ETH zürich

Erdős Goes Neural

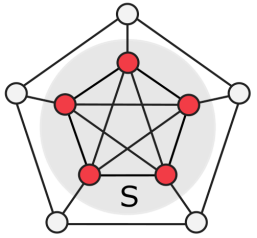
Research Objective

Objective

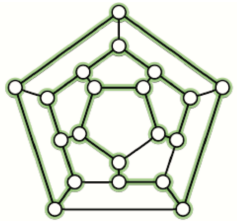
Solve combinatorial problems with an unsupervised learning framework

$$\min_{S \subseteq V} \quad f(S; G) \quad \text{subject to} \quad S \in \Omega$$

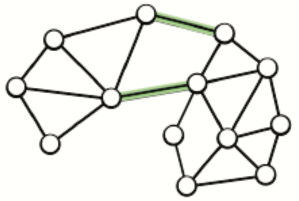
Cost function $S \subseteq V$ $G = (V, E)$ Feasible Set



Maximum Clique

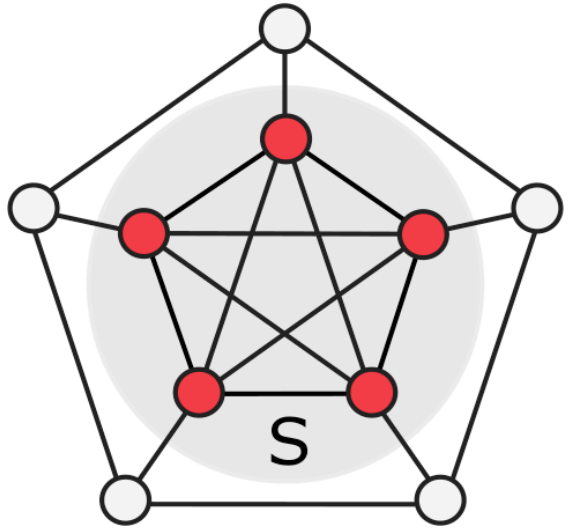


Traveling Salesman



Minimum Cut

Example: Maximum Clique



Objective:

find the largest clique S (fully connected subgraph) within a given graph

$$\min_{S \subseteq V} \quad -|S| \quad \text{subject to} \quad S \in \Omega_{\text{clique}}$$

Why learning?

Many key combinatorial problems are NP-hard.

Learning Based

- Learns heuristics automatically from data
- Generalizes across problem sizes and distributions
- Fast inference after training

Classic algorithms — heuristics

- Hardcoded heuristics – no adaptation to unseen instance distributions
- No learning from previous problem instances
- Require human-engineered problem-specific knowledge

Learning Paradigms

Supervised (Pointer Networks, Neurocore)

- Stable training
- Good performance

- High computational cost for labeling hard instances
- Limited generalization

Reinforcement Learning (S2V-DQN, REINFORCE)

- Flexible
- Straightforward for simple instances
- Combinatorial decision making problems

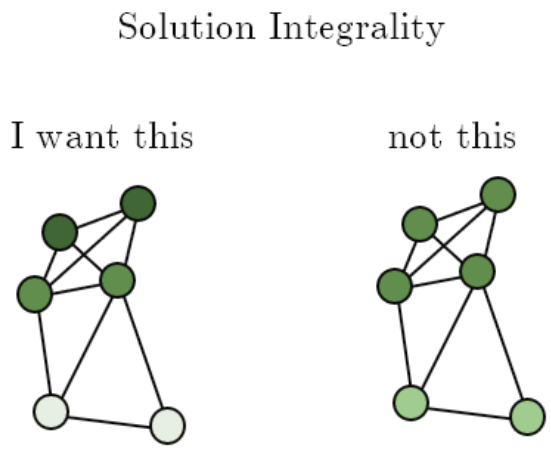
- Training stability
- Convergence issues
- Resource intensive

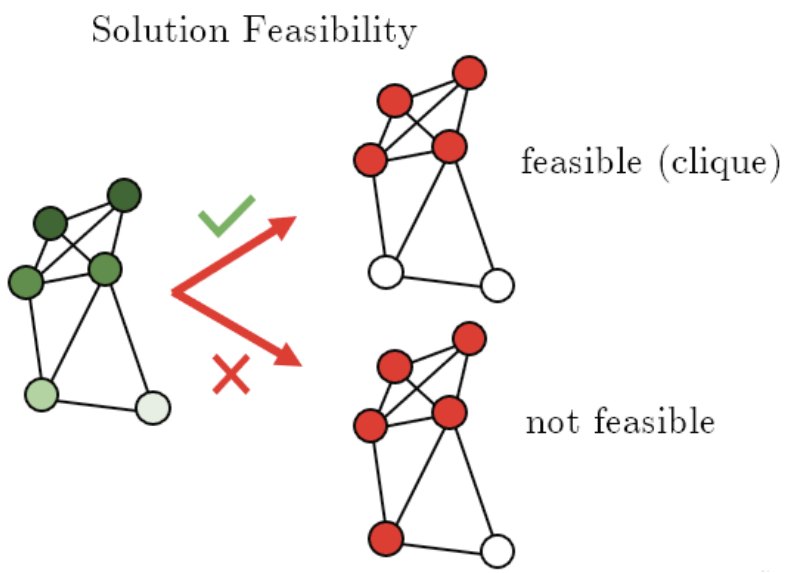
Learning Paradigms

Unsupervised

- Stable training
- Computationally efficient for labeling
- Better generalization

- Hard to obtain an integral solution





Background

Probabilistic Method

A non-constructive technique that proves the existence of a mathematical object by showing that the probability of randomly selecting such an object is greater than zero.

