Explaining Anomalies in Graphs with Grammars

Daniel Gonzalez Cedre

17th of August, 2023

Introduction to the Problem

Service	Task	Archetype
Risk Mitigation	Assessing and hedging risk	Modeling, Prediction
Financial Services	Accounting, taxes, auditing	Modeling, Pattern mining
Fraud Detection	Finding & predicting suspicious behavior	Classification, Clustering
Preventative Maintenance	Predicting when to service Mechanical components	Regression, Graph completion
Cyber Security	Preventing attacks and accidental breaches	Anomaly detection, Graph completion

Service	Task	Archetype
Risk Mitigation	Assessing and hedging risk	Modeling, Prediction
Financial Services	Accounting, taxes, auditing	Modeling, Pattern mining
Fraud Detection	Finding & predicting suspicious behavior	Classification, Clustering
Preventative Maintenance	Predicting when to service Mechanical components	Regression, Graph completion
Cyber Security	Preventing attacks and accidental breaches	Anomaly detection, Graph completion

We want to detect *fraudulent transactions* and *suspicious agents*.

A special case of *anomaly detection*, this is typically addressed by *classifying* or *clustering* datapoints.

This has clear applications to detecting *Medicare fraud, money laundering, suspicious transactions fake user reviews*, and more.

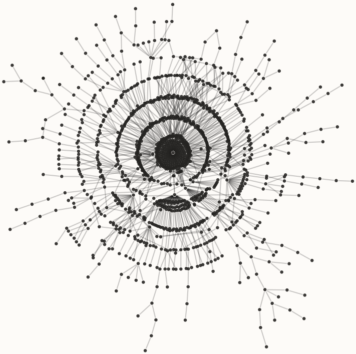
Anomaly detection is relevant to many problems posed on relational data.

1. Classifying Medicare providers as categorically *risky*.
2. Given hubs of American Express salespeople:
   - find salespeople with high rates of misconduct complaints,
   - provide *behavioral explanations* for these decisions,
   - and give the client a way to monitor their hubs.
3. Detecting and predicting *fraud* on financial interaction data.
4. Given a network of sensors and a database of facts for an assembly line,
   - find *subnetworks* with *elevated risk* profiles,
   - preventively predict when hardware will need maintenance.

Agents can often be characterized by *behavioral interaction patterns* and *domain-specific features*.

Behavioral patterns are often *infrequent* and *difficult to detect*, and *features* alone are not enough.

We want to leverage both of these modalities to bisect the latent space.



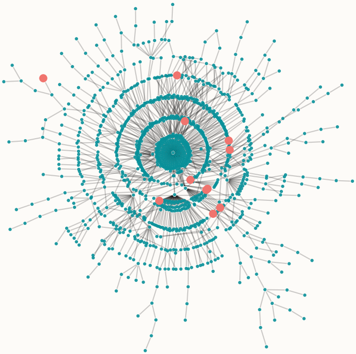
The largest connected component of the labeled EllipticBitcoin[a] dataset. Nodes in *blue* are *licit agents*, while those in *red* are *illicit*.

[a] Weber, M. et al. *Anti-Money Laundering in Bitcoin*. 2019.

Agents can often be characterized by *behavioral interaction patterns* and *domain-specific features*.

Behavioral patterns are often *infrequent* and *difficult to detect*, and *features* alone are not enough.

We want to leverage both of these modalities to bisect the latent space.



The largest connected component of the labeled EllipticBitcoin[a] dataset. Nodes in *blue* are *licit agents*, while those in *red* are *illicit*.

[a] Weber, M. et al. *Anti-Money Laundering in Bitcoin*. 2019.

Since these decisions are often made in *high-stakes situations*[1] and often subject to strict *scrutiny* and *regulation*, *justifying decisions* is often crucial.

We need an *explainable* way to make these decisions.

[1] *e.g.*, Medicare, finance, commerce

Node-based explainers can be thought of as 0^{th}-order explanations. Edge-based explainers are then 1^{st}-order aggregates.

Subgraph[2,3] explainers aggregate *higher-order* information, but are hierarchically ambiguous.

Can we gain context by arranging subgraph explanations *hierarchically*?

[2] Yuan, H. et al. "On Explainability of Graph Neural Networks via Subgraph Explorations". 2021.

