Machine Learning for discrete optimization: Graph Neural Networks, generalization under shifts, and loss functions

Stefanie Jegelka
MIT

based on joint work with
Keyulu Xu, Ching-Yao Chuang, Nikolaos Karalias, Joshua Robinson, Andreas Loukas, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi
Learning and Algorithms

- Predicting good configurations / solvers

- Predicting problem parameters

- Machine learning oracles within a fixed algorithm
  (online algorithms, branch-and-bound,…) 

- Learning a full algorithm, Algorithm implemented as a neural network

Figures: Bengio, Lodi, Provost 2020
Why learn an algorithm?

• Adapt to specific input distributions (without analytic form)
• Helping abstraction of “natural” inputs, e.g., infer costs/weights (Veličković & Blundell 2021, Lombardi & Milano 2018)
• Bypass expensive exact algorithm
• Discover new heuristics/algorithms, augment expert knowledge (Xhonneux et al 2021, Veličković et al 2020)

• Many other ML “reasoning” tasks are algorithmic in nature…

Figure: Veličković & Blundell. Neural Algorithmic Reasoning. Patterns, 2021
Today:
Some thoughts on learning (for) algorithms

Disclaimer

• Learning for algorithms is still a nascent field
  (“Although most of the approaches [...] are still at an exploratory level of deployment [...] , we strongly believe that this is just the beginning of a new era for combinatorial optimization algorithms.” (Bengio et al 2020))

• Not the “latest & greatest” here…

• … but some ideas to better understand ML models’ behavior
How to do learning for algorithms?

\[ \min_x c^T x \]
\[ Ax \leq b \]
\[ l \leq x \leq u \]
\[ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \]

Focus here:  
Input is a graph with attributes

(Gasse et al 2019)  
(Selsam et al 2018)
Setup: learning task

“graph regression”

• **Input** $x$: graph (or set) with attributes (optimization instance)

• **Desired output** $g(x)$: optimal value

• **Training data**: $\{(x^{(i)}, g(x^{(i)}))\}_{i=1}^N$ with $x^{(i)} \sim P \leftarrow P$ unknown

• **Goal**: find a function $f \in \mathcal{F}$ that generalizes (= low “risk”)

$$\mathbb{E}_{x \sim P} [\ell(f(x), g(x))]$$

Other aspects:
- learning setting
- loss
- constraints
Learning

Training data:

\[
\min_{f \in F} \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), g(x_i))
\]

“Learn” function

\[
\hat{f}(\cdot) = \text{predicted label}
\]
Learning

Training data:

Test data:

Evaluation: generalization
Outline

• Graph Neural Networks

• **To what kinds of instances will my model generalize?**
  Prediction under distribution shifts
  • **stability**: measuring data shifts appropriately
  • **large shifts**: understanding model behavior by decomposition

• Beyond regression: extending set functions as loss functions for neural networks
**Idea:**

1. Encode each node (node’s neighborhood): *node embedding*
2. Aggregate set of node embeddings into a *graph embedding*
Node embedding: message passing

In each round $k$:

**Aggregate** over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)$$

**Update**: Combine with current node

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, m_{\mathcal{N}(v)}^{(k)}\right)$$

$$h_v^{(0)} = x_v, \quad \forall v \in V \quad h_v^{(t)} \in \mathbb{R}^{d_t}$$
In each round $k$:

**Aggregate** over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \sum_{v \in \mathcal{N}(u)} \text{MLP}^{(k)}(h_u^{(k-1)}, h_v^{(k-1)}, w_{(v,u)})$$

**Feature description of node $u$ in round $k-1**

(Merkwirth & Lengauer 2005; Scarselli et al 2009; Bruna et al 2014; Dai et al 2016; Battaglia et al., 2016; Defferrard et al., 2016; Duvenaud et al., 2015; Hamilton et al., 2017; Gilmer et al 2017; ...)

(Verma & Zhang, 2018; Ying et al., 2018; Zhang et al., 2018; ...)

