Seminar: Learning to Generate SAT Formulas

Yu-Zhe Shi

April 6, 2020

Author Profiles

- Jiaxuan You, 3rd year PhD, CS, Stanford;
  `https://cs.stanford.edu/~jiaxuan/`
- Haoze Wu, 2nd year PhD, CS, Stanford;
  `https://anwu1219.github.io/`
- Clark Barrett, Associate Prof, CS, Stanford;
  `http://theory.stanford.edu/~barrett/`
- Raghuram Ramanujan, Assistant Prof, MATH-CS, Davidson;
  `https://www.davidson.edu/people/raghu-ramanujan`
- Jure Leskovec, Associate Prof, CS, Stanford.
  `https://cs.stanford.edu/people/jure/`

Prerequisites: SAT

- SATisfiability problem is the first NP-Complete problem proved.
- A SAT formula ϕ is a composition of Boolean variables x_i connected with logical operators $\vee$, $\wedge$, $\neg$.

$$\exists x_i,\ \phi(x_i) = 1, \Rightarrow \phi \textit{ is satisfiable}. \tag{1}$$

- CNF(Conjunctive Normal Form):
$(x_1 \vee x_2 \vee \dots) \wedge (x_c \vee x_2 \vee \dots)$

Prerequisites: LCG

- LCG(Literal-Clause Graph): Node$\leftarrow$Literal x_i, Clauses $x_i \wedge x_j$. Edge$\leftarrow$ $x_i \Leftrightarrow x_i \vee x_j$. LCGs can be presented in Bipartite graphs where the vertex set can be splitted into a vertex set of literals and a vertex set of clauses.

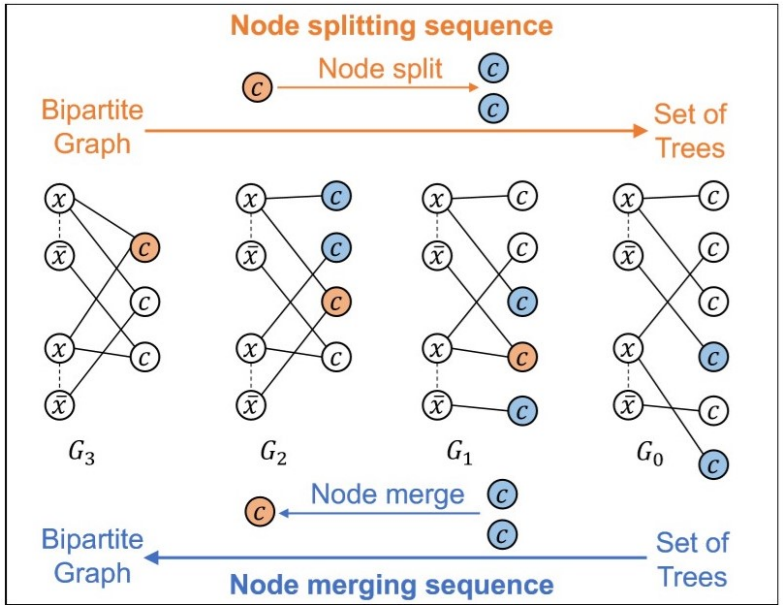
$$\begin{aligned} G = (\mathcal{V}, \mathcal{E}), \Rightarrow \mathcal{V} = \mathcal{V}_2 \cup \mathcal{V}_1 \\ \mathcal{V}_1 = \{l_1, \ldots, l_n\}, \mathcal{V}_2 = \{c_1, \ldots, c_m\} \end{aligned} \tag{2}$$

where there are n literals and m clauses in the LCG.