Approach to prove the existence of a set of nodes S with a desired property:

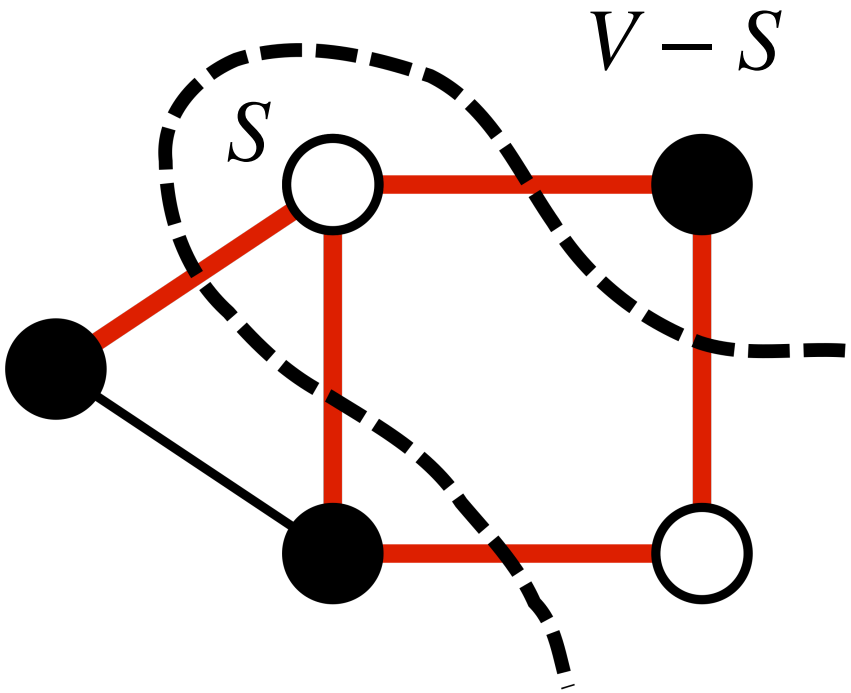
1. Define a probability distribution over the space of all possible set S on the graph

2. Show that $P(S \text{ has property}) > 0$

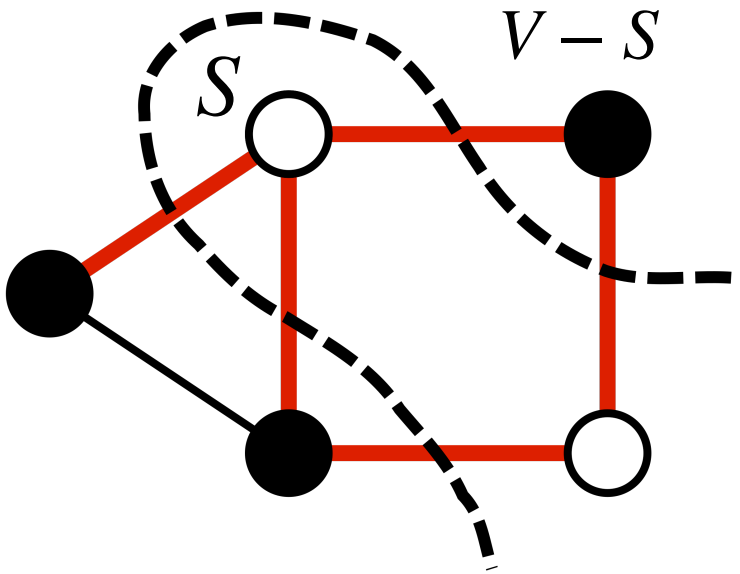
Probabilistic Method - Max Cut

Objective: bipartition graph such that cut is maximized

$$\max_{S \subset V} cut(S, V - S)$$



Probabilistic Method - Max Cut



Prove Max-Cut(G) $\geq \frac{|E|}{2}$

1. Define a probability distribution: sample every node (into S)

   with probability $1/2$. $P(u \in S) = \frac{1}{2}$

$$\begin{aligned}
\mathbb{E}[\text{cut}(S, V-S)] &= \sum_{\{u,v\}\in E} P(\{u,v\} \text{is a cut edge}) \\
&= \sum_{\{u,v\}\in E} P[u \in S, v \notin S] + P[v \in S, u \notin S] \\
&= \sum_{\{u,v\}\in E} \frac{1}{4} + \frac{1}{4} \\
&= \frac{|E|}{2}
\end{aligned}$$

