Erdos Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs

Robert Wang

March 28, 2024

Combinatorial Optimization Problems

- Optimizing functions over discrete sets instead of continuous spaces like $\mathbb{R}^n$
- **Example:** shortest path, TSP, max-cut, min-cut, Knapsack etc.
- Many are NP-hard, likley not solvable in polynomial time
- Objective us not differentiable, cannot apply methods like gradient descent

Why is the problem Important?

- Many fundamental problems in computer science are combinatorial in nature
- Have applications in clustering, routing, game theory etc.
- In theory, we try to understand these problems by seeing what we can provably solve using algorithms
- The best practical solvers don't necessarily have provable guarantees but can do well on real instances of the problem

Existing Approaches

- Approximation algorithms: have theoretical guarantees but often hard to implement and may not perform well on real instances

- State of the Arc Solvers using combinatorial enumeration methods like branch and bound. Can provably solve the problem exactly but not guaranteed to run in polynomial time. Ex. Gurobi ILP solver

Existing Approaches

- Combinatorial Approximation algorithms: can be easy to implement, especially if it's a greedy algorithm
- Branch and Bound/enumeration algorithms: Guaranteed to get optimal solution but may not run in polynomial time
- Convex Relaxation: formulate the problem as a convex program that can be solved efficiently but only integer solutions correspond to feasible solutions.
- **Nonconvex Relaxation**: formulate problem as a general continuous minimization problem whose optimum is equalled to the integer optimum but the objective may not be convex so not solvable in polynomial time

ML-based Approaches

- People also try to use learning-based approaches to tackle these problems
- **Supervised-Approach:** train networks for solving CO problems using instances with known solution
  - **Issue:** How to find instances with known solutions if they are NP-hard?
- **Unsupervised-approach:** (this work) define a suitable objective function that is differentiable but also captures the optimal combinatorial solution. Train a network to minimize this objective

This Work

- Given a graph $G = (V, E)$ Use a GNN to solve graph problems of the form

$$\min_{S \subseteq G} f(S, G)$$
$$\text{s.t.} S \in \Omega$$

- Use GNN to train a probability distribution over the graph $(p_i)_{i \in V}$ where $p_i = \Pr[i \in S]$. Define Loss as

$$L(p, G) = \min_{p} E_{S \sim p}[f(S, G)] + \beta \Pr_{S \sim p}[S \notin \Omega]$$

- Reduce combinatorial optimization problem to a non-convex optimization problem of minimizing the loss

Closer Look at the Learning Objective

- For $i \in V$, let $(X_i)_{i \in V}$ be independent Bernouli random variables such that

$$X_i = \begin{cases} 1 & i \in S \\ 0 & i \notin S \end{cases}$$

- Because X_i's are independent, the distribution is fully determined by $p_1, \ldots p_n$, and we would like to learn these probabilities with our network

- To ensure the resulting set satisfies constraints, we use the hyperparameter β

Conditions for Recovery

Theorem

Suppose f is non-negative, $\beta > \max_{S \subseteq V} f(S, G)$, and we have a probability distribution $p_1, ..p_n$ such that

$$E_{S \sim p}[f(S, G)] + \beta \Pr[S \notin \Omega] = \ell < \beta$$

Then there exists a set $S \subseteq G$ such that

$$f(S, G) \leq \ell \quad \textit{and} \quad S \in \Omega$$

Proof

- Let $f_\beta(S, G) = f(S, G) + \beta \mathbb{1}\{S \notin \Omega\}$
- Objective value

$$\begin{aligned} \ell := E[f(S, G)] + \beta \Pr[S \notin \Omega] &= E[f(S, G) + \beta \mathbb{1}\{S \notin \Omega\}] \\ &= E[f_\beta(S, G)] \end{aligned}$$

- There must exist a set S such that $f_\beta(S, G) \leq E[f_\beta(S, G)] < \beta$
- Since $0 < f(S, G) < \beta$, $f_\beta(S, G) < \beta$ iff $S \in \Omega$
- Thus, we have $f_\beta(S, G) = f(S, G) \leq \ell$