Prerequisites: Graph Splitting and Merging

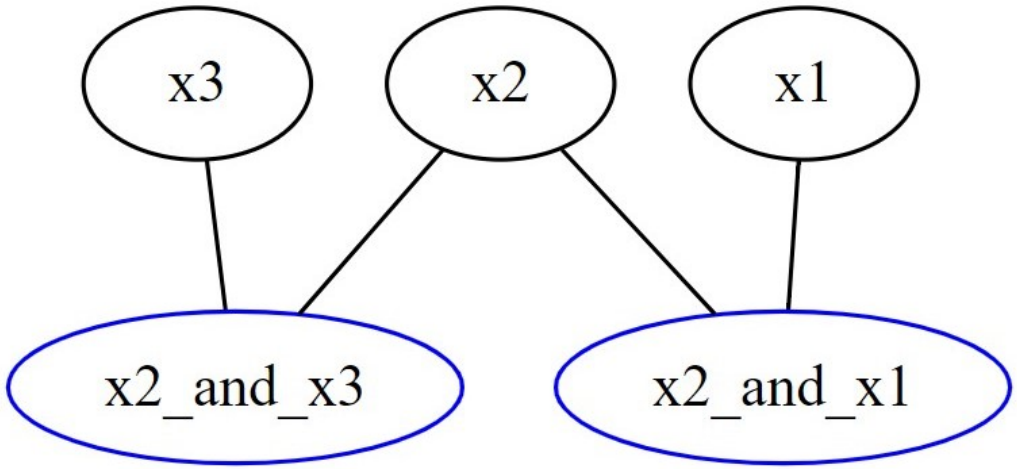
- $$\begin{aligned} NodeSplit(u, G) &\Rightarrow (u, v, G') \\ NodeMerge(u, v, G) &\Rightarrow (u, G') \end{aligned} \tag{3}$$

Hence, the split-merge operation is symmetric.



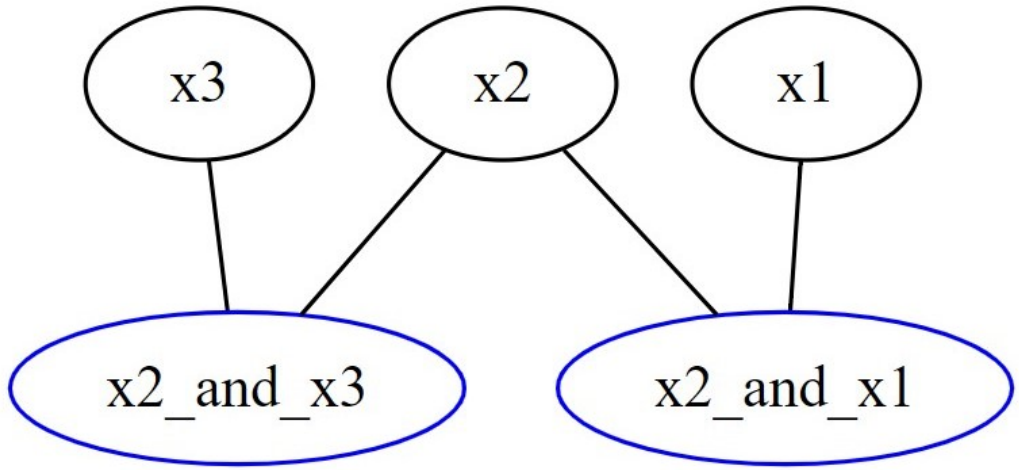
A Toy Example: LCG in Bipartite Representation

$$(x_1 \vee x_2) \wedge (x_2 \vee x_3) \tag{4}$$



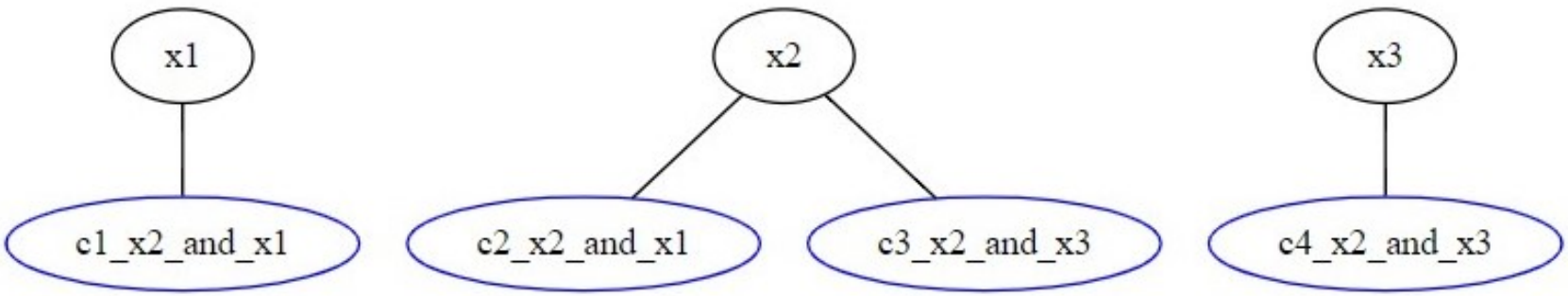
$$\mathcal{V}_1 = x_1, x_2, x_3, \ \mathcal{V}_2 = x_2 \wedge x_3, x_1 \wedge x_2 \tag{5}$$

A Toy Example: Split Iteration 2



Note: $Deg(x_2 \vee x_3) = 2 > 1, Deg(x_2 \vee x_1) = 2 > 1$
Split G_i in next 2 iterations!