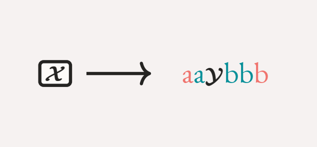
[3] Serra, G. and Niepert, M. *L2XGNN: Learning to Explain Graph Neural Networks*. 2023.

Proposed Solution

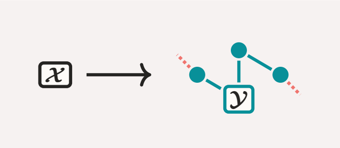
Can we use *graph grammars* for *explainable anomaly detection*?

Well, what *is* a graph grammar?

A rule-based generative model for graphs.



(a) A string rule with *left-hand* 𝒳 and right-hand string with boundary characters.



(b) A graph rule with *left-hand* 𝒳 and right-hand graph with boundary edges.

Graph grammars are often used for

1. Molecular generation and drug synthesis[4][5][6]
2. Statistical hypothesis testing and generative modeling[7][8][9][10]
3. Software engineering and formal methods[11][12][13]

---

[4] Guo, M. et al. "Polygrammar: Grammar for Digital Polymer Representation and Generation". 2022b.

[5] Guo, M. et al. "Data-Efficient Graph Grammar Learning for Molecular Generation". 2022a.

[6] Guo, M. et al. "Hierarchical Grammar-Induced Geometry for Data-Efficient Molecular Property Prediction". 2023.

[7] Sikdar, S. et al. "The Infinity Mirror Test for Graph Models". 2023.

[8] Sikdar, S., Hibshman, J., and Weninger, T. "Modeling Graphs with Vertex Replacement Grammars". 2019.

[9] Sikdar, S., Shah, N., and Weninger, T. "Attributed Graph Modeling with Vertex Replacement Grammars". 2022.

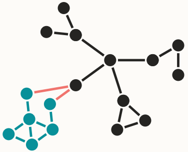
[10] Hibshman, J., Sikdar, S., and Weninger, T. "Towards Interpretable Graph Modeling with Vertex Replacement Grammars". 2019.

[11] Baresi, L. and Heckel, R. "Tutorial introduction to graph transformation: A software engineering perspective". 2002.

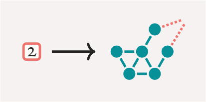
[12] Engels, G., Lewerentz, C., and Schäfer, W. "Graph grammar engineering: A software specification method". 1987.

[13] Zhao, C., Kong, J., and Zhang, K. "Program behavior discovery and verification: A graph grammar approach". 2010.

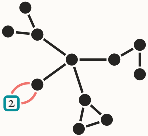
How do we traditionally *learn* rules?



(a) Select a subgraph and compute its boundary.



(b) Create a rule corresponding to the subgraph and boundary.

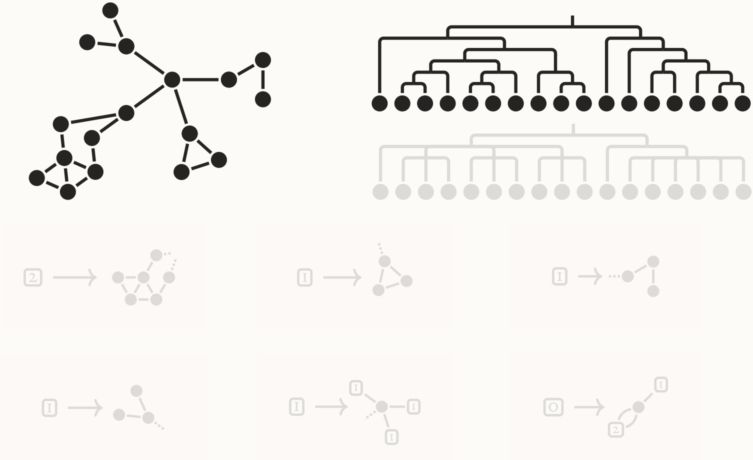


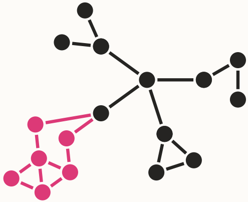
(c) Compress the subgraph.

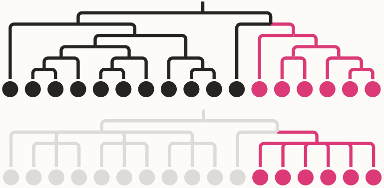
Subgraphs are induced by *hierarchical clustering*.

