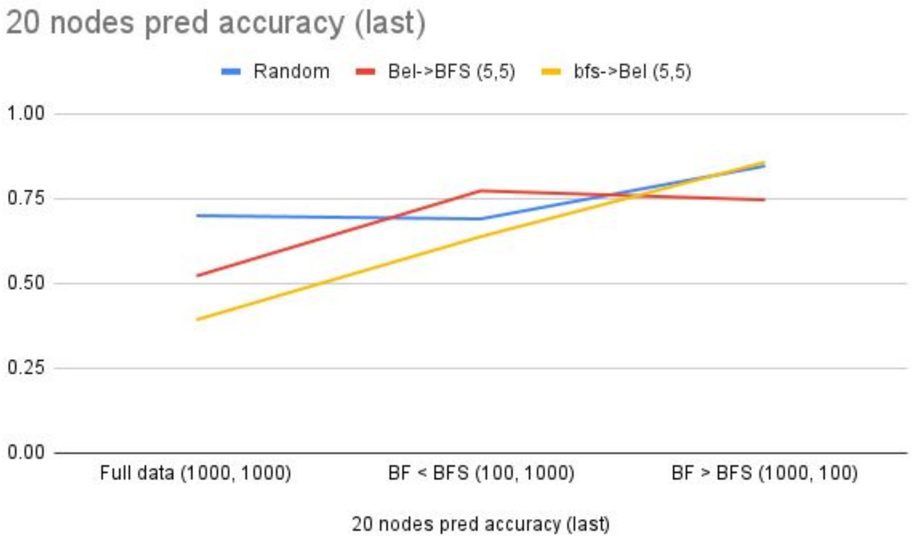
Seq Reptile vs Vanilla Multitasking

20 Nodes	Reachability Acc (mean/last)	Predecessor Acc (mean/last)	MSE (mean/last)	Termination Acc (BF/BFS)
Sequential Reptile	0.9882/0.9882	0.8628/0.8460	0.08/0.08	0.8240/0.772
Vanilla Multitasking	0.9921/0.9799	0.7365/0.6999	0.147/0.151	0.7544/0.7544

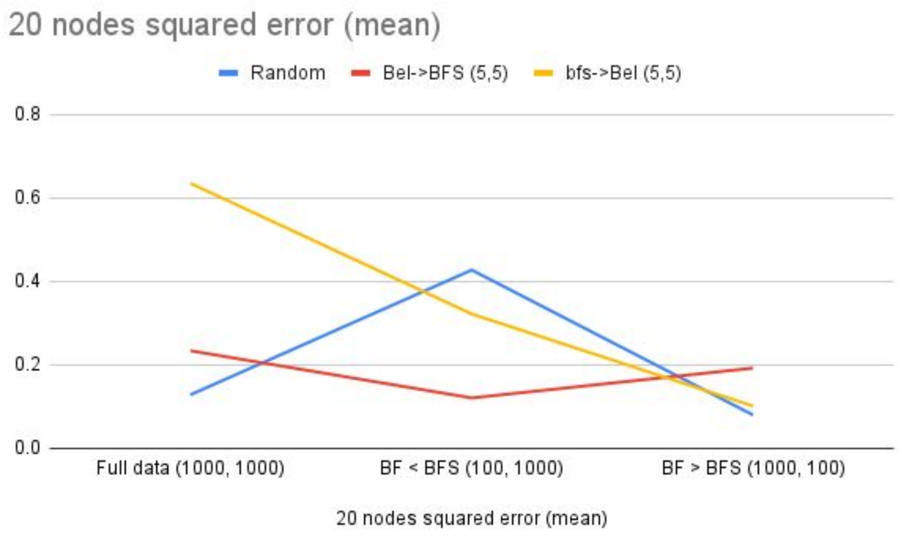
50 Nodes	Reachability Acc (mean/last)	Predecessor Acc (mean/last)	MSE (mean/last)	Termination Acc (BF/BFS)
Sequential Reptile	0.9970/ 0.9970	0.8356/0.8076	0.2593/0.3359	0.8828/0.78
Vanilla Multitasking	0.9979/0.9595	0.6954/0.6889	0.5378/1.2226	0.6751/0.6751