2. We can see that $P(\text{cut}(S, V-S) \geq \frac{|E|}{2}) > 0$

Method of Conditional Expectation

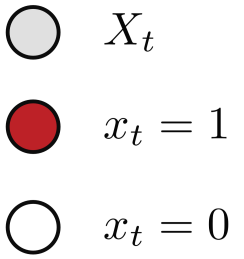
Recap

- Each vertex is placed randomly in S or $V - S$. $\mathbb{E}[\text{cut}(S, V - S)] = \frac{|E|}{2}$
- Deterministically choose where to place each vertex to guarantee a cut of size $\geq \frac{|E|}{2}$

Derandomization via Conditional Expectation

- At each step, fix the value of one random choice.
- Choose it so that the conditional expectation of the total cut size does not decrease.
- After all choices are fixed, the total cut size is still ≥ the original expected value.

Method of Conditional Expectation



Algorithm

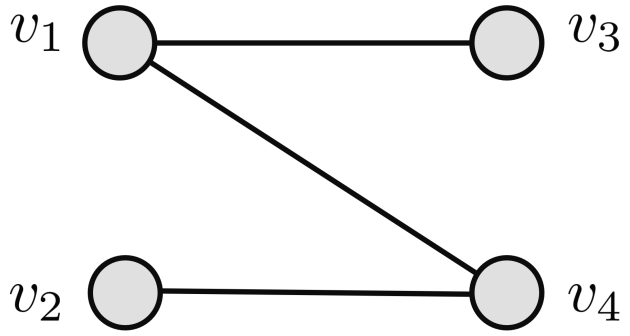
For $t = 1, \ldots, n$ **do**

  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

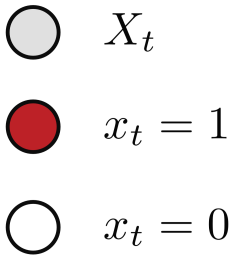
    $x_t \leftarrow 1$

  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$



Method of Conditional Expectation



Algorithm

For $t = 1, \ldots, n$ **do**

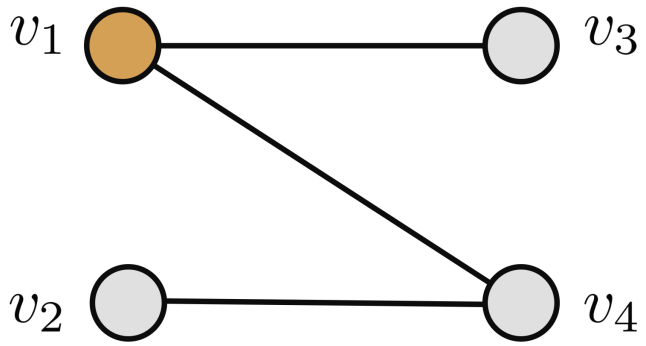
  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

    $x_t \leftarrow 1$

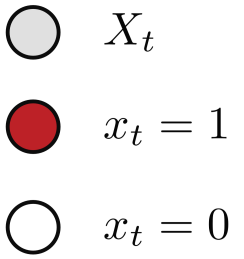
  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$

$$\mathbb{E}[\text{cut} \mid x_1 = 1] > \mathbb{E}[\text{cut}]$$



Method of Conditional Expectation



Algorithm

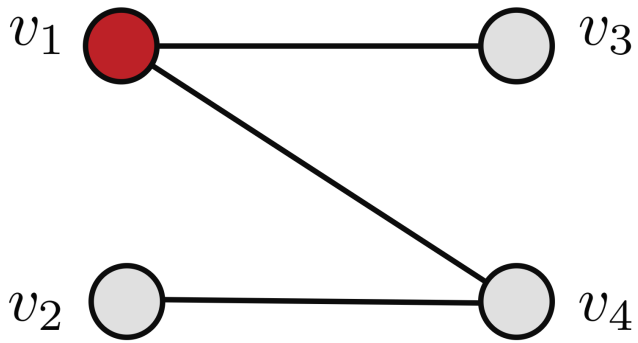
For $t = 1, \ldots, n$ **do**

  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

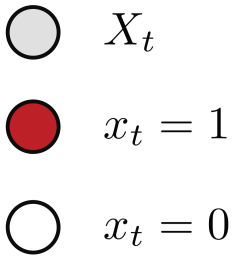
    $x_t \leftarrow 1$

  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$



Method of Conditional Expectation



Algorithm

For $t = 1, \ldots, n$ **do**

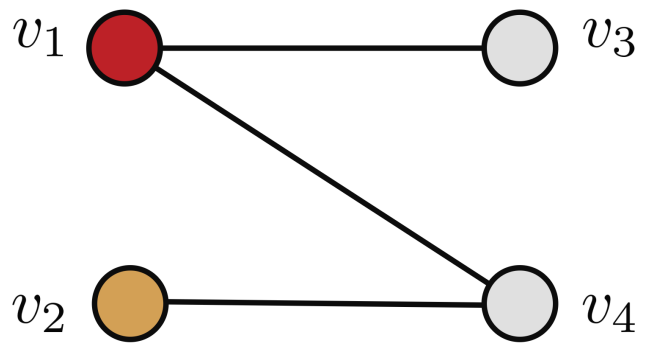
  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