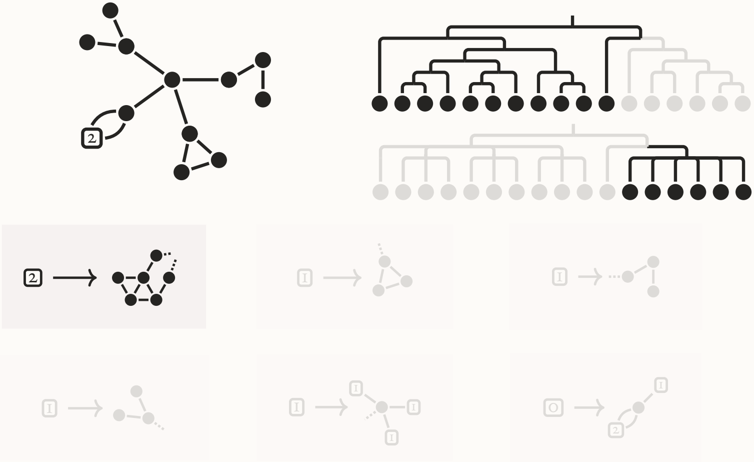
Proposed Solution

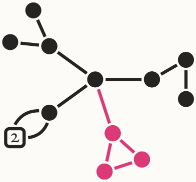
An Example

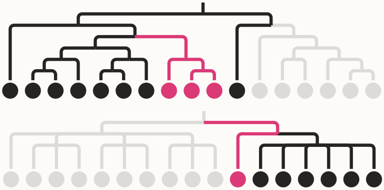


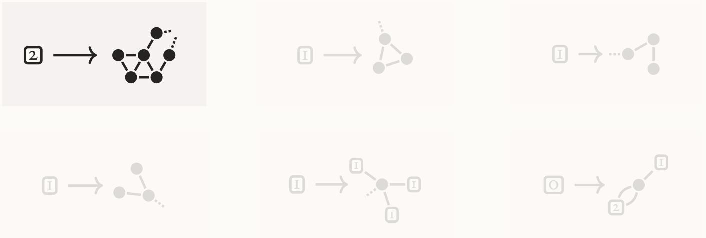






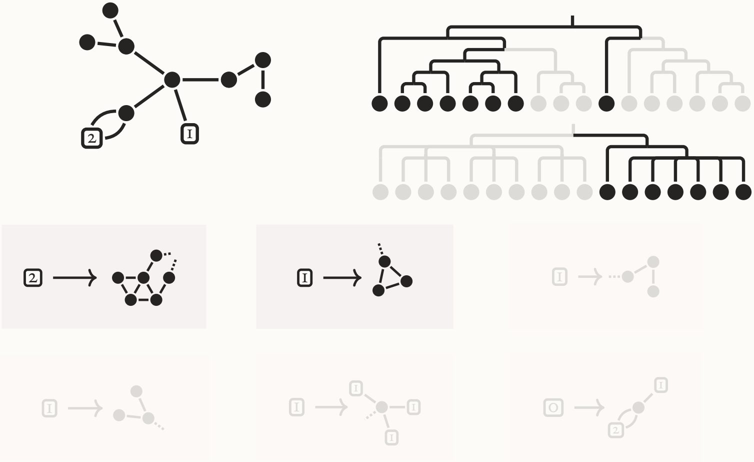

2

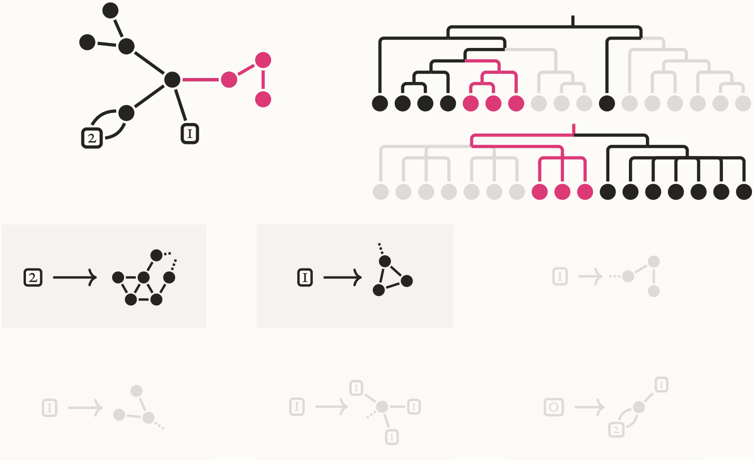


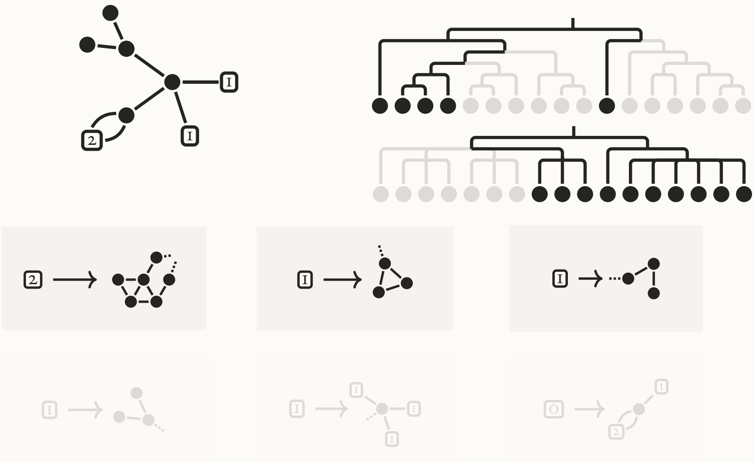

2
1
1
1
1
1
1
1
1
0
1
2

Proposed Solution

An Example

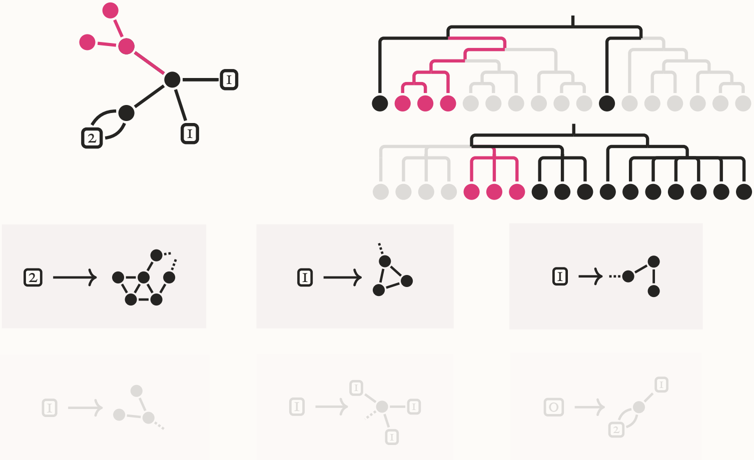


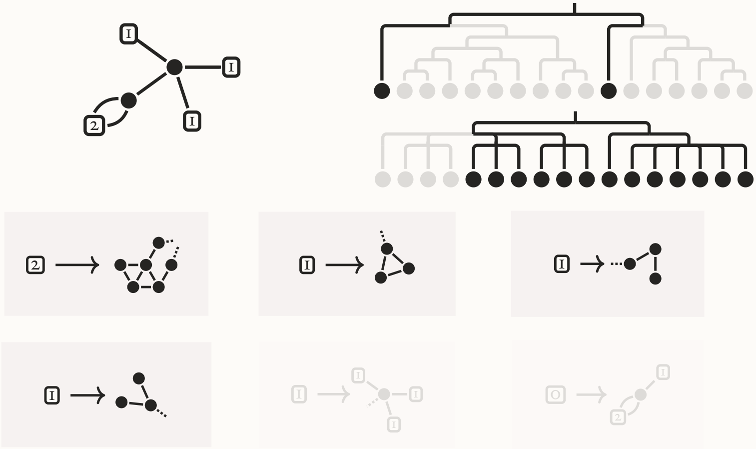

2
1
2
1

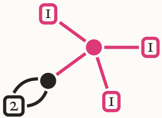

1
1
2
2
1
1
1
1
0
1
2

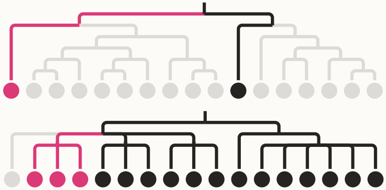