A Toy Example: Split Terminated

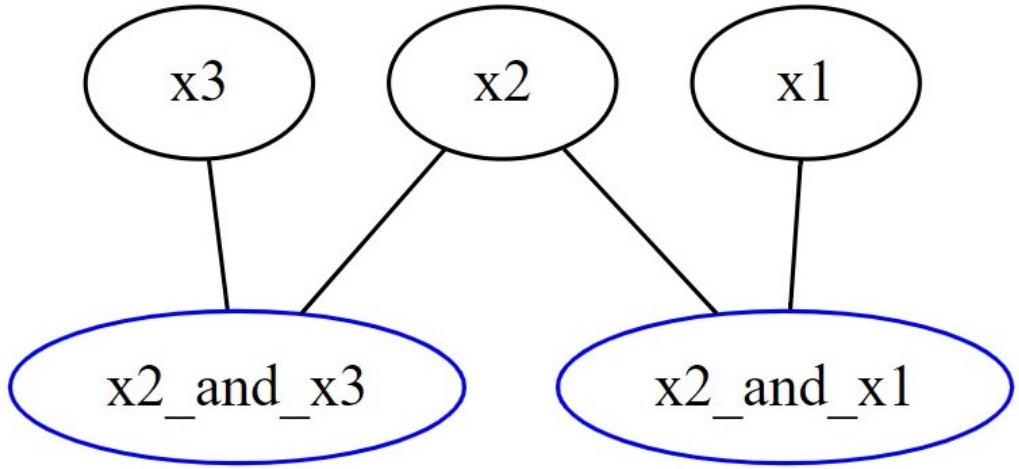


Note: $Deg(x_2 \vee x_3) = 1, Deg(x_2 \vee x_1) = 1$
Terminate splitting and Save G_{i-1} as G_0!

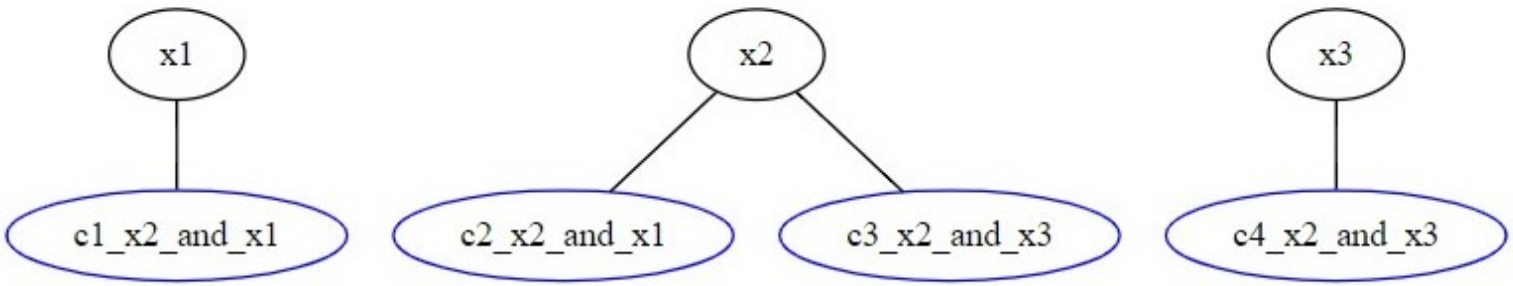
Motivation and Intuition

- We want to generate an LCG "like" G_i, but larger than G_i.
- Can we learn LCG directly from G_i? Nope!
- Maybe... We can learn what to merge from meta LCG.