    $x_t \leftarrow 1$

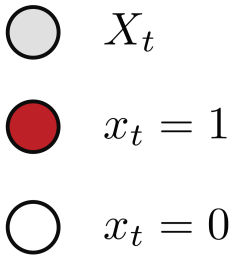
  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$

$\mathbb{E}[cut \mid x_1 = 1, x_2 = 1] > \mathbb{E}[cut \mid x_1 = 1]$



Method of Conditional Expectation



Algorithm

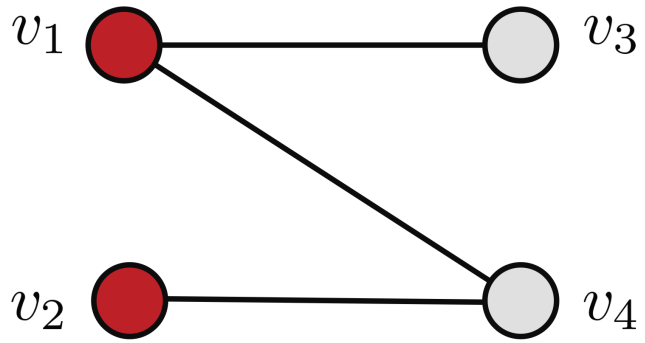
For $t = 1, \ldots, n$ **do**

  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

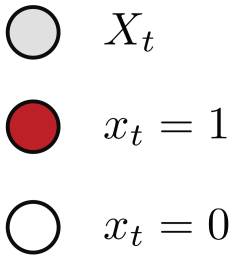
    $x_t \leftarrow 1$

  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$



Method of Conditional Expectation



Algorithm

For $t = 1, \ldots, n$ **do**

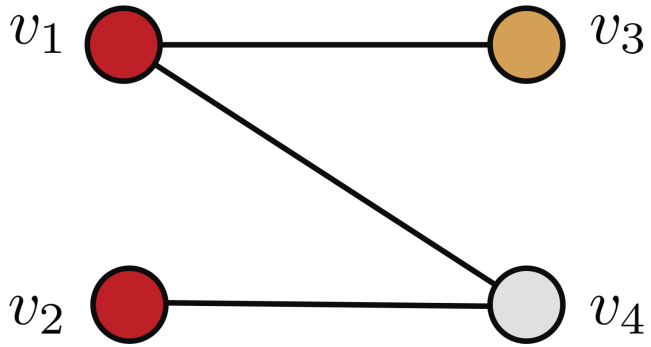
  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

    $x_t \leftarrow 1$

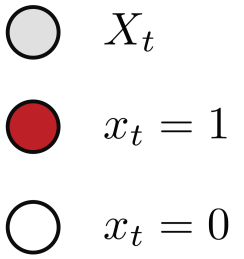
  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$

$\mathbb{E}[cut \mid x_1 = 1, x_2 = 1, x_3 = 1] \leq$
$\mathbb{E}[cut \mid x_1 = 1, x_2 = 2]$



Method of Conditional Expectation



Algorithm

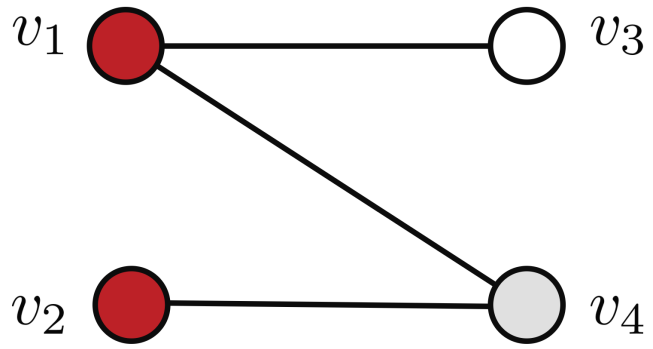
For $t = 1, \ldots, n$ **do**

  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

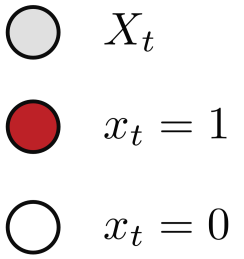
    $x_t \leftarrow 1$

  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$



Method of Conditional Expectation



Algorithm

For $t = 1, \ldots, n$ **do**

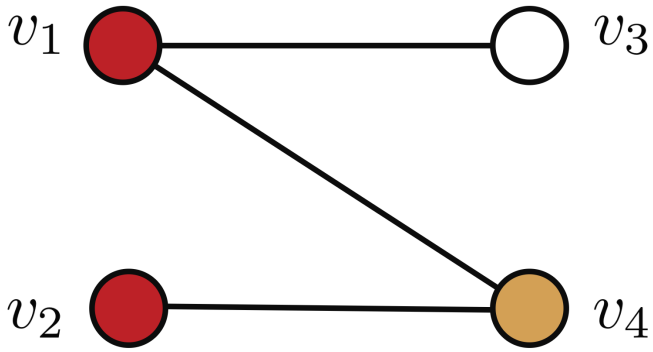
  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