Conditional Expectation

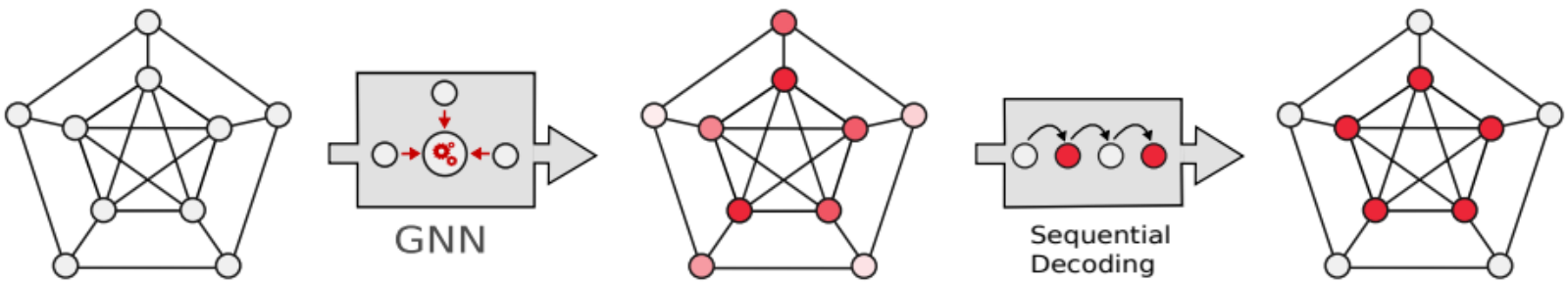
- How to explicitly find a set S satisfying $f_\beta(S, G) \leq \ell$?
- **Condition on node 1:**

  $$E[f_\beta(S, G)] = p_1 E[f_\beta(S, G)|1 \in S] + (1 - p_1) E[f_\beta(S, G)|1 \notin S]$$

- Since variables are independent, we can calculate both conditional expectation terms
- One of the terms must be at most $E[f_\beta(S, G)]$
- If the $E[f_\beta(S, G)|1 \in S]$ is smaller, fix $1 \in S$ and recurse on $E[f_\beta(S, G)|1 \in S]$
- otherwise fix $1 \notin S$ and recurse on $E[f_\beta(S, G)|1 \notin S]$

Output

- input $\rightarrow$ probability distribution $p \rightarrow$ set S satisfying $f_\beta(S) < E_{S\sim p}[f_\beta(S)] \Rightarrow S \in \Omega$ and $f(S, G) \leq E_{S\sim p}[f_\beta(S)]$ if $E_{S\sim p}[f_\beta(S)] < \beta$



Example: Max-clique

- Given a graph $G = (V, E)$, a clique is a set S such that $\forall i, j \in S$, $(i, j) \in E$
- **Goal:** maximize number of edges in S s.t. S is a clique
- The objective function is at most $\binom{n}{2}$ so we can minimize the objective function $\binom{n}{2} - \#$ edges in S s.t. S is a clique

Loss Function for Max-clique

- **Objective Function:**

$$\binom{n}{2} - E[\sum_{i,j} \mathbb{1}\{i,j \in S\}] = \binom{n}{2} - \sum_{(i,j)\in E} p_i p_j$$

- Constraint violated for some pair i, j, we have $i, j \in S$ but $(i,j) \notin E$. The number of volating pairs is $\sum_{i,j\in S} 1 - \mathbb{1}\{(i,j) \in E\}$

- Thus, we have

$$\begin{aligned}
\Pr[S \notin \Omega_{clique}] &\leq E[\#\text{ of violating pairs}] \\
&= \sum_{i\neq j} p_i p_j (1 - \mathbb{1}\{(i,j) \in E\}) \\
&= \sum_{i\neq j} p_i p_j - \sum_{(i,j)\in E} p_i p_j
\end{aligned}$$

Loss Function for Max-clique

- **Loss Function:**

$$L(p, G) = \binom{n}{2} - \sum_{(i,j)\in E} p_i p_j + \beta\left(\sum_{i\neq j} p_i p_j - \sum_{(i,j)\in E} p_i p_j\right)$$

$$= \binom{n}{2} - (1+\beta) \sum_{(i,j)\in E} p_i p_j + \beta \sum_{i\neq j} p_i p_j$$

- Formulation as a Quadratic Program:

$$\min_{p\in[0,1]^n} p^\top (\beta A(K_n) - (1+\beta)A(G))p$$

Where $A(G)$ is the adjacency matrix of G and $A(K_n)$ is the adjacency matrix of the complete graph

Example: Conductance/normalized cut

- Given a weighted graph $G = (V, E)$ we say the volume of a set S is $Vol(S) = \sum_{i \in S} d_i$ where d_i is the degree of i
- We say $\delta(S) = \{(i, j) \in E : i \in S, j \notin S\}$ is the boundary set of S
- **Normalized cut:** $\min_{S \subseteq V} |\delta(S)|$ s.t. $Vol(S) \in [\ell, u]$
- **Conductance:** $\min_{S \subseteq V} \frac{|\delta(S)|}{\min(Vol(S), Vol(V \setminus S)}$

Loss Function for Normalized Cut

- How to satisfy constraint $S \in [\ell, u]$ with high probability?
- GNN trains parameters $p_1^0, \ldots p_n^0$. We project them by applying the transform $p_i = \text{clip}(cp_i^0, 0, 1)$ for some scalar c such that $\sum_i d_i p_i = \frac{\ell+u}{2}$
- Loss fuction:

$$\begin{aligned} E[\delta(S)] &= \sum_{(i,j)\in E} p_i(1-p_j) + p_j(1-p_i) \\ &= \sum_{(i,j)\in E} p_i + p_j - 2p_i p_j \\ &= \sum_i d_i p_i - 2 \sum_{(i,j)\in E} p_i p_j \end{aligned}$$