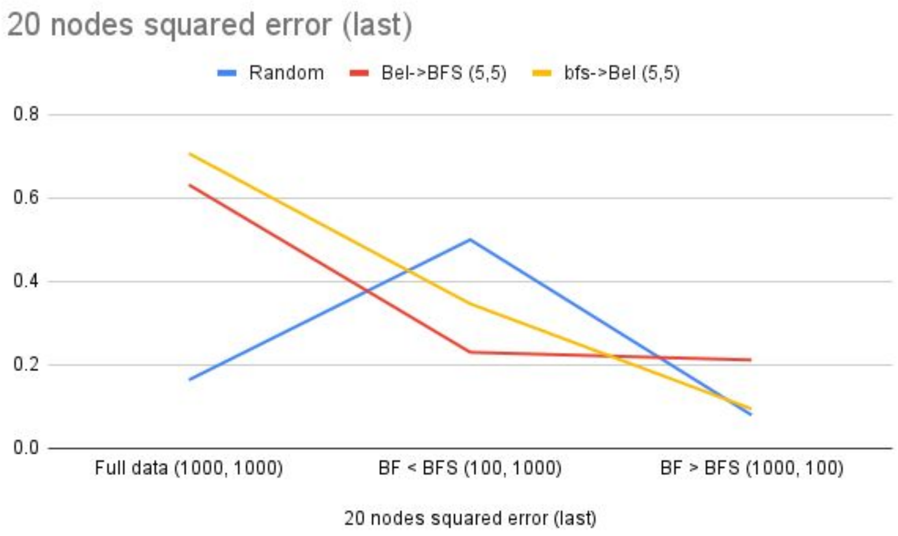
Data Imbalanced Setting



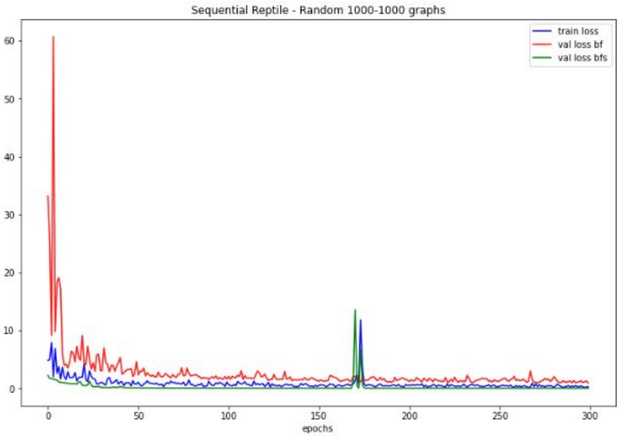


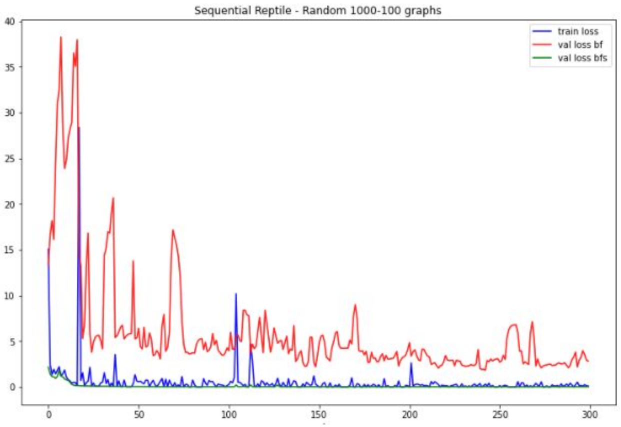
Data Imbalanced Setting

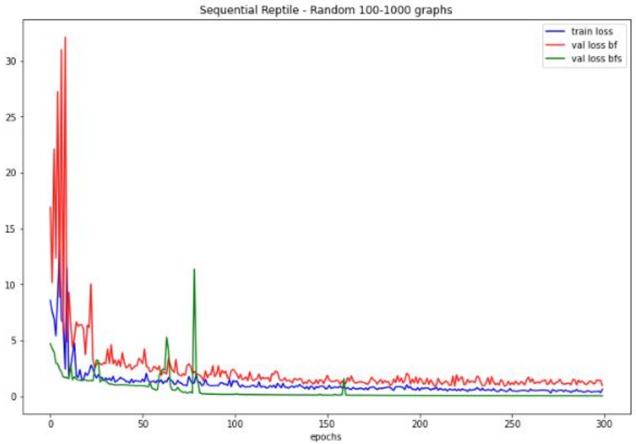