Node embedding: message passing
Message passing unrolled

In each round $k$:
**Aggregate** over neighbors and update representation

“Unrolled”: **computation tree**

*Structured arrangement of learnable “modules”*
Outline

• Graph Neural Networks

• **To what kinds of instances will my model generalize?**
  Prediction under distribution shifts
  • **stability:** measuring data shifts appropriately
  • **large shifts:** understanding model behavior by decomposition

• Beyond regression: extending set functions as loss functions for neural networks
Graph predictions and distribution shifts

\[ \mathbb{E}_{x \sim Q} [\ell(f_\theta(x), g(x))] \]

\[
\text{support}(Q) \supset \text{support(\text{training dist})}
\]

\textbf{different graph size, }
\textbf{graph structure, edge weights, …}

\textbf{Physical reasoning}
\textbf{different position, mass, number of objects}
Graph predictions and distribution shifts

When may this work?

\[ \mathbb{E}_{x \sim Q} [\ell(f_\theta(x), g(x))] \]

support(Q) \supset support(train dist)
Big picture: when may extrapolation “work”? 

1) Data distributions in training and test are sufficiently similar 
   same distribution of computation trees (message passing GNNs) (Yehudai-Fetaya-Meirion-Chechik-Maron 21)
   shared underlying structure (spectral GNNs) (Levie et al 2019, Ruiz et al 2020)

   --- or … ---

2) Understand what the model “learns”, and work around that:
   restrict the model via prior knowledge
   (Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21)

   Neural network structure, optimization algorithm, data geometry
Graph predictions and distribution shifts

- **Worst-case scenario**: arbitrary predictions on unseen computation trees

\[ \mathbb{E}_{x \sim \mathcal{Q}}[\ell(f_\theta(x), g(x))] \]

\[
\text{support}(\mathcal{Q}) \supset \text{support}(\text{training dist})
\]

**Theorem (Yehudai et al 2021)**: Let \( \mathcal{P} \) and \( \mathcal{Q} \) be finitely supported distributions on graphs, and \( \mathcal{P}^t \), \( \mathcal{Q}^t \) the distribution of computation trees at depth \( t \). If any graph in \( \mathcal{Q} \) contains a tree in \( \mathcal{Q}^t \setminus \mathcal{P}^t \), then there is a GNN with depth at most \( t + 3 \) that perfectly solves the task on \( \mathcal{P} \) but has arbitrarily large error on all graphs from \( \mathcal{Q} \).

Smöother degrading performance with appropriate metric on graphs?
Stability and measuring perturbations: Tree mover’s distance

C. Chuang, S. Jegelka. Tree Mover’s Distance: Bridging Graph Metrics and Stability of Graph Neural Networks. NeurIPS, 2022
An appropriate metric?

- Metric should capture Lipschitz/stability properties of GNNs, including invariances

Message passing GNN compares sets of computation trees (subtree patterns)
- Idea: Optimal transport distance
Tree mover’s distance

- Earth mover’s distance between sets in Euclidean space:

\[
\min_{\gamma \in \Gamma(X,Y)} \sum_{i,j=1}^{n,m} d(x_i, y_j) \cdot \gamma_{ij}
\]

\[
\Gamma(X, Y) = \{ \gamma \in \mathbb{R}^n_{+}^{m} | \gamma 1_m = 1_n; \gamma^\top 1_n = 1_m \}\]

- Tree mover’s distance: \(x_i, x_j\) are trees.

Distance between trees?
Tree Distance via Hierarchical OT

\[
\text{Tree Distance via Hierarchical OT} = \|x_{r_a} - x_{r_b}\| + \text{OT}_{TD_w}(\rho(T_{r_a}, T_{r_b}))
\]

where \( L = \max(\text{Depth}(T_a), \text{Depth}(T_b)) \) and \( w : \mathbb{N} \to \mathbb{R}^+ \) is a depth-dependent weighting function.
Properties & implications of Tree mover’s distance

- pseudo-metric: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!