What to learn?



$$p(G_i|G_{i-1}) = ? \tag{6}$$



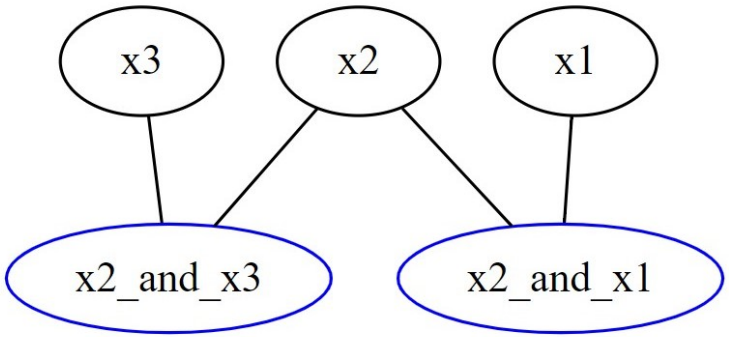
Learn p(G) Iteratively

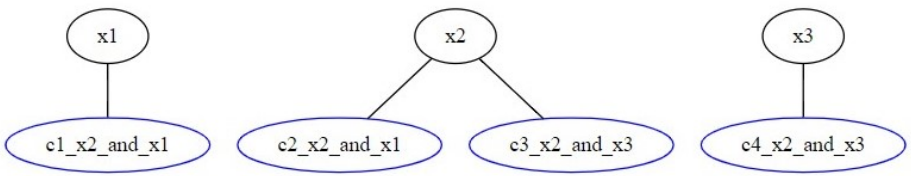
$$p(G) = \prod_{i=1}^{n} p(G_i | G_1, \ldots, G_{i-1}) \tag{7}$$

where G_{i-1} denotes intermediate results.

How to learn?

- What distincts an LCG from others?
- It merges specific nodes!

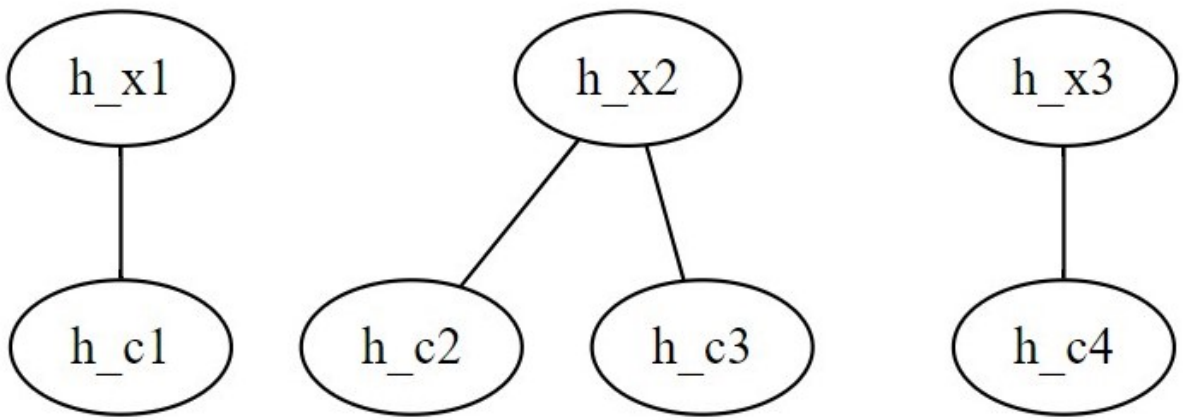




Embedding via GraphSAGE

$$
\begin{aligned}
p(G_i|G_{i-1}) &= p(NodeMerge(u, v, G_{i-1})|G_{i-1}) \\
&= Multinomial(h_u^T h_v / Z | \forall u, v \in \mathcal{V}_2^{G_{i-1}})
\end{aligned} \tag{8}
$$

where Z is a normalization term.