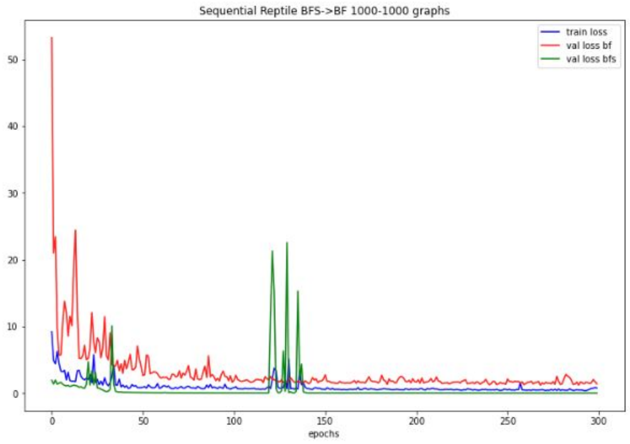
Training vs Validation Loss Curve: Random

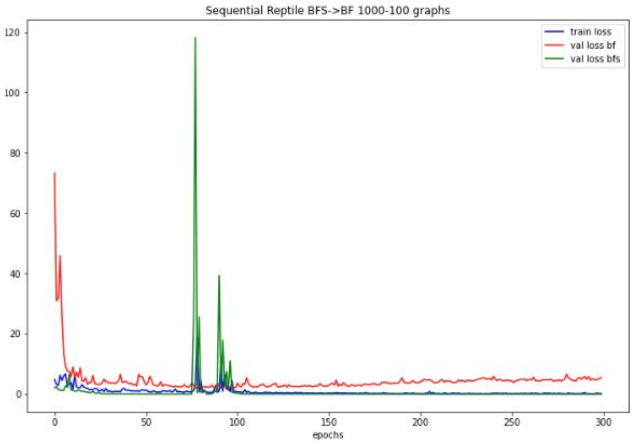


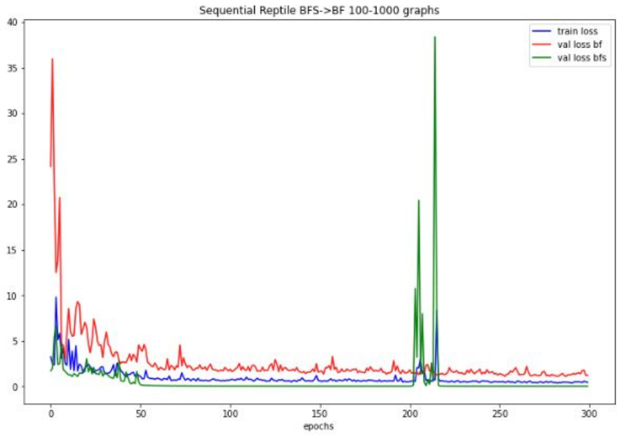




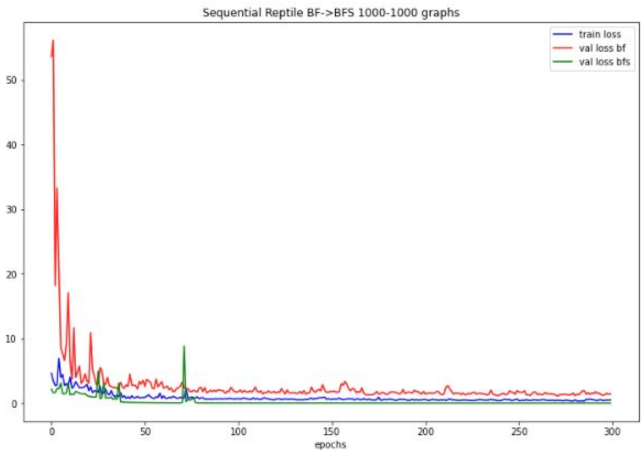
Training vs Validation Loss Curve: BFS->BF