    $x_t \leftarrow 1$

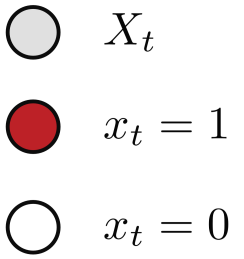
  Else $x_t \leftarrow 0$

Return $\{x_t = 1 \mid v_t \in V\}$

$\mathbb{E}[cut \mid x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1] \leq$
$\mathbb{E}[cut \mid x_1 = 1, x_2 = 2, x_3 = 0]$



Method of Conditional Expectation



Algorithm

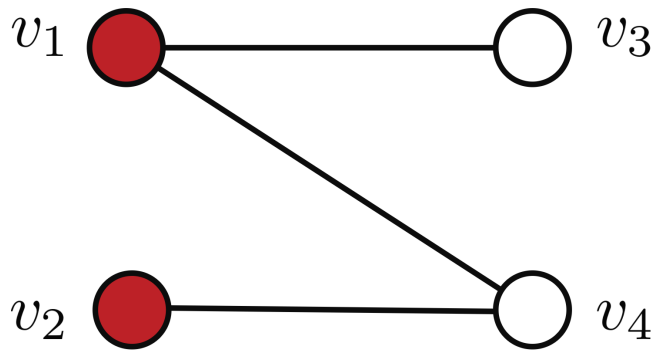
For $t = 1, \ldots, n$ **do**

  If $\mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}, X_t = 1] > \mathbb{E}[\text{cut} \mid x_1, \ldots, x_{t-1}]$

    $x_t \leftarrow 1$

  Else $x_t \leftarrow 0$

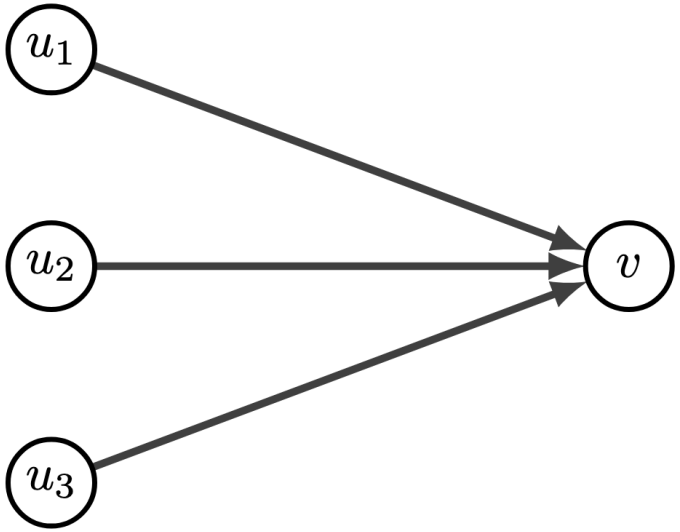
Return $\{x_t = 1 \mid v_t \in V\}$



Graph Neural Networks (GNN)

Input: $G = (V, E)$

Goal: learn node or graph representations



GNN Core Idea

- Each node updates its feature by combining information from its neighbors
- Repeat over multiple layers

Limitation

- All neighbors treated equally
- Sometimes cannot distinguish different graphs

Message Passing

Key Idea

- Each node updates its feature by receiving messages from neighbors

- This is repeated over multiple layers to capture larger neighborhoods

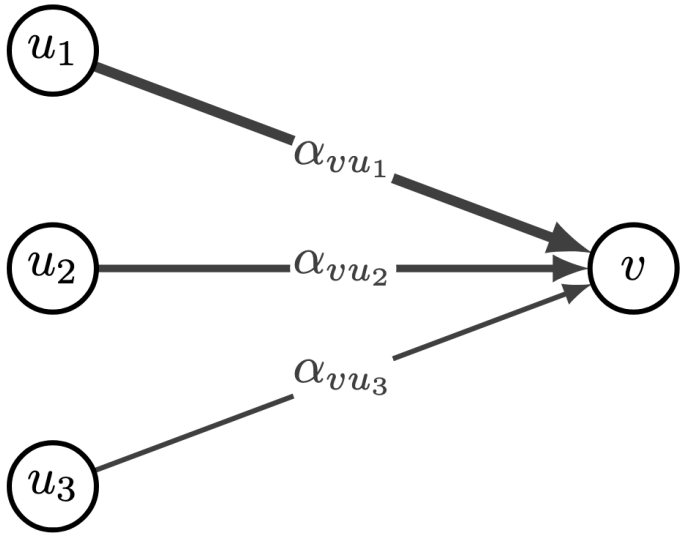
Message passing steps

1. **Message**: Each node collects features from its neighbors.
2. **Aggregation**: Combines the incoming messages (e.g., mean, sum)
3. **Update**: Applies a transformation (like a neural network) to update its own feature

