Simulation of Graph Algorithms with Looped Transformers

Artur Back de Luca, Kimon Fountoulakis

March 21, 2025

University of Waterloo

Presented by: George Giapitzakis

Agenda

- ➔ Introduction & Motivation
- ➔ Architecture: Looped Transformers with Graph Attention
- ➔ Simulation Examples
- ➔ Theoretical Results
- ➔ Training Limitations
- ➔ Conclusion & Future Work

Introduction

Key Question: Can neural networks *simulate* classical graph algorithms?

- Transformers excel in NLP, vision, but algorithmic reasoning is understudied.
- Goal: Prove looped transformers can simulate algorithms like BFS, Dijkstra, SCC.
- **Key Insight**: Encode graphs via adjacency matrices in attention mechanisms.

What is "Simulation"?

For every algorithmic step, the transformer produces the correct output.

- Example: Dijkstra's edge relaxation $\rightarrow$ transformer updates node distances.

Modifications to Standard Transformer:

➔ **Looped Execution**: Repeatedly apply transformer until termination.

➔ **Graph Attention Heads**: Interact with adjacency matrix A:

$$\psi^{(i)}(X, \tilde{A}) = \tilde{A}\,\sigma(XW_Q W_K^\top X^\top)XW_V$$

➔ Avoids storing A in input $\Rightarrow$ parameters stay constant w.r.t. graph size.

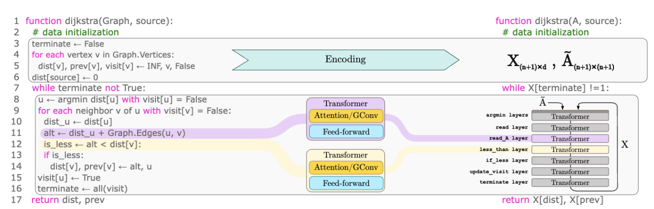

Figure 1: Simulation of Dijkstra's algorithm using Looping.

Simulation Examples: Less-than & if-else Operations

Key Subroutines: Compare node distances in Dijkstra's algorithm and conditionally select values.

- ➔ Approximate $x < y$ using ReLU and tolerance ϵ:

$$x < y \approx \frac{\phi(y - x) - \phi(y - x - \epsilon)}{\epsilon}$$

- ➔ Implemented via MLP layers with scratchpad memory.
- ➔ Approximate $\texttt{if-else}(c_1, c_0, \gamma)$ as

$$\texttt{if-else}(c_1, c_0, \gamma) \approx \phi(c_0 - \gamma\Omega) + \phi(c_1 - (1 - \gamma)\Omega)$$

when $c_1, c_2 \in [-\Omega, \Omega]$.

Theoretical Results

By Constructive Proofs: Networks simulate algorithms step-by-step.

Main Theorems

- **Dijkstra**: 17 layers, 3 heads, $O(1)$ width. Handles weighted graphs.
- **DFS/BFS**: 15/17 layers, $O(1)$ width. Queue/stack emulated via priorities.
- **SCC (Kosaraju)**: 22 layers, 4 heads, $O(1)$ width. Uses two DFS passes.

Key Limitations:

- Finite precision restricts graph size (angular encoding of nodes).
- Maximum edge weight value Ω bounds conditional operations.

Turing Completeness

Result: Looped transformers with graph attention are Turing complete.

- Simulate **SUBLEQ** (single-instruction computer) via:
  - Reduction to **Graph-SUBLEQ**, a Turing complete variant of SUBLEQ.
  - Simulation of Graph-SUBLEQ with looped transformers with graph attention.
- Requires 11 layers, 3 heads, $O(1)$ width.

```
Instruction  subleq a, b, c
    Mem[b] = Mem[b] - Mem[a]
    if (Mem[b] ≤ 0)
        goto c
```

Figure 2: SUBLEQ's only instruction.

Training Limitations

Why is Training Hard?

- ➔ Discontinuous operations (e.g., conditional jumps) cause ill-conditioning.
- ➔ Sharp transitions in loss landscape hinder gradient-based optimization.
- ➔ Empirical validation shows perfect simulation, but parameter recovery is fragile.

Takeaway

Theoretical existence $\neq$ practical learnability. New algorithms may avoid discontinuities.

Conclusion & Future Work

Summary:

- ➔ Looped transformers with graph attention simulate graph algorithms *exactly*.
- ➔ Constant width enables generalization across graph sizes.
- ➔ Turing completeness shown via SUBLEQ simulation.

Future Directions:

- ➔ PAC-learning framework for algorithmic reasoning.
- ➔ Scaling to more complex algorithms (e.g., max flow).
- ➔ Bridging theory-practice gap in training.

Questions?

Thank you!

Any Questions?