Graph Representation Learning for Optimization on Graphs

Bistra Dilkina

Assistant Professor of Computer Science, USC
Associate Director, USC Center of AI in Society
AI for Sustainability and Social Good

Biodiversity Conservation  Disaster resilience  Public Health & Well-being

Design of policies to manage limited resources for best impact translate into large-scale decision / optimization and learning problems, combining discrete and continuous effects
ML  Combinatorial Optimization

- Exciting and growing research area
- Design discrete optimization algorithms with learning components
- Learning methods that incorporate the combinatorial decision making they inform
Constraint Reasoning and Optimization

Decision making problems of larger size and new problem structure drive the continued need to improve combinatorial solving methods.
## Constraint Reasoning and Optimization

<table>
<thead>
<tr>
<th>Tackling NP-Hard problems</th>
<th>Design rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact algorithms</td>
<td>Tight formulations, good IP solvers</td>
</tr>
<tr>
<td>Approximation algorithms</td>
<td>Worst-case guarantees</td>
</tr>
<tr>
<td>Heuristics</td>
<td>Intuition, Empirical performance</td>
</tr>
</tbody>
</table>

### A realistic setting

- **Same problem** is solved repeatedly with slightly different data
- Delivery Company in Los Angeles:
  - Daily routing in the same area with slightly different customers

### Opportunity:

**Automatically tailor** algorithms to a **family of instances** to **discover novel search strategies**
ML-Driven Discrete Algorithms

ML Paradigm

Self-Supervised Learning

Reinforcement Learning

Supervised Learning

Graph Optimization

Integer Programming

Problem Type

Greedy Heuristic

General IP Heuristic

Exact Solving

Branching

Heuristic Selection

Algorithmic Template: Greedy

- **Minimum Vertex Cover:** Find smallest vertex subset $S$ s.t. each edge has at least one end in $S$
  - Example: advertising optimization in social networks
  - 2-approx:
    - greedily add vertices of edge with max degree sum
Learning Greedy Heuristics

**Given:** graph problem, family of graphs

**Learn:** a scoring function to guide a greedy algorithm

<table>
<thead>
<tr>
<th>Problem</th>
<th>Minimum Vertex Cover</th>
<th>Maximum Cut</th>
<th>Traveling Salesman Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Social network snapshots</td>
<td>Spin glass models</td>
<td>Package delivery</td>
</tr>
<tr>
<td>Greedy operation</td>
<td>Insert nodes into cover</td>
<td>Insert nodes into subset</td>
<td>Insert nodes into sub-tour</td>
</tr>
</tbody>
</table>

Joint work with Elias Khalil, Hanjun Dai, Yuyu Zhang and Le Song [NIPS 2017]
Challenge #1: How to Learn

Possible approach: **Supervised learning**

- **Data**: collect (partial solution, next vertex) pairs
  
  features  label

  from precomputed (near) optimal solutions

**PROBLEM**

Supervised learning → Need to compute good/optimal solutions to NP-Hard problems in order to learn!!
Reinforcement Learning Formulation

Minimum Vertex Cover

\[
\min_{x_i \in \{0, 1\}} \sum_{i \in V} x_i \\
\text{s.t. } x_i + x_j \geq 1, \forall (i, j) \in \mathcal{E}
\]

Start with COVER = empty
Repeat until all edges covered:
1. Compute score for each vertex
2. Select vertex with largest score
3. Add best vertex to COVER

Reward: \( r^t = -1 \)

State \( S \): current partial solution

Action value function: \( \hat{Q}(S, v) \)

Greedy policy:
\( v^* = \arg\max_v \hat{Q}(S, v) \)

Update state \( S \)

SOLUTION

Improve policy by learning from experience → no need to compute optima
Challenge #2: How to Represent

- **Action value function**: $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex $v$ in state $S_t$

- **Representation of $v$: Feature engineering**
  - Degree, 2-hop neighborhood size, other centrality measures…

**PROBLEMS**
1- Task-specific engineering needed
2- Hard to tell what is a good feature
3- Difficult to generalize across diff. graph sizes
We first provide an introduction to structure2vec with a softmax-layer, so that the parameters can be trained end-to-end by minimizing the cross-entropy.