Proposed Solution

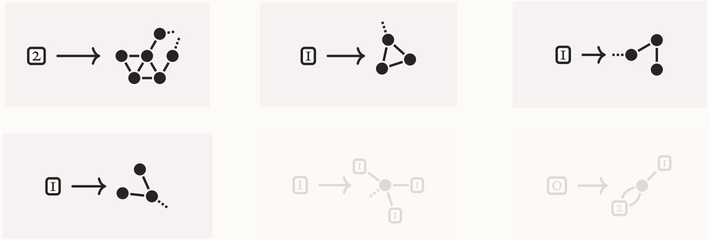
An Example




1
1
2
1
2
1
1
1
1
1
1
1
0
1
2

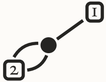

1
1
1
2

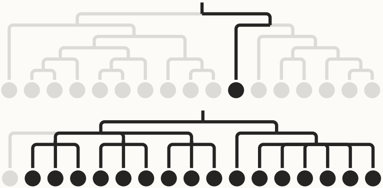


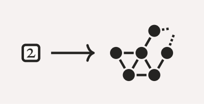

2
1
1
1
1
1
1
1
1
0
1
2

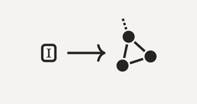
Proposed Solution

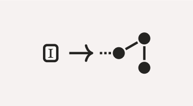
An Example

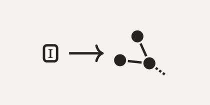


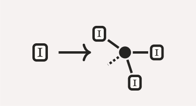


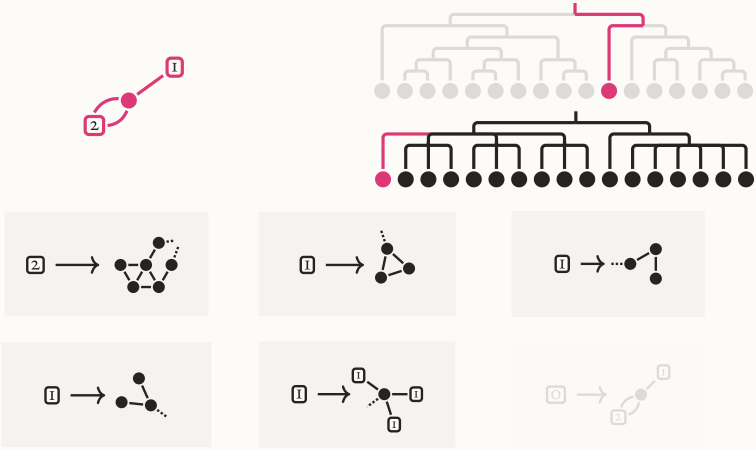


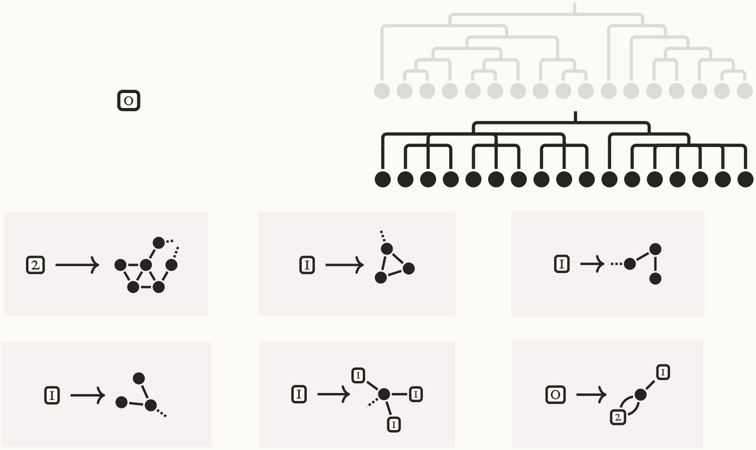




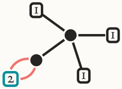




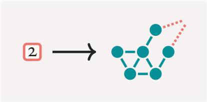




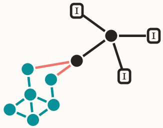
How do we *use* rules for generation?



(a) Select a nonterminal with its boundary edges.



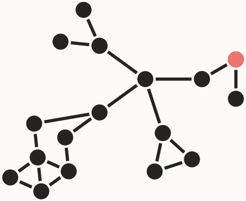
(b) Choose a rule with matching *left-hand* side.



(c) Integrate the subgraph with its boundary edges.

This is an example of a *rule derivation*.

Grammar rules can highlight *subgraphs* relevant to particular *node decisions*.



Above, we found a *suspicious node*.

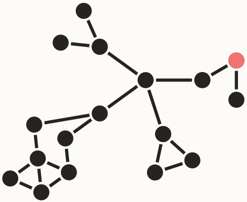
Rule explanation.