Graph Attention Networks (GAT)

Input: $G = (V, E)$

Goal: learn node or graph representations



GAT Core Idea

- Learn to weigh neighbors differently
- Important neighbors contribute more

Key Features

- Focuses on relevant parts of the graph
- Adapts to node importance dynamically

Erdős goes Neural

Erdos goes Neural

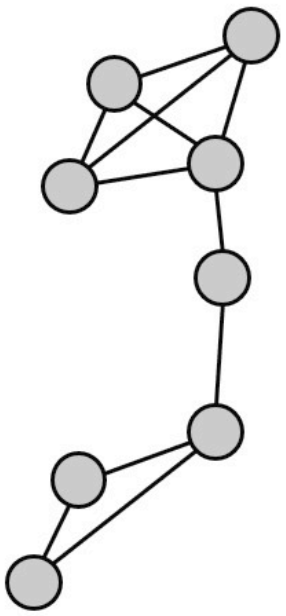
To solve the problem $\min_{S \subseteq V} \quad f(S; G) \quad$ subject to $\quad S \in \Omega$

1. Probability function learned by a GNN

2. $P(\text{S has property}) > 0$ is optimized trough the derived loss function

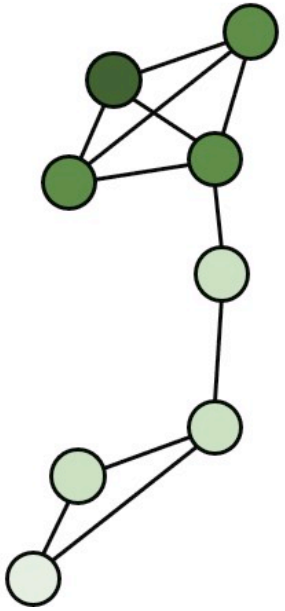
Pipeline

Construct a GNN that outputs a probability p_i on each node v_i.



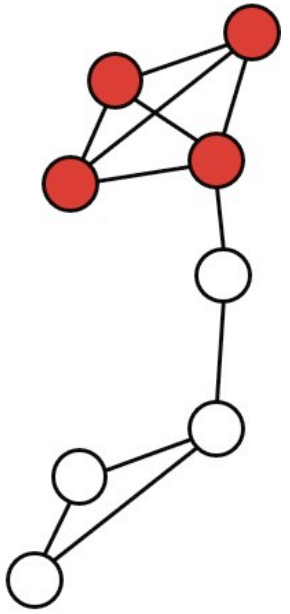
Training

Minimizing the *probabilistic penalty loss* ensures the learned distribution contains a small cost and feasible set with sufficient probability.



Decoding

Recover the *discrete* solution by sampling or by the method of conditional expectation.



Erdos goes neural: Loss function

$$\mathscr{L}(D, G) \quad = \quad E[f(S; G)] \quad + P(S \notin \Omega)\beta$$

Loss distribution of the GNN over the graph's nodes

Probability of constraint violation

Penalty coefficient

Theorem (informal): let *f* be *non-negative. There exists a set* $S^* \sim D$ *that satisfies:*

$$f(S^*; G) < \mathscr{L}(D; G), \quad S^* \in \Omega,$$

with positive probability.

Experiments

Experimental Setup

Problem: Maximum clique

Baseline: neural networks (Toenshoff 2019), classic heuristics, and solvers (CBC, Gurobi 9.0)

Datasets:
- Real world: TWITTER, IMDB, COLLAB
- Hard instances for maximum clique

Architecture
- GIN and GAT layers
- Residual connection and batch norm
- One hot encoding of random node as input
- Train with Adam and early stopping