The parameters will be learned. Previously, structure2vec defines the graph topology. A new round of embedding sweeping across the nodes will start only after the node features. One variant of the structure2vec architecture will initialize the embedding $v$.

\[ \mu_v^{(t+1)} \leftarrow \text{relu}(\theta_1 x_v + \text{Node's own tag } x_v) \]

\[ \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^{(t)} + \text{Neighbors’ features} \]

\[ \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v, u)) \]

\[ \text{Neighbors’ edge weights} \]

Theta: model parameters

Repeat embedding $T$ times
We first provide an introduction to

\[ \hat{Q}(S_t, v; \Theta) \]

with a softmax-layer, so that the parameters can be trained end-to-end by minimizing the cross-entropy loss. This approach is not applicable to our case due to the lack of training labels. Instead, we train \( G \), on a collection of 7 parameters \( p \)-dimensional feature embedding over the entire graph, \( Q \), which define the embedding update for all nodes from the previous round has finished. It is easy to see that the update will contain information about its neighbors over nodes is one way of aggregating neighborhood information invariant to the permutation of degree distribution, triangle counts, distance to tagged nodes, etc. In order to represent such complex phenomena over combinatorial structures, we will leverage a deep learning architecture over graphs, specifically, \( \text{structure2vec} \), and the computation graph of \( \Theta \), \( \sum_{u \in V} \mu_{u}^{(T)}, \theta_{7} \mu_{v}^{(T)} \)

**Compute Q-value:**

\[
\hat{Q}(h(S), v; \Theta) = \theta_{5}^{\top} \text{relu}([\theta_{6} \sum_{u \in V} \mu_{u}^{(T)}, \theta_{7} \mu_{v}^{(T)}])
\]

**Sum-pooling over nodes**

\( \Theta \): model parameters
Minimum Vertex Cover - BA

Our Approach [Vinyals et al. 2015] is near-optimal, barely visible.

S2V-DQN is near-optimal, barely visible.
MaxCut - BA

Approximation Ratio

- S2V-DQN
- PN-AC
- SDP
- MaxcutApprox

Our Approach [Vinyals et al. 2015]

Number of nodes in train/test graphs:
- 15-20
- 40-50
- 50-100
- 100-200
- 200-300
TSP - clustered

Approximation Ratio

- S2V-DQN
- 2-opt
- PN-AC
- Cheapest
- Christofides
- Closest
- Nearest
- MST

[Vinyals et al. 2015]
Learning-Driven Algorithm Design

Takeaways

‣ RL tailors greedy search to family of graph instances
‣ Learn features jointly with greedy policy
‣ Human priors encoded via meta-algorithm (Greedy)
The data-decisions pipeline

Many real-world applications of AI involve a common template:

[Horvitz and Mitchell 2010; Horvitz 2010]

Observe data → Predictions → Decisions
Data

Training: maximize accuracy

arg \max_{x \in X} f(x, \theta) \rightarrow \text{Decisions}

Standard two stage: predict then optimize
Data

Decisions

Training: maximize accuracy

Standard two stage: predict then optimize

Challenge: misalignment between “accuracy” and decision quality
Pure end to end: predict decisions directly from input
Pure end to end: predict decisions directly from input

Challenge: optimization is hard to encode in a NN
Decision-focused learning: differentiable optimization during training
Data

Challenge: how to make optimization differentiable?

Decision-focused learning: differentiable optimization during training

Training: maximize decision quality

Decision-focused learning: differentiable optimization during training

Challenge: how to make optimization differentiable?

Decision-focused learning: differentiable optimization during training

Challenge: how to make optimization differentiable?
Relax + differentiate

Forward pass: run a solver

Backward pass: sensitivity analysis via KKT conditions

Linear and submodular programs [Wilder, Dilkina, Tambe 2019]
MAXSAT (via SDP relaxation) [Wang, Donti, Wilder, Kolter 2019]
MIPs [Ferber, Wilder, Dilkina, Tambe 2019]

Some problems don’t have good relaxations
Slow to solve continuous optimization problem
Slow to backprop through – $O(n^3)$
Our Alternative

• Learn a representation that maps the original problem to a simpler (efficiently differentiable) proxy problem.

• Instantiation for a class of graph problems: k-means clustering in embedding space.

Graph learning + graph optimization
Problem classes

- Partition the nodes into K disjoint groups
  - Community detection, maxcut, …

- Select a subset of K nodes
  - Facility location, influence maximization, …

- Methods of choice are often combinatorial/discrete

Approach

- Observation: **clustering nodes** is a good proxy
  - Partitioning: correspond to well-connected subgroups
  - Facility location: put one facility in each community

- Observation: graph learning approaches already embed into $R^n$
ClusterNet Approach

Node embedding (GCN) → K-means clustering → Locate 1 facility in each community
Differentiable K-means

Forward pass

\[ \mu_k = \frac{\sum_j r_{jk} x_j}{\sum_j r_{jk}} \]

\[ r_{jk} = \frac{\exp(-\beta \|x_j - \mu_k\|)}{\sum_{\ell} \exp(-\beta \|x_j - \mu_\ell\|)} \]

Update cluster centers

Softmax update to node assignments
Differentiable K-means

- Option 1: differentiate through the fixed-point condition

\[ \mu^t = \mu^{t+1} \]