- relation to stability of GNNs: Lipschitz constant of GNN (GIN)

\[
\| h(G_a) - h(G_b) \| \leq \prod_{l=1}^{L+1} K^{(l)}_\phi \cdot \text{TMD}_{L+1}^1(G_a, G_b)
\]

- use TMD in cross-domain generalization bound (Shen et al, 2018):

\[
R_T(h) \leq R_S(h) + 2 \text{Lip}(h) \cdot W_1(p_S, p_T) + \text{small value}
\]
Empirically

- comparison with Wasserstein Weisfeiler-Leman (WWL) metric (Togninalli et al 2019)

1. **stability:**

   MUTAG data, randomly sampled pairs

2. correlation with accuracy drops under domain shifts (PTC data):

   - WWL: 0.489
   - TMD: 0.712
Large perturbations: inductive biases

Generalization to very different data

Different graph size, graph structure, edge weights, ... 

Physical reasoning

different position, mass, number of objects
Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar
   - same distribution of computation trees (message passing GNNs) (Yehudai-Fetaya-Meirn-Chechik-Maron 21)
   - shared underlying structure (spectral GNNs) (Levie et al 2019, Ruiz et al 2020)

   --- or … ---

2) Understand what the model “learns”, and work around that:
   - restrict the model via prior knowledge (Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21)

   Neural network structure, optimization algorithm, data geometry
Neural network architecture and algorithms

- Algorithm = structured arrangement of subroutines
- (Graph) Neural network = structured arrangement of learnable “modules”

**Bellman-Ford**

\[
\text{for } k = 1 \ldots |S| - 1: \\
\quad \text{for } u \in S:\n\]

\[
d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)
\]

**GNN**

\[
\text{for } k = 1 \ldots \text{GNN iter:} \\
\quad \text{for } u \in S:\n\]

\[
h_u^{(k)} = \Sigma_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})
\]

**Algorithmic Alignment:**
Neural Network can mimic algorithm via few, easy-to-learn “modules”
Empirically: MLP with ReLU activations

\[ h^{(k)}_{v_i} = \sum_{v_j \in N(i)} \text{MLP}^{(k)}(h^{(k-1)}_{v_i}, h^{(k-1)}_{v_j}) \]
Extrapolation in fully connected ReLU networks

**Theorem** *(Xu-Zhang-Li-Du-Kawarabayashi-J 21)*

Let $f$ be a 2-layer ReLU MLP trained with Gradient Descent. Along any direction $v \in \mathbb{R}^d$, $f$ approaches a linear function: let $x = tv$. As $t \to \infty$: $f(x + hv) - f(x) \to \beta_v h$ with rate $O(1/t)$.

(Linear regions: Montufar et al 2014, Arora et al 2018, Hanin & Rolnick, 2019; Hein et al., 2019, XZDKJ20)
Implications

1. Can only extrapolate linear functions

2. Training Data geometry
Implications for the full GNN

Shortest Path: \[ \text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v) \]

GNN:

\[ h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v)) \]

GNN II:

\[ h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w) \]

(Veličkovic et al. 2020)

Need MLP to be nonlinear!
Implications for the full GNN

Shortest Path: \[ \text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v) \]

GNN: \[ h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v)) \]

GNN II: \[ h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w) \]

Task-specific nonlinearities help extrapolation.
Empirically reflected in many works

Need MLP to be nonlinear!

(Veličkovic et al 2020)
Encode nonlinearities in the ...

... architecture

NALU: \[ y = g \odot a + (1 - g) \odot m \]

\[ m = \exp W(\log(|x| + \epsilon)), \quad g = \sigma(Gx) \]

Exp log for learning multiplication

(Trask et al 2018)

Library of programs

(Johnson et al 2017, Yi et al 2018, Mao et al 2019, ...)

