Answer set computation of negative two-literal programs based on graph neural networks

Preliminary results

Antonio Ielo Francesco Ricca

Università della Calabria

CILC 2021, 8 September

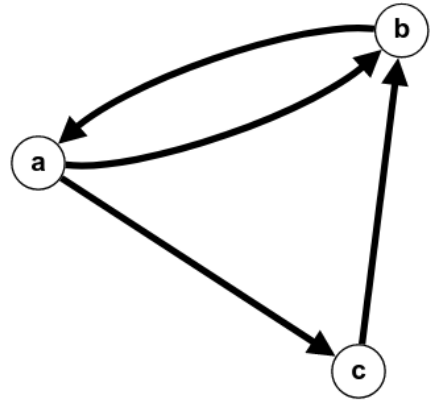
Introduction

- *Neural combinatorial optimization*: deep learning applied to combinatorial optimization
- Examples: (approximate) end-to-end solvers, branching heuristics
- Graph coloring (GNN-COL), SAT (NeuroSAT, NeuroCore), binary CSP (RUNCSP)
- What about Answer Set Programming?

Negative two-literal programs (N2LPs)

- Programs with rules of the form $\alpha \leftarrow \neg\beta$.

```
% example.lp
b :- not a.
c :- not a.
c :- not b.
a :- not b.
```



Properties

- All normal logic programs can be rewritten as equivalent N2LPs.
- Computing answer sets for N2LPs is an NP-complete problem.
- Co-kernels of $G(P)$ are answer sets of P
- Random N2LPs exhibit *easy-hard-easy* pattern (Namasivayam, Truszczyński)

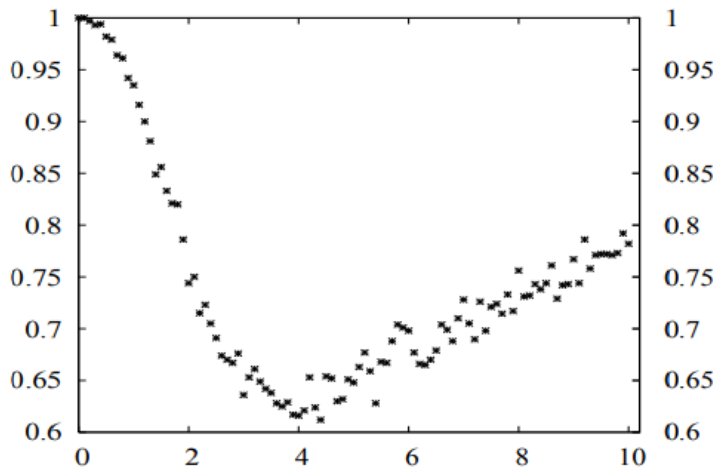


Figure: Probability P has an answer set as function of rule density for N2LPs with 150 atoms

Message passing graph neural networks

Let $G(V, E)$ be a (directed) graph. A (message passing) graph neural network models a function $V \mapsto \mathbb{R}^k$ by performing several *message passing rounds*.

- Neighbouring nodes exchange messages generated by a function $M : \mathbb{R}^k \times \mathbb{R}^k \mapsto \mathbb{R}^k$.
- Each node aggregates, combines incoming messages via a function $C : \{\{\mathbb{R}^k\}\} \mapsto \mathbb{R}^k$
- Each node updates its own embedding based on the incoming messages, via a function $U : \mathbb{R}^k \times \mathbb{R}^k \mapsto \mathbb{R}^k$.
- The network "output" is a function $R : \mathbb{R}^k \mapsto \mathbb{R}^d$.

Message passing graph neural networks

A message passing round consists the following computation (node equations):

$$h(v)^{i+1} = U(h(v)^i, C(\{\{M(h(w)^i, h(v)^i) : (w, v) \in E\}\}))$$

If (M, U, R, C) are differentiable functions (e.g. feedforward neural networks), a sequence of message passing rounds is differentiable and we can train the network with standard deep learning optimization techniques.

RUNCSP (Toenschoff et al, 2020)

RUNCSP is an architecture that approximately solves binary MAX-CSP problems, performing a node classification task. Considering single-predicate CSPs, we can simplify it to a message passing neural network where:

- M is a linear function
- R is a linear function composed with a sigmoid non linearity
- U is a recurrent neural network (LSTM)
- C the mean of incoming messages
- Successfully applied to SAT, MIS, 3COL, MAX-2SAT

RUNCSP (Toenschoff et al, 2020)

- Encode a problem instance as a graph (nodes as variables, edges as constraints)
- Use the GNN to produce a *soft assignment* $\alpha : V \mapsto [0,1]^{|C|}$
- Minimize a loss function $\mathcal{L}_\alpha$ that penalizes violated constraints
- Decode $\alpha(v_1), ..., \alpha(v_n)$ to obtain a "candidate solution" to the problem instance