- Prohibitively slow, memory-intensive
Differentiable K-means

- Option 1: differentiate through the fixed-point condition
  \[ \mu^t = \mu^{t+1} \]
  - Prohibitively slow, memory-intensive
- Option 2: unroll the entire series of updates
  - Cost scales with # iterations
  - Have to stick to differentiable operations
Differentiable K-means

Backward pass

• Option 1: differentiate through the fixed-point condition
  \[ \mu^t = \mu^{t+1} \]
  • Prohibitively slow, memory-intensive

• Option 2: unroll the entire series of updates
  • Cost scales with # iterations
  • Have to stick to differentiable operations

• Option 3: get the solution, then unroll one update
  • Do anything to solve the forward pass
  • Linear time/memory, implemented in vanilla PyTorch
Differentiable K-means

Theorem [informal]: provided the clusters are sufficiently balanced and well-separated, the Option 3 approximate gradients converge exponentially quickly to the true ones.

Idea: show that this corresponds to approximating a particular term in the analytical fixed-point gradients.
ClusterNet Approach

1. GCN node embedding
2. K-means clustering
3. Locate 1 facility in each community
ClusterNet Approach

GCN node embedding $s$

K-means clustering

Locate 1 facility in each community

Loss: quality of facility assignment
ClusterNet Approach

- GCN node embedding
- K-means clustering
- Locate 1 facility in each community
- Differentiate through K-means
- Loss: quality of facility assignment
ClusterNet Approach

Update GCN params

GCN node embedding $s$

K-means clustering

Differentiate through K-means

Loss: quality of facility assignment

Locate 1 facility in each community
Example: community detection

Observe partial graph

Predict unseen edges

Find communities

\[
\max \text{ modularity} \\
\frac{1}{2m} \sum_{u,v \in V} \sum_{k=1}^{K} \left[ A_{u,v} - \frac{d_u d_v}{2m} \right] r_{uk} r_{vk} \\
r_{uk} \in \{0,1\} \ \forall u \in V, k = 1 \ldots K \\
\sum_{k=1}^{K} r_{uk} = 1 \ \forall u \in V
\]
Example: community detection

### Useful in scientific discovery
- Useful in scientific discovery (social groups, functional modules in biological networks)

### In applications, two-stage approach is common:
- In applications, two-stage approach is common: [Yan & Gregory ’12, Burgess et al ‘16, Berlusconi et al ‘16, Tan et al ‘16, Bahulker et al ’18…]
Experiments

• **Learning problem**: link prediction

• **Optimization**: community detection and facility location problems

• Train **GCNs** as predictive component
Experiments

- **Learning problem**: link prediction
- **Optimization**: community detection and facility location problems
- Train **GCNs** as predictive component

- **Comparison**
  - Two stage: GCN + expert-designed algorithm (**2Stage**)
  - Pure end to end: Deep GCN to predict optimal solution (**e2e**)
Results: single-graph link prediction

Community detection (higher is better)

Facility location (lower is better)

Representative example from cora, citeseer, protein interaction, facebook, adolescent health networks

Community algos: CNM, Newman, SpectralClustering
Facility Locations algos: greedy, gonzalez2approx
ClusterNet learns generalizable strategies for optimization!
Results: optimization only
ClusterNet as a solver

<table>
<thead>
<tr>
<th>Optimization</th>
<th>cora</th>
<th>cite.</th>
<th>prot.</th>
<th>adol</th>
<th>fb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClusterNet</td>
<td>0.71</td>
<td>0.76</td>
<td>0.52</td>
<td>0.55</td>
<td>0.80</td>
</tr>
<tr>
<td>GCN-e2e</td>
<td>0.07</td>
<td>0.08</td>
<td>0.14</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>Train-CNM</td>
<td>0.08</td>
<td>0.34</td>
<td>0.05</td>
<td>0.60</td>
<td>0.80</td>
</tr>
<tr>
<td>Train-Newman</td>
<td>0.20</td>
<td>0.22</td>
<td>0.29</td>
<td>0.30</td>
<td>0.47</td>
</tr>
<tr>
<td>Train-SC</td>
<td>0.15</td>
<td>0.08</td>
<td>0.07</td>
<td>0.46</td>
<td>0.79</td>
</tr>
</tbody>
</table>

ClusterNet learns an effective graph optimization solver!
Takeaways

• Good decisions require integrating learning and optimization
• Pure end-to-end methods miss out on useful structure
• Even simple optimization primitives provide good inductive bias
Augment discrete optimization algorithms with learning components

Learning methods that incorporate the combinatorial decisions they inform
Thank you!

ML ↔ Combinatorial Optimization

- Exciting and growing research area
- Design discrete optimization algorithms with learning components
- Learning methods that incorporate the combinatorial decision making they inform