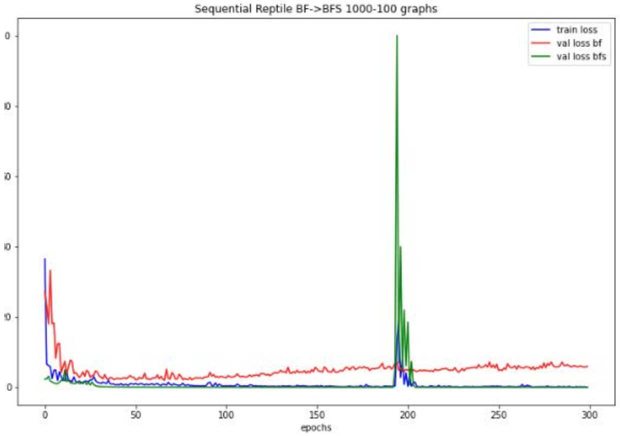


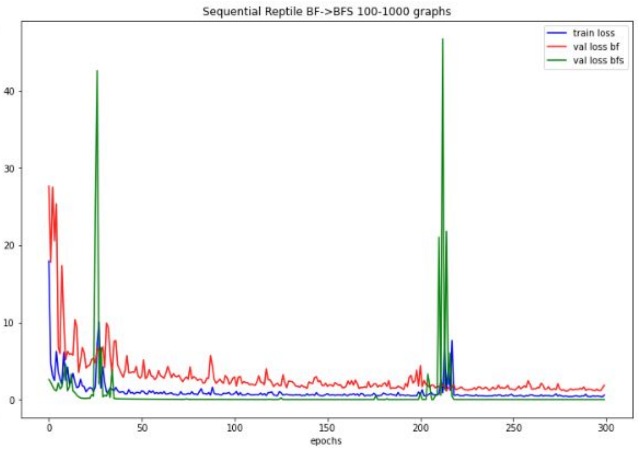




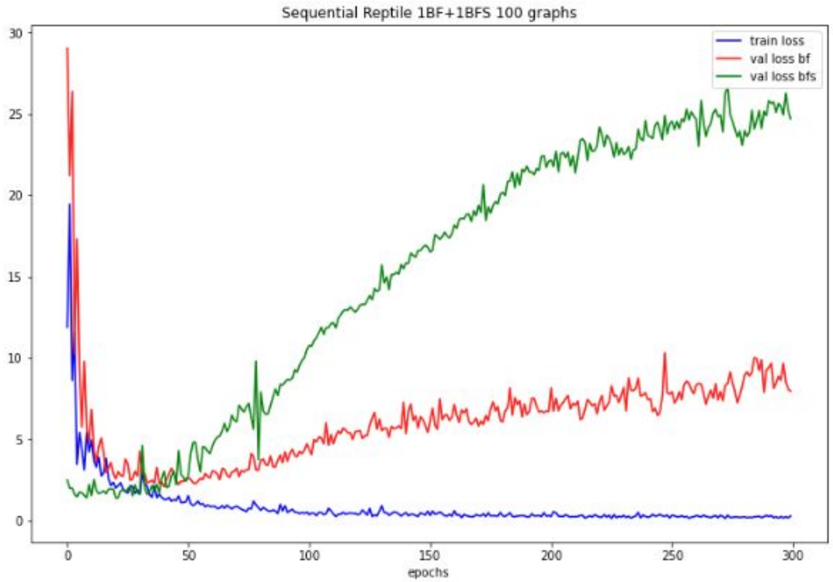
Training vs Validation Loss Curve: BF->BFS

