A Framework for Differentiable Discovery Of Graph Algorithms

Hanjun Dai, Xinshi Chen, Yu Li, Xin Gao and Le Song

Google Research & Georgia Tech & KAUST
Combinatorial optimization over graph

Repeat till all edges covered:
• Select uncovered edge with largest total degree

NP-hard problems

Manually designed policy. Can we learn from data?

Minimum vertex cover
\[
\min_{x_i \in \{0,1\}} \sum_{i \in V} x_i \\
\text{s.t. } x_i + x_j \geq 1, \forall (i, j) \in E
\]
GNN = Parametrized distributed local graph algorithm

Distributed Local Graph Algorithm =
Graph Representation + Iterative Update

\[ h_i \leftarrow \mathcal{T} \left( X_i, \{ h_j \}_{j \in N(i)} \right) \]
Differentiable Algorithm Discovery (DAD) framework

1. Can we discover new algorithms?

2. How to learn these algorithms?
   - Supervised, Unsupervised
   - Reinforcement Learning

3. Can we interpret what’s discovered?

### Graph Instance with Augmented Global Features

- **Spanning Tree 1**
- **Spanning Tree N**
- **Greedy Solution**

### Learn Graph Neural Networks

### Explainable Model
Search (Input) Space Design for GNN
Motivating example

- The best known algorithm for solving a general linear system takes time $O(n^{2.373})$
- Kelner et al. (2013) proposed an algorithm for solving Laplacian system:

$$Lx = b,$$

where $L$ is Laplacian matrix

in nearly-linear time.

Step 1: Find a low-stretch spanning tree and obtain an initial solution on the tree.

Step 2: Refine the initialized solution by iteratively operating on local cycles in the original graph.
Cheap Solution as Global Features

input graph $G$

spanning tree

optimal solutions $y$ over tree with dynamic programming

Binary edge feature

Binary node feature
Cheap Solution as Global Features

Input graph $G$

Spanning trees $T(1), \ldots, T(n)$

Optimal solutions over trees $y^{T(1)}, \ldots, y^{T(n)}$
Cheap Solution as Global Features

Optimal solutions on \( N \) spanning trees

Approximate solutions obtained via \( M \) greedy algorithms on original graph

Global Features

Spanning Tree 1

\vdots

Spanning Tree \( N \)

\vdots

Greedy Solution 1

\vdots

Greedy Solution \( M \)

Augment

Graph Instance with Augmented Global Features
Learning Local Iterative Algorithms with GNN
Supervised

- For each graph $G$, a solution $y^*$ is obtained by running expensive solver

- Learn GNN-based algorithm which can imitate $y^*$ but runs much faster
Unsupervised

Many graph problems can be formulated as integer programming (IP) problems:

\[
\min_{y \in \{0,1\}^{|V|}} f(Y; G) \quad \text{subject to} \quad g_i(y; G) \leq 0 \quad \text{for } i = 1, \ldots, l
\]

Construct unsupervised training loss based on optimization objective \( f \) and constraints \( g \)

\[
L_U(p, G) := E[f(Y; G)] + \beta \cdot P[g_i(Y; G) \leq 0]
\]

where \( Y \sim \text{Bernoulli}(p) \)
Better learned algorithms with global information

- Comparison of our features to other features
  - Random features
  - Random one-hot encoding
  - Port Numbering + Weak 2-coloring (CPNGNN)

Our approach is consistently better

Minimum Vertex Cover
A deeper understanding of the performance

- Extrapolation
  - train on small graphs
  - test on graphs up to 1024 nodes

Minimum Vertex Cover on Barabasi Albert random graphs
Explain the Learned GNN Algorithms
Explainer

Learned Algorithm:

GNN

Output Layer

Node selection probability

\( p_i = \text{Output}(h_i^{(T)}) \)

\( Y_i \sim \text{Bernoulli}(p_i) \)

\( (V_s^i, E_s^i, S_s^i) \)

Selected Graph Structure & Features

T-hop subgraph
**Explainer architecture**

**Learned Algorithm:**

- **T-hop subgraph**
- **Node embeddings**

**Explainer:**

- **GNN**
- **MLP1**
- **MLP2**
- **MLP3**

**Node/Edge/Feature Scores**

- **Top k1**
- **Top k2**
- **Top k3**

**Selected Graph Structure & Features**

\[(V^i_S, E^i_S, S^i_S)\]

**Node selection probability**

\[p_i = \text{Output} \left(h_i^{(T)} \right)\]

\[Y_i \sim \text{Bernoulli}(p_i)\]
Information theoretic learning

Learned Algorithm:

Node embeddings

GNN Output Layer

Node selection probability

$\mathbf{p}_i = \text{Output}(h_i^{(T)})$

$Y_i \sim \text{Bernoulli}(p_i)$

T-hop subgraph

Node/Edge/Feature Scores

MLP1

Top k1

Node selection probability

$\mathbf{p}_i = \text{Output}(h_i^{(T)})$

$Y_i \sim \text{Bernoulli}(p_i)$

maximize mutual information

$\text{MI}(Y_i; (V^i_S, E^i_S, S^i_S))$

Selected Graph Structure & Features

Explainer:
Discovery of greedy-like behavior

Explanation setting:
- limit to 5 nodes and 10 edges to explain each target node

Minimum Vertex Cover

Greedy like behavior on some targets!

Takeaway:
- Greedy heuristics are the best performing ones on these tasks
- GNN understands and learns the meaning of greedy algorithm features
Discovery of anchor nodes

What global features are effective?

Minimum Vertex Cover

Max-Cut

Budget = 2

Budget = 3
Q/A
Discovery of anchor nodes

- Node color: the darker, the more frequent of being selected for explanation

Observation
- There exists a set of "anchor nodes"
- Anchor nodes tend to be diverse

Hypothesis
- Anchor nodes are like "landmarks" in the graph
- GNN compares the target node with anchor nodes to make prediction

Connections: GNN with anchor nodes: Position/distance aware GNNs (You et.al, 2018; Li et.al, 2020)
What global features are effective?

**Explanation setting:**
- Max-cut problem
- Limit to 2 or 3 global features

**Budget = 2**
- Greedy is always selected;
- AKPW or max-spanning tree can be selected with equal chances;
- Two max-spanning tree solutions will not be selected at the same time;

**Budget = 3**
- Feature selection is consistent across different target nodes;
- The \{Greedy, AKPW, Max-Spanning\} are the best performing three;
- Again, two Max-spanning trees solutions will not appear at the same time, even though itself performs better than AKPW;