Tweaking RUNCSP for N2LPs

- Encode a N2LP P with a graph $G(P)$
- Use the GNN to produce a *soft assignment* $\alpha : A \mapsto [0, 1]$, where $\alpha(a)$ represents the probability a belongs to the candidate answer set.
- Compute $S : A \mapsto [0, 1]$, max-probability atom a is supported
- Minimize $\mathcal{L}_{\alpha,S}$ that penalizes violated constraints (and takes into account atom supportedness).
- Decode $\alpha(a_1) \cdot S(a_1), ..., \alpha(v_n) \cdot S(a_n)$ to obtain a candidate answer set for P.

Addressing supportedness of atoms

- It is possible to compute $\mathcal{P}(a \text{ is } \mathrm{supported} | \alpha)$, not effective in practice. We compute instead the maximum probability of a being supported by a single rule, which we denote by $S(a)$.
- Each node communicates $1 - \alpha(v)$ (probability of not being in the answer set), nodes aggregate incoming messages by max.
- We formulate loss function in terms of $\alpha'(v) = \alpha(v) \cdot S(v)$ rather than $\alpha(v)$ alone in order to enforce supportedness of atoms.

$$\mathcal{L}_{\alpha,S} = \frac{1}{n} \sum_{(b \leftarrow \neg a) \in P} - \log 1 - (1 - \alpha'(a))(1 - \alpha'(b))$$

Training and evaluation

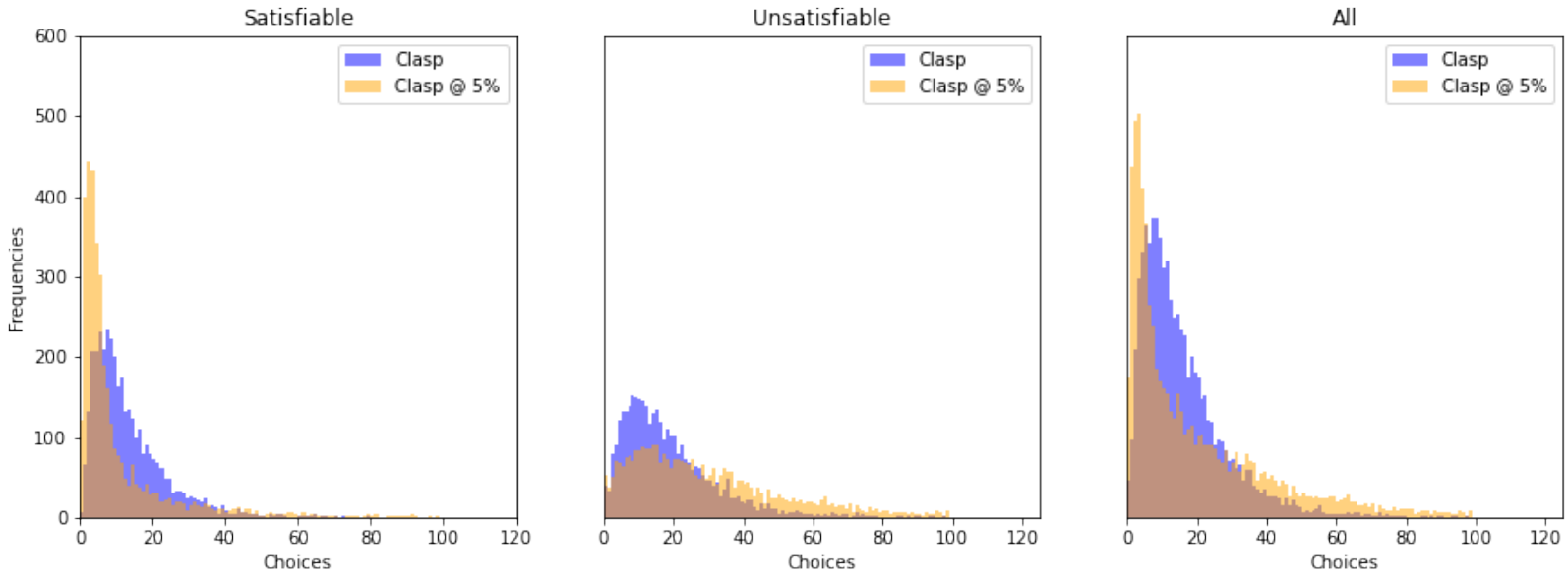
- Trained on random stream of Erdos-Renyi graphs 20-50 nodes, density ranging from 2.0 to 5.0
- Testing on random stream of graphs 150 nodes, density ranging from 2.0 to 9.0 (accuracy, F-score)
- Testing on random stream of graphs 150 nodes, 600 rules (choices distribution, peak of the hard phase)