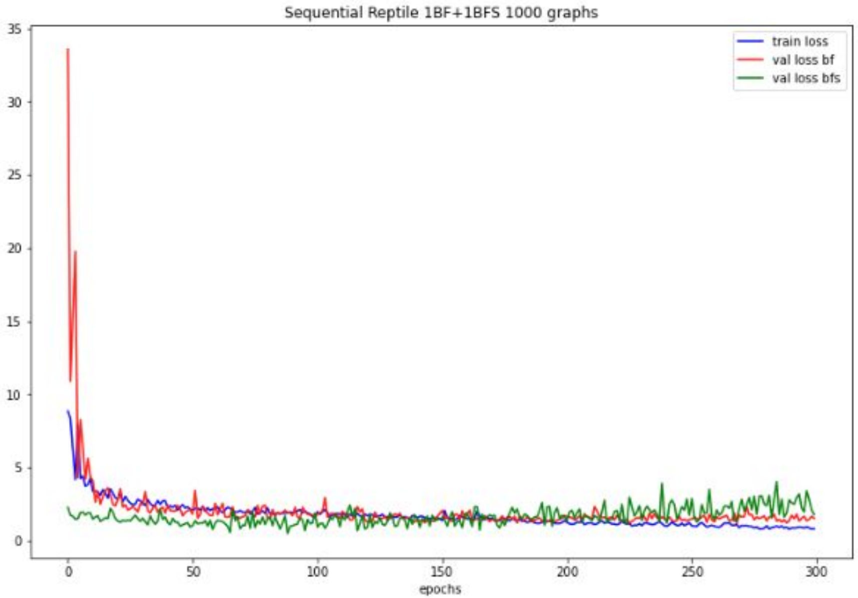




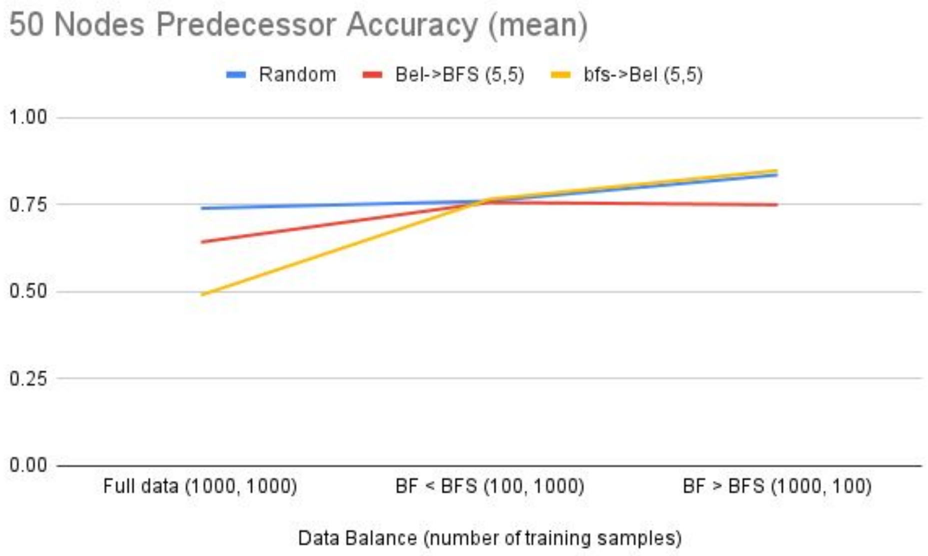


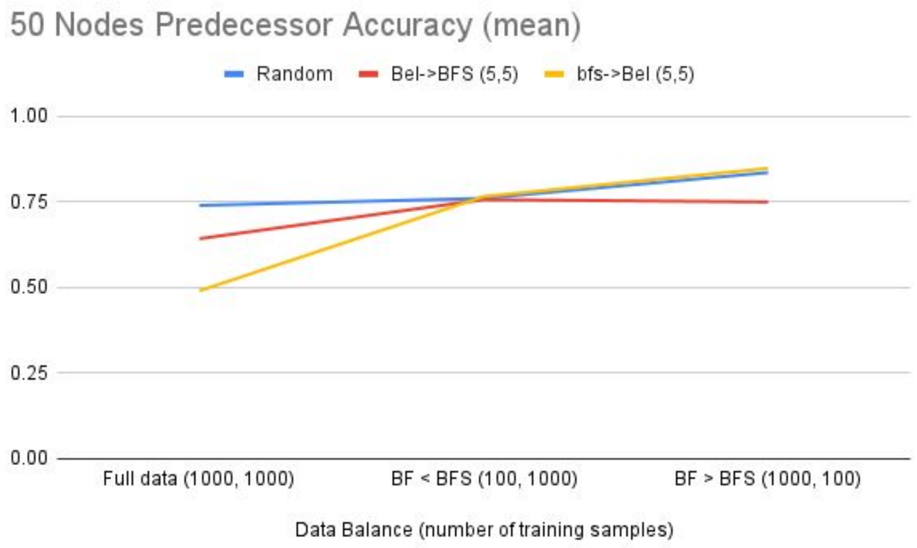
Training vs Validation Loss Curve:1BF+1BFS