Loss Function for Normalized Cut

- Note we can also formulate this as a Quadratic Program

$$\min_{p \in [0,1]^n} -p^\top A(G)p$$
$$\text{s.t.} \sum_i d_i p_i = \frac{\ell + u}{2}$$

- Sample S according to p_i
- Since distribution is independent, $\sum_{i \in S} d_i$ will be concentrated around its expectation $\sum_i d_i p_i = \frac{\ell+u}{2}$
- With high probability, $Vol(S) \in [\ell, u]$

Reduction of Conductance to Normalized Cut

- Reduce to Normalized cut as follows:
- Search over possible volumes intervals, for example iterate over $(\ell_i, u_i) = (i\epsilon, (i+1)\epsilon)$ for some small ϵ
- For each volume interval, find the smallest normalized S_i cut and return $\min_i \delta(S_i)/Vol(S_i)$

Nework Architecture

- **Normalized Cut:** 6-Layers of Graph Isomorphism Network with skip connections and batch normalization followed by GAT.
- **Clique:** Did not use GAT layer. Added some batch and graph size normalizations
- Re-scale output values to lie in $[0, 1]$ at the end

Local Algorithm

- Wanted the cut and clique finding algorithms to be local, i.e. find a sparse cut or clique around an initial seed vertex
- Local algorithm can be faster and useful in settings where we have some idea of where our cut or clique should be
- Input seed vector is a 1-hot encoding of starting vertex
- After k convolution, only k hop neighbors of see allowed to have non-zero values

Results for Max Clique

		IMDB	COLLAB	TWITTER
other NN	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)
other obj	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)*
greedy	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)**
ILP	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)

Constraint Violations

	IMDB	COLLAB	TWITTER	RB (all datasets)
Erdős' GNN (fast)	**0%**	**0%**	**0%**	**0%**
Erdős' GNN (accurate)	**0%**	**0%**	**0%**	**0%**
Bomze GNN	**0%**	11.8%	78.1%	–
MS GNN	1%	15.1%	84.7%	–

Table 5: Percentage of test instances where the clique constraint was violated.

Results for Max Clique

	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)

Results for Local Conductance

- Consider all possible starting nodes as seed vertex
- The L1 and L2 objectives are standard relaxations for $\delta(S)$

$$L1(x) = \sum_{(i,j)\in E} |x_i - x_j| \quad L2(x) = \sum_{(i,j)\in E} (x_i - x_j)^2$$

- Didn't specify how they are normalizing it. The normalizations for these objectives are different.

		SF-295	FACEBOOK	TWITTER
	Erdős' GNN	**0.124 ± 0.001 (0.22 s/g)**	**0.156 ± 0.026 (289.28 s/g)**	**0.292 ± 0.009 (6.17 s/g)**
other objs	L1 GNN	0.188 ± 0.045 (0.02 s/g)	0.571 ± 0.191 (13.83 s/g)	**0.318 ± 0.077 (0.53 s/g)**
other objs	L2 GNN	**0.149 ± 0.038 (0.02 s/g)**	**0.305 ± 0.082 (13.83 s/g)**	0.388 ± 0.074 (0.53 s/g)
Classical algs	Pagerank-Nibble	0.375 ± 0.001 (1.48 s/g)	N/A	0.603 ± 0.005 (20.62 s/g)
Classical algs	CRD	0.364 ± 0.001 (0.03 s/g)	0.301 ± 0.097 (596.46 s/g)	0.502 ± 0.020 (20.35 s/g)
Classical algs	MQI	0.659 ± 0.000 (0.03 s/g)	0.935 ± 0.024 (408.52 s/g)	0.887 ± 0.007 (0.71 s/g)
Classical algs	Simple-Local	0.650 ± 0.024 (0.05 s/g)	0.955 ± 0.019 (404.67 s/g)	0.895 ± 0.008 (0.84 s/g)
	Gurobi (10s)	**0.105 ± 0.000 (0.16 s/g)**	0.961 ± 0.010 (1787.79 s/g)	0.535 ± 0.006 (52.98 s/g)

Questions

- They reduce the combinatorial problems to solving Quadratic Programs, but why is GNN the best why to solve them? Should maybe benchmark against standard/state of the arc QP solvers
- For Conductance, why only restrict to the local regime and not to the global regime? Seems like their network should be able to handle both
- Why use GNN to solve the L1 and L2 objective?
- Is their formulation of the problem in Gurobi really the best way to formulate it as an integer program?