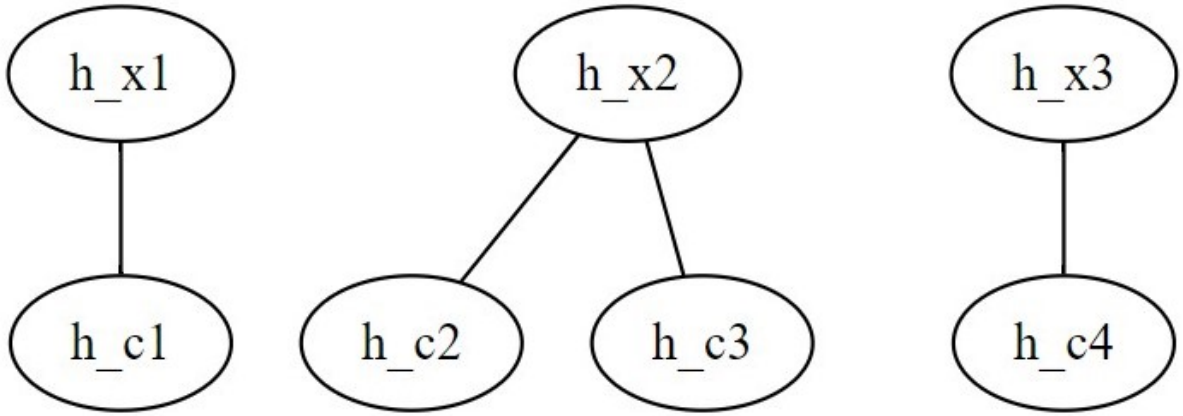


Embedding via GraphSAGE

- 3 node types to be embedded: positive literals, negative literals, clauses.
- embedding of node u at n-layer of GraphSAGE

$$\begin{aligned} n_u^l &= MeanPooling(ReLU(Q^l h_v^l + q^l | v \in Neighbor(u))) \\ h_u^{l+1} &= ReLU(W^l CONCAT(h_u^l, n_u^l)) \end{aligned} \tag{9}$$

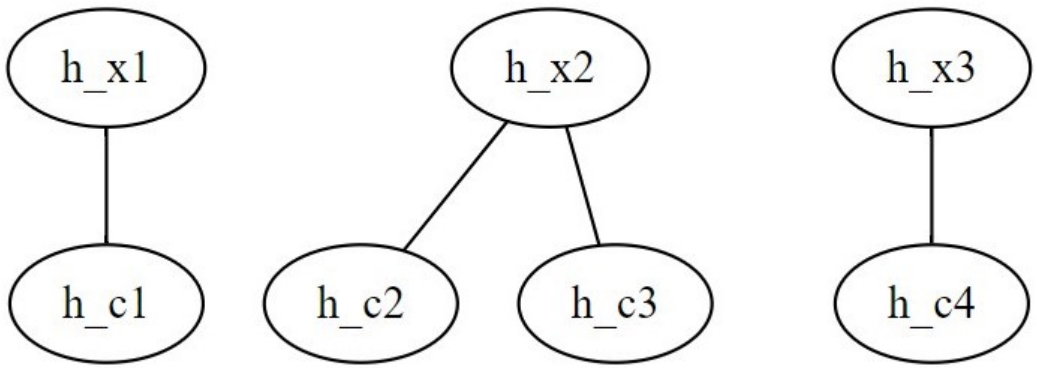
Oops...



- If we enumerate any two clause nodes in a single iteration, here we have $C(4, 2) = 6$ pairs.
- An LCG in practice may have 10^6 nodes or more, $C(10^6, 2)$...
- Computing Z is ... a disaster.

Pair Proposal Strategy

- We randomly select a node pair and decide whether to merge!
- Reduce the multinomial distribution to binary distribution!
- The selected pair shouldn't be empty.



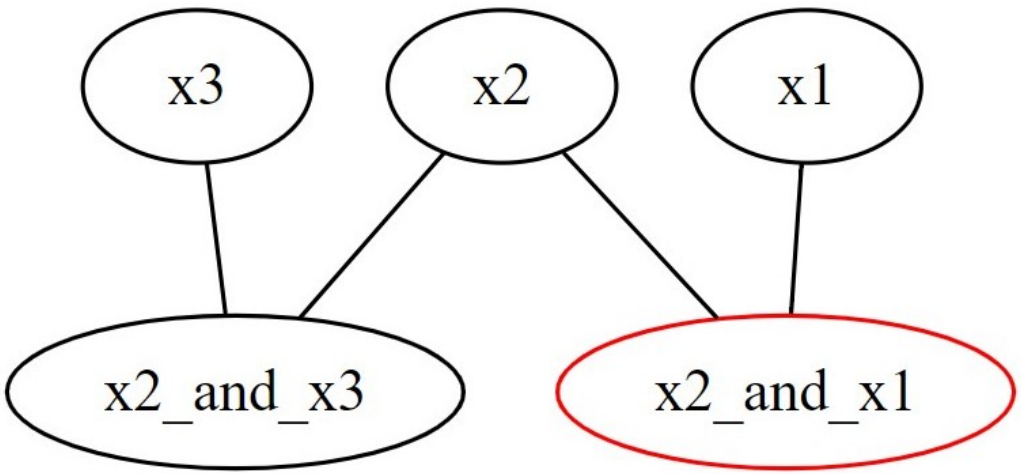
Now we begin to merge!