Learning physics laws

(Cranmer et al 2019, 2020)

... input representation

Prior knowledge or representation learning

(figure of n-body system: Battaglia et al 2018)
Neural Network Losses from Set Function Extensions

Setup

- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function?  \( F(S), S \subseteq [n] \)

Continuous extension:

\[
F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \rightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}
\]

Want that:

- \( f \) is continuous
- \( f(1^S) = F(S) \)

Strategy:

\[
f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)
\]

**Example: Lovász extension**

of submodular set function

\[
p_x(i \in S') = x_i
\]
Extensions: higher-dimensional

**NN outputs…**

\[ S \in \{0, 1\}^n \quad F(S) \]

\[ x \in [0, 1]^n \quad f(x) = \sum_{S \subseteq [n]} p_x(S) F(S) \quad f(1_S) = F(S) \]

\[ X \in S_+ \quad f(X) = \sum_{S,T \subseteq [n]} p_X(S,T) F(S \cap T) \quad f(1_S 1_S^\top) = F(S) \]
Derivation in a nutshell

- vector extension for \( x \in [0, 1]^n \): \( p_x(S') \) solution to dual of “\( f \) is a convex envelope of \( F \):”

\[
\min_{\{y_S \geq 0\} S \subseteq [n]} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S 1_S = x, \sum_{S \subseteq [n]} y_S = 1
\]

\( \Rightarrow \) marginals

\( p_x(i \in S') = x_i \)
Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_X(S')$ solution to dual of “$f$ is a convex envelope of $F$”:

  \[
  \min_{\{y_S \geq 0\} S \subseteq [n]} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S 1_S = x, \quad \sum_{S \subseteq [n]} y_S = 1
  \]

- use SDP version of this

- any valid vector extension leads to a valid matrix extension: Let $X = \sum_i \lambda_i v_i v_i^T$

  \[
  \sum_{S, T \subseteq [n]} p_X(S, T) F(S \cap T) \quad \text{with} \quad p_X(S, T) = \sum_i \lambda_i p_{v_i}(S)p_{v_i}(T)
  \]
Empirical results

**max clique**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ENZYMES</th>
<th>PROTEINS</th>
<th>IMDB-Bin</th>
<th>MUTAG</th>
<th>COLLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximation ratio</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
<td>1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

**k-clique**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ENZYMES (k = 3)</th>
<th>PROTEINS (k = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1-score</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Approximation ratio**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ENZYMES</th>
<th>PROTEINS</th>
<th>IMDB-Bin</th>
<th>MUTAG</th>
<th>COLLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1-score</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>
GNNs for Learning for Combinatorial Optimization

- Active, recent area, but need to understand learned functions

- **To what examples will my models generalize?** - understanding the data space metric from the GNN perspective

- **What are important architectural choices?** - understanding the model nonlinearities and alignment with algorithms

- **How to train the model?** - choice of loss function higher-dimensional extensions of set functions tend to work better
\[ \tilde{f}(\mathbf{x}) = \max_{\mathbf{z}, b \in \mathbb{R}^n \times \mathbb{R}} \{ \mathbf{x}^\top \mathbf{z} + b \} \text{ subject to } \mathbf{1}_S^\top \mathbf{z} + b \leq f(S) \text{ for all } S \subseteq [n]. \] (primal LP)

\[ \tilde{f}(\mathbf{x}) = \min_{\{y_s \geq 0\} S \subseteq [n]} \sum_{S \subseteq [n]} y_S f(S) \text{ subject to } \sum_{S \subseteq [n]} y_S \mathbf{1}_S = \mathbf{x}, \sum_{S \subseteq [n]} y_S = 1, \text{ for all } S \subseteq [n], \] (dual LP)

\[ \max_{\mathbf{z} \geq 0, b \in \mathbb{R}} \{ \text{Tr}(\mathbf{X}^\top \mathbf{Z}) + b \} \text{ subject to } \frac{1}{2} \text{Tr}((\mathbf{1}_S \mathbf{1}_T^\top + \mathbf{1}_T \mathbf{1}_S^\top)\mathbf{Z}) + b \leq f(S \cap T) \text{ for } S, T \subseteq [n], \] (primal SDP)

\[ \min_{\{y_{s,t} \geq 0\}} \sum_{S,T \subseteq [n]} y_{s,t} f(S \cap T) \text{ subject to } \mathbf{X} \preceq \sum_{S,T \subseteq [n]} \frac{1}{2} y_{s,t} (\mathbf{1}_S \mathbf{1}_T^\top + \mathbf{1}_T \mathbf{1}_S^\top) \text{ and } \sum_{S,T \subseteq [n]} y_{s,t} = 1 \] (dual SDP)