Results

Test set approximation ratios for all methods on real-world datasets

	IMDB	COLLAB	TWITTER
Erdős' GNN (fast)	1.000 (0.08 s/g)	0.982 ± 0.063 (0.10 s/g)	**0.924 ± 0.133 (0.17 s/g)**
Erdős' GNN (accurate)	1.000 (0.10 s/g)	0.990 ± 0.042 (0.15 s/g)	0.942 ± 0.111 (0.42 s/g)
RUN-CSP (fast)	0.823 ± 0.191 (0.11 s/g)	0.912 ± 0.188 (0.14 s/g)	0.909 ± 0.145 (0.21 s/g)
RUN-CSP (accurate)	0.957 ± 0.089 (0.12 s/g)	0.987 ± 0.074 (0.19 s/g)	0.987 ± 0.063 (0.39 s/g)
Bomze GNN	0.996 ± 0.016 (0.02 s/g)	~~0.984 ± 0.053 (0.03 s/g)~~	~~0.785 ± 0.163 (0.07 s/g)~~
MS GNN	0.995 ± 0.068 (0.03 s/g)	~~0.938 ± 0.171 (0.03 s/g)~~	~~0.805 ± 0.108 (0.07 s/g)~~
NX MIS approx.	0.950 ± 0.071 (0.01 s/g)	0.946 ± 0.078 (1.22 s/g)	0.849 ± 0.097 (0.44 s/g)
Greedy MIS Heur.	0.878 ± 0.174 (1e-3 s/g)	0.771 ± 0.291 (0.04 s/g)	0.500 ± 0.258 (0.05 s/g)
Toenshoff-Greedy	0.987 ± 0.050 (1e-3 s/g)	0.969 ± 0.087 (0.06 s/g)	**0.917 ± 0.126 (0.08 s/g)**
CBC (1s)	0.985 ± 0.121 (0.03 s/g)	0.658 ± 0.474 (0.49 s/g)	0.107 ± 0.309 (1.48 s/g)
CBC (5s)	1.000 (0.03 s/g)	0.841 ± 0.365 (1.11 s/g)	0.198 ± 0.399 (4.77 s/g)
Gurobi 9.0 (0.1s)	**1.000 (1e-3 s/g)**	0.982 ± 0.101 (0.05 s/g)	0.803 ± 0.258 (0.21 s/g)
Gurobi 9.0 (0.5s)	1.000 (1e-3 s/g)	0.997 ± 0.035 (0.06 s/g)	0.996 ± 0.019 (0.34 s/g)
Gurobi 9.0 (1s)	1.000 (1e-3 s/g)	0.999 ± 0.015 (0.06 s/g)	**1.000 (0.34 s/g)**
Gurobi 9.0 (5s)	1.000 (1e-3 s/g)	**1.000 (0.06 s/g)**	1.000 (0.35 s/g)

Results

Approximation ratios for best methods on hard instances

	Training set	Test set	Large Instances
Erdős' GNN (fast)	0.899 ± 0.064 (0.27 s/g)	0.788 ± 0.065 (0.23 s/g)	0.708 ± 0.027 (1.58 s/g)
Erdős' GNN (accurate)	0.915 ± 0.060 (0.53 s/g)	0.799 ± 0.067 (0.46 s/g)	0.735 ± 0.021 (6.68 s/g)
RUN-CSP (fast)	0.833 ± 0.079 (0.27 s/g)	0.738 ± 0.067 (0.23 s/g)	0.771 ± 0.032 (1.84 s/g)
RUN-CSP (accurate)	0.892 ± 0.064 (0.51 s/g)	0.789 ± 0.053 (0.47 s/g)	0.804 ± 0.024 (5.46 s/g)
Toenshoff-Greedy	**0.924 ± 0.060 (0.02 s/g)**	0.816 ± 0.064 (0.02 s/g)	**0.829 ± 0.027 (0.35 s/g)**
Gurobi 9.0 (0.1s)	0.889 ± 0.121 (0.18 s/g)	**0.795 ± 0.118 (0.16 s/g)**	0.697 ± 0.033 (1.17 s/g)
Gurobi 9.0 (0.5s)	**0.962 ± 0.076 (0.34 s/g)**	**0.855 ± 0.083 (0.31 s/g)**	0.697 ± 0.033 (1.54 s/g)
Gurobi 9.0 (1.0s)	**0.980 ± 0.054 (0.45 s/g)**	**0.872 ± 0.070 (0.40 s/g)**	0.705 ± 0.039 (2.05 s/g)
Gurobi 9.0 (5.0s)	**0.998 ± 0.010 (0.76 s/g)**	**0.884 ± 0.062 (0.68 s/g)**	0.790 ± 0.285 (6.01 s/g)
Gurobi 9.0 (20.0s)	**0.999 ± 0.003 (1.04 s/g)**	**0.885 ± 0.063 (0.96 s/g)**	0.807 ± 0.134 (21.24 s/g)

Conclusions

Main Features

- Merges probabilistic method with deep learning
- Principled way to ensure feasibility of solutions for various CO problems
- Competitive performance

Conclusions

Limitations

- Theoretical guarantees — No worst-case approximation bounds
- Lacks strong generalization
- MIP Solvers like Gurobi are still SoA and greedy algorithms can be extremely efficient in practice
- Can be hard to obtain analytically a differentiable expression for certain constraints (e.g., imposing path or tree structure on solutions).