- $p(G_i|G_{i-1})$ to a joint distribution

$$
\begin{aligned}
p(G_i, u, v|G_{i-1}) &= p(u, v|G_{i-1})p(G_i|G_{i-1}, u, v) \\
&= p(u, v|G_{i-1})p(NodeMerge(u, v, G_{i-1})|G_{i-1}, u, v) \\
&= Uniform(\{(u, v)|\forall u, v \in \mathcal{V}_2^{G_{i-1}}\})Bernoulli(\sigma(h_u^T h_v)|u, v)
\end{aligned} \tag{10}
$$

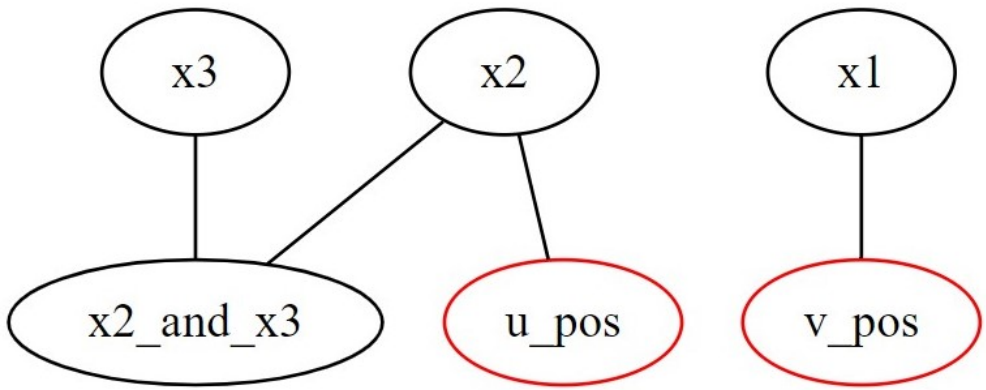
Training: Positive Sampling

Choose a random node s with $Deg(s) > 1$.



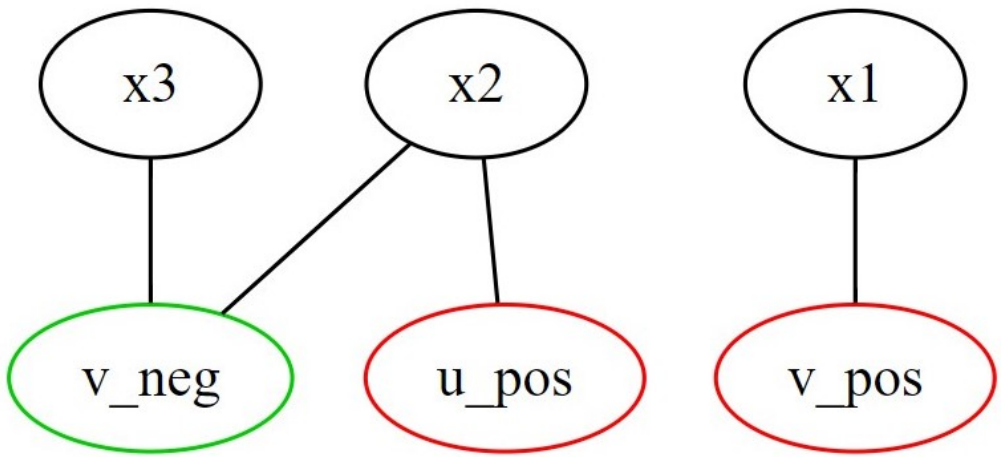
Training: Positive Pair

Split it into (u^+, v^+), thus it is a positive pair.



Training: Negative Sampling

Select another node in $\mathcal{V}_2^{G_{i-1}}$ as negative sample, thus u^+, v^- is a negative pair.