Results

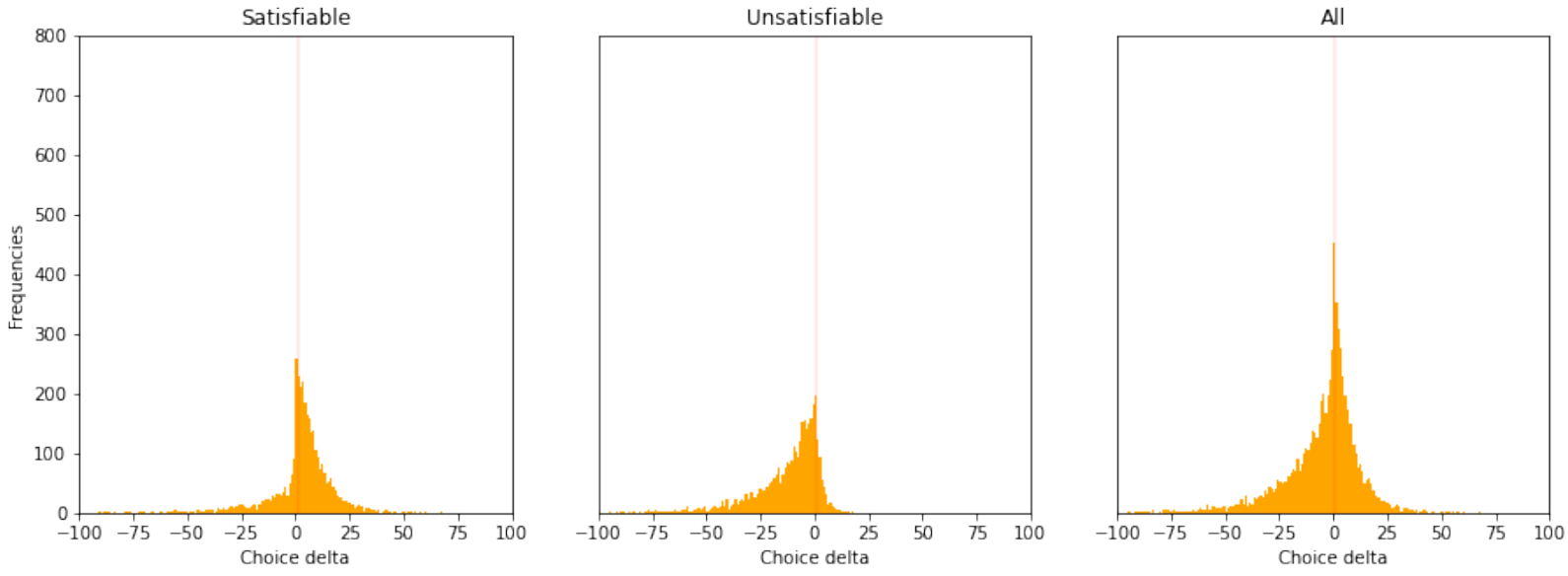
Table 1. Results of the experiment on Erdős-Rényi graphs of 150 nodes.

Avg. Degree	F1 Score		Accuracy	
	Neural	Random	Neural	Random
2.0	0.996	0.372	0.996	0.473
2.5	0.982	0.379	0.978	0.452
3.0	0.974	0.381	0.967	0.432
3.5	0.949	0.386	0.932	0.419
4.0	0.927	0.392	0.899	0.412
4.5	0.906	0.393	0.868	0.401
5.0	0.890	0.396	0.842	0.394
5.5	0.862	0.395	0.811	0.386
6.0	0.837	0.397	0.787	0.379
6.5	0.795	0.396	0.748	0.371
7.0	0.749	0.401	0.709	0.370
7.5	0.667	0.402	0.651	0.365
8.0	0.622	0.404	0.615	0.363
8.5	0.549	0.404	0.564	0.359
9.0	0.498	0.405	0.527	0.356

Results


Satisfiable
Unsatisfiable
All
Clasp
Clasp @ 5%
Frequencies
Choices
0
100
200
300
400
500
600
0
20
40
60
80
100
120

Results


Satisfiable
Unsatisfiable
All
Frequencies
0
100
200
300
400
500
600
700
800
−100 −75 −50 −25 0 25 50 75 100
Choice delta

Conclusion

- ML-wise: better than random (+), some degree of generalization (+), degrades quickly as programs are more dense (-)
- Solver-wise: positively affects coherent programs' choice distribution (+), negatively affects incoherent programs' choice distribution (-), net effect is slightly worse than vanilla (-)

Thanks for your attention!