Follow-up

Erdos goes neural laid the groundwork for unsupervised neural approaches to combinatorial optimization

- Meta-EGN (ICLR 2023) Learns to generalize across tasks by optimizing initial weights — great generalization but higher training cost
- Objective Relaxation (NeurIPS 2022) Relax objectives to ensure quality of relaxed solution — Theoretically grounded
- DIFUSCO (NeurIPS 2023) leverages graph-based diffusion models to iteratively refine solutions

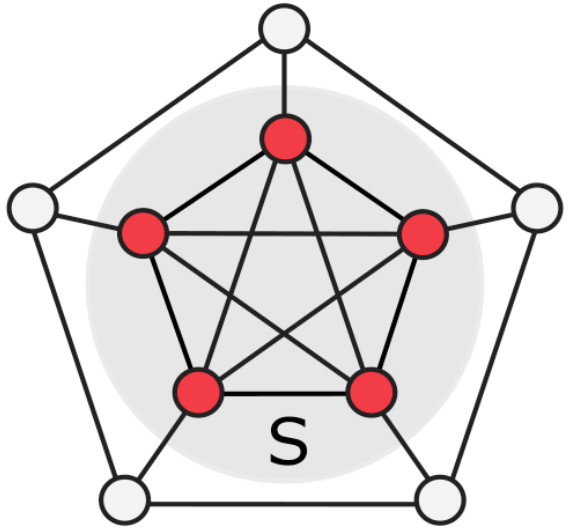
Thank You

Erdos Goes Neural: architecture

- 6 GIN layers followed by a multi-head GAT layer
- Residual connection and batch normalization at every layer
- Graph size normalization, for optimization stability
- The output of GNN to a 2 layers MLP, giving as output one value per node
- Min-Max normalization to rescale the output values in the interval [0,1]

Case study: Maximum clique

$$G = (V, E) \quad |V| = N, \quad |E| = M$$



Objective

$$\min_{S \subseteq V} \quad \gamma - |S| \quad \text{subject to } S \in \Omega_{\text{clique}}$$

Probabilistic penalty Loss

$$\mathscr{L}_{clique}(D, G) = \gamma + E[|S|] + \beta E[|V - S|]$$
$$= \gamma - (\beta + 1) \sum_{(v_i, v_j) \in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j$$

With probability at least t, set $S^* \sim D$ satisfies

$$\gamma - \frac{\mathscr{L}(D, G)}{1 - t} \leq |S^*| \quad S^* \in \Omega_{\text{clique}}$$

Toenshoff 2019

Maximum Clique: Step 1

$G = (V, E) \quad |V| = N, \quad |E| = M$

1. Use a Bernoulli random variable for each node

$$x_i = \begin{cases} 1, & \text{with probability } p_i \\ 0, & \text{with probability } 1 - p_i \end{cases}$$

2. Write the cost function for the problem

$$\min_{S \subseteq V} \quad \gamma - |S| \quad \text{subject to } S \text{ is a clique}$$

3. Build a GNN
    1. Take in input some node features
    2. Output Nx1 vector of probabilities

Maximum Clique: Step 2

Derivation of the loss function

$$l(p_1, \ldots, p_n, G) = E[f(S; G)] + P(S \notin \Omega)\beta$$

$$l_{clique}(p_1, \ldots, p_n, G) = \gamma + E[\,|S|\,] + \beta E[\,|V - S|\,]$$

$$= \gamma - (\beta + 1) \sum_{(v_i, v_j) \in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j$$

Maximum Clique: Step 3

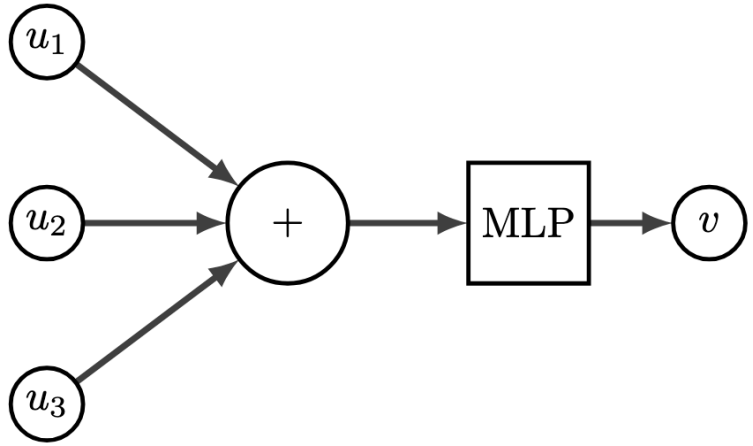
1. Train the network

2. Retrieve the set S from the probabilities of the network using the method of conditional expectation:
   Sort the nodes according to their probabilities. Starting from the high probability nodes, for each node v_i do:
   1. Evaluate the loss for $p_i = 1$ and for $p_i = 0$
   2. Set p_i to either 1 or 0 depending on what achieved the better loss
   3. Move on to the next node and repeat from step 1

Graph Isomorphism Networks (GIN)

Input: $G = (V, E)$

Goal: learn node or graph representations



GIN Core Idea

- Stronger aggregation: sum all neighbor features
- Apply a neural network (MLP) after aggregation

Key Features

- Best at capturing subtle graph structures
- MLP can learn complex local patterns