Subgraphs covered by isomorphic rules.

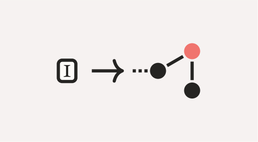
Proposed Solution

Rules as Subgraph Explainers

Grammar rules can highlight *subgraphs* relevant to particular *node decisions*.



Above, we found a *suspicious node*.

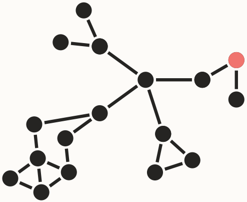


Rule explanation.

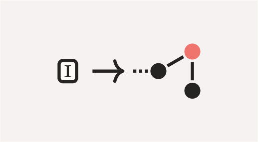
Subgraphs covered by isomorphic rules.

Rules as Subgraph Explainers

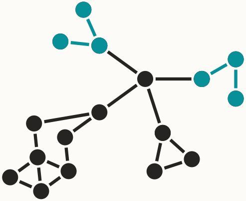
Grammar rules can highlight *subgraphs* relevant to particular *node decisions*.



Above, we found a *suspicious node*.

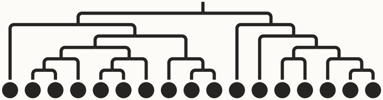


Rule explanation.

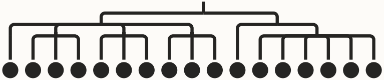


Subgraphs covered by isomorphic rules.

Graph grammars are traditionally determined by *heuristic methods* like hierarchical clustering[14][15], tree decomposition[16][17], or hand-written rules.



Hierarchical clustering dendrogram



Grammar-induced parse tree

[14] Sikdar, S., Hibshman, J., and Weninger, T. "Modeling Graphs with Vertex Replacement Grammars". 2019.

[15] Sikdar, S., Shah, N., and Weninger, T. "Attributed Graph Modeling with Vertex Replacement Grammars". 2022.

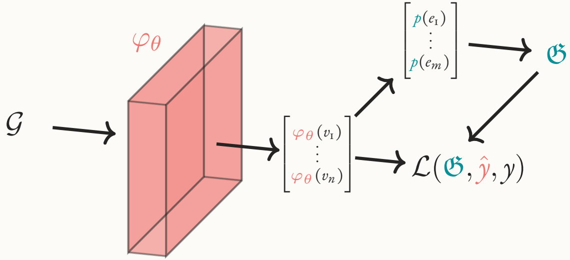
[16] Aguiñaga, S. et al. "Growing Graphs from Hyperedge Replacement Graph Grammars". 2016.

[17] Aguiñaga, S., Chiang, D., and Weninger, T. "Learning Hyperedge Replacement Grammars for Graph Generation". 2019.

Can we learn *data-driven grammar rules*?

Neural Architecture

Grammar Neural Network Architecture



- $\mathcal{G}$: the input graph
- φ_θ : graph neural network
- $\varphi_\theta(v_i)$: node embeddings
- $p(e_i)$: edge probabilities
- $\mathfrak{G}$: graph grammar
- $\hat{y}$: predicted node labels
- y : true node labels
- $\mathcal{L}$: loss function

Given an input graph $\mathcal{G}$, we compute node embeddings $\varphi_\theta(v_i)$ and predict class labels $\hat{y}$. The embeddings also determine probabilities $p(e_i)$ for *iid* Bernoulli random variables on each edge. We obtain a grammar $\mathfrak{G}$ by iteratively sampling edges. The *joint* loss function $\mathcal{L}(\mathfrak{G}, \hat{y}, y)$ then optimizes for a *good grammar and good classification performance*.

For simplicity, nodes are embedded by a graph isomorphism network[18]

$$\varphi_\theta(\mathbf{X}) = \mathrm{MLP}\Big(\big(\mathbf{A} + (1 + \varepsilon)\mathbf{I}\big)\mathbf{X}\Big)$$

with edge embeddings as a paraboloidal function of node embeddings[19]

$$p(x_u, x_v) = \frac{\left(4{x_u}^2 - 4x_u + 1\right) + \left(4{x_v}^2 - 4x_v + 1\right) + 2\varepsilon}{2 + 4\varepsilon}$$

[18] Xu, K. et al. "How Powerful are Graph Neural Networks?