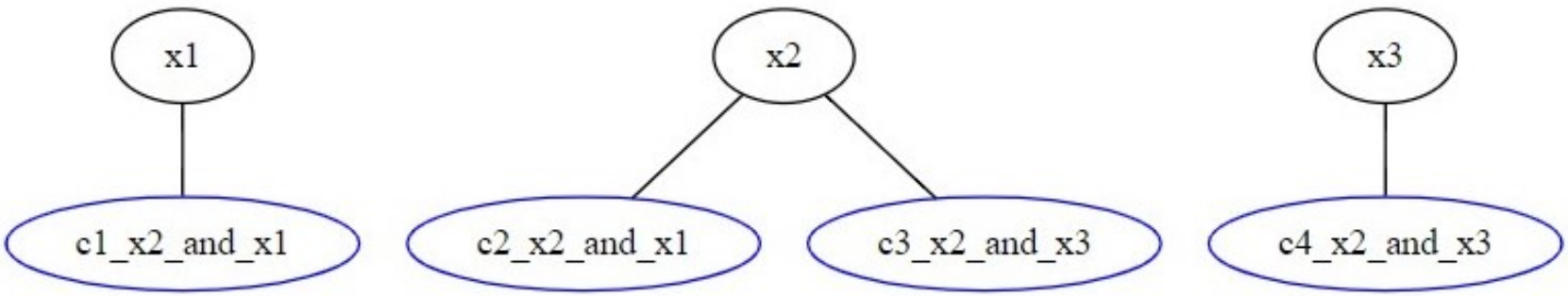


Training Loss

A binary classification problem.

$$\mathcal{L} = -\mathbb{E}_{u^+,v^+}[\log(\sigma(h_u^T h_v))] - \mathbb{E}_{u^+,v^-}[\log(1-\sigma(h_u^T h_v))] \quad (11)$$

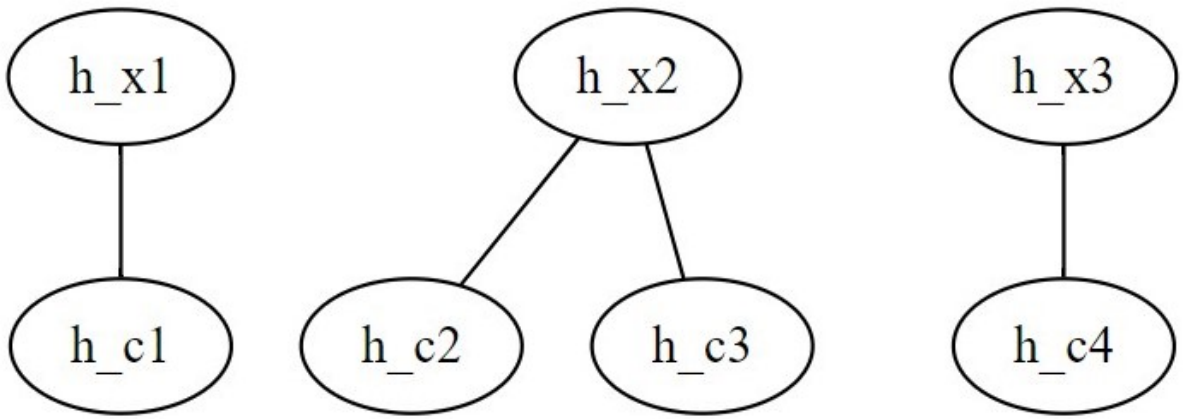
The end of training and the beginning of inference



Note: $Deg(x_2 \vee x_3) = 1, Deg(x_2 \vee x_1) = 1$
Save G_{i-1} as G_0!

Inference

Select pairs with high similarity in parallel. Although biased, the result is resonable.



Experiment1

- Similarity of properties between training graph and generated graph.
- Two properties for Graph statistics:
  - Modularity
  - Clustering Coefficient

  Hence, we can measure the how much does the generated graph maintain the properties of training data.

Experiment1: Graph Statistics

Table 1: Graph statistics of generated formulas (mean ± std. (relative error to training formulas)).