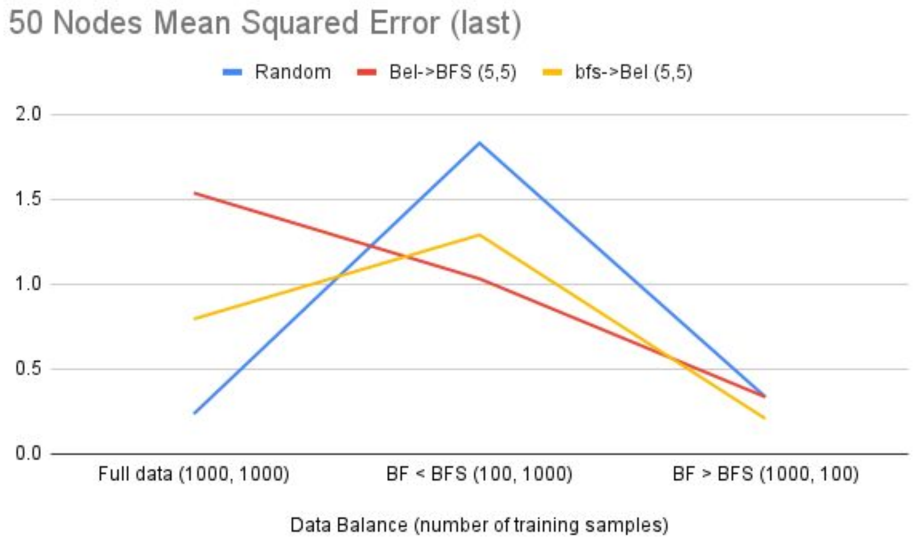


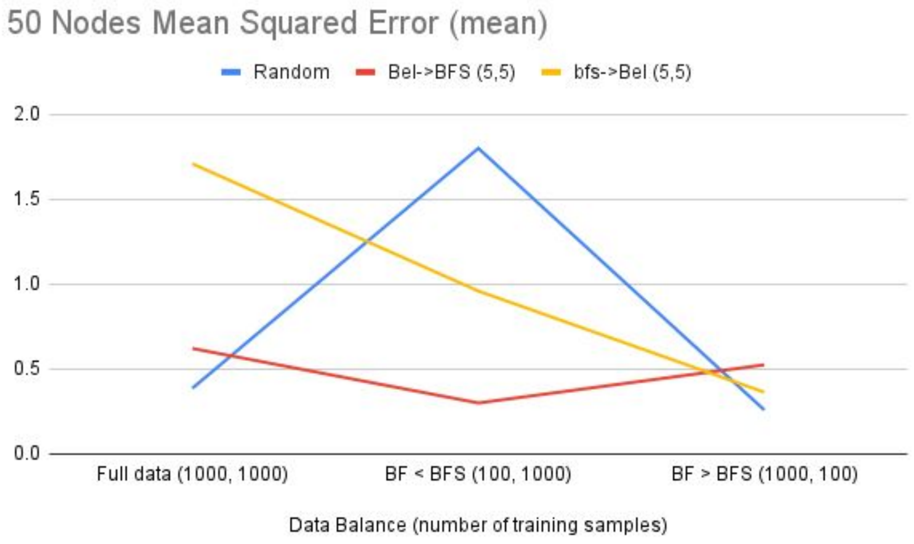
Data Imbalanced Setting





Data Imbalanced Setting





Future Works

- Compare the convergence rate of Sequential Reptile with vanilla multitask learning in the context of neural graph execution
    - The number of graphs needed to achieve comparable performance
- Make Sequential Reptile more suitable for the context of neural graph execution
    - Currently for each step in the inner loop, the distribution the next task is sampled from a distribution based on the sizes of training datasets for each task
    - Can utilize domain specific knowledge to serve as a prior for the task distribution. E.g. use GNN to extract features of the graphs in the training set and utilize these features
- Experiment with training on more algorithms
    - Parallel: Widest
    - Sequential: Prims, Dijkstra
- Experiment with other multitask learning frameworks
    - Adaptive scheduling using validation performance to leverage the benefits of ordering tasks

Thank You

References

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, Charles Blundell. "Neural Execution of Graph Algorithms."

Louis-Pascal A. C. Xhonneux , A. Deac, P. Velickovic, and J. Tang. "How to transfer algorithmic reasoning knowledge to learn new algorithms?"

P. Veliˇckovi´c, A. P. Badia, D. Budden, R. Pascanu, A. Banino, M. Dashevskiy, R. Hadsell, and C. Blundell."The clrs algorithmic reasoning benchmark."

Alex Nichol, Joshua Achiam, John Schulman. "On First-Order Meta-Learning Algorithms."

Seanie Lee, Hae Beom Lee, Juho Lee, Sung Ju Hwang. "Sequential Reptile: Inter-Task Gradient Alignment for Multilingual Learning."

Related Work

- Multi-task learning has shown success in applications such as:
    - Natural language processing (learning multiple languages simultaneously) Xue et al.
    - Visual classification X. Yuan and S. Yan
    - Identifying influenza variants Han et al.
- Sequential reptile can improve the optimization process of multi-task learning in these instances
- More specific to our case, effectively applying sequential reptile to multi-task learning with graph neural networks can open doors to other multi-task graph problems (e.g. molecule generation, physical simulations, social networks)