Method	VIG		VCG			LCG
	Clustering	Modularity	Variable α_v	Clause α_c	Modularity	Modularity
Training	0.50±0.07	0.58±0.09	3.57±1.08	4.53±1.09	0.74±0.06	0.63±0.05
CA	0.33±0.08(34%)	0.48±0.10(17%)	6.30±1.53(76%)	N/A	0.65±0.08(12%)	0.53±0.05(16%)
PS(T=0)	0.82±0.04(64%)	0.72±0.13(24%)	3.25±0.89(9%)	**4.70±1.59(4%)**	0.86±0.05(16%)	**0.64±0.05(2%)**
PS(T=1.5)	0.30±0.10(40%)	0.14±0.03(76%)	4.19±1.10(17%)	6.86±1.65(51%)	0.40±0.05(46%)	0.41±0.05(35%)
G2SAT	**0.41±0.09(18%)**	**0.54±0.11(7%)**	**3.57±1.08(0%)**	4.79±2.80(6%)	**0.68±0.07(8%)**	0.67±0.03(6%)

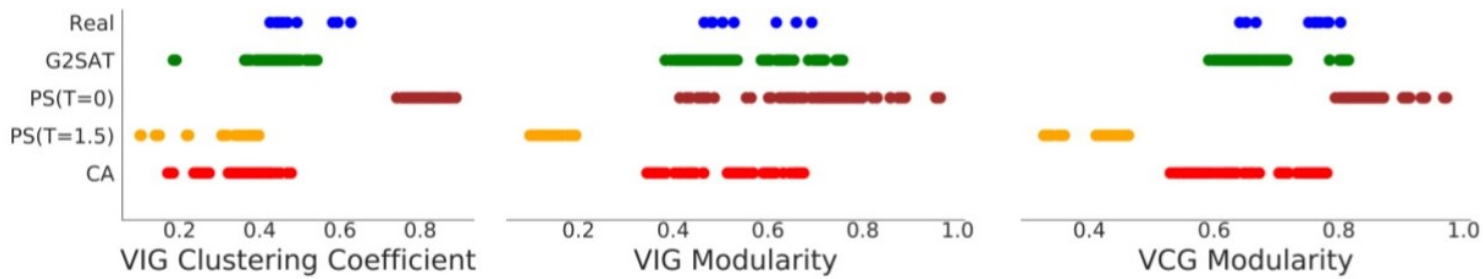


Figure 2: Scatter plots of distributions of selected properties of the generated formulas.

Experiment2

- Comparing the performance of SAT solvers both on real data and generated data.
- Training deep SAT solvers on generated data boosts their performance on real data.

Experiment2: SAT Solver Performance

Table 2: Relative SAT Solver Performance on training as well as synthetic SAT formulas.

Method	Solver ranking	Accuracy
Training	$I_2, I_3, I_1, R_2, R_3, R_1$	100%
CA	$I_2, I_3, I_1, R_2, R_3, R_1$	**100%**
PS(T=0)	$R_3, I_3, R_2, I_2, I_1, R_1$	33%
PS(T=1.5)	$R_3, R_2, I_3, I_1, I_2, R_1$	33%
G2SAT	$I_1, I_2, I_3, R_2, R_3, R_1$	**100%**

Table 3: Performance gain when using generated SAT formulas to tune SAT solvers.

Method	Best parameters	Runtime(s)	Gain
Training	(0.95, 0.9)	2679	N/A
CA	(0.75, 0.99)	2617	2.31%
PS(T=0)	(0.75, 0.999)	2668	0.41%
PS(T=1.5)	(0.95, 0.9)	2677	0.07%
G2SAT	(0.95, 0.99)	**2190**	**18.25%**

Summary

- The training objective of the model is to decide "what node to merge". In essence, the model learns the intra-clause pattern and inter-literal pattern from training data.
- Training phase is node splitting while inference phase is node merging.
- To prune hypothesis space, G2SAT relaxes $p(G_i|G_{i-1})$ to $p(G_i, u, v|G_{i-1})$, thus transforming multinomial distribution to binary distribution.

Inspiration

- Decomposition: When the objective is hard to learn, we can solve the problem iteratively via intermediate objectives.
- Hybrid Model: We can use neural networks to obtain appropriate feature embeddings. This work doesn't exploit the reciprocation between statistical learning models and logical learning models.
- Reduce multinomial distribution to binary distribution.
- Reduce enumerative traversal in the heposythesis space to random selection when the operation is commutative (i.e. the result is independent of operation sesquence). (Maybe similar to changing sliding window to anchor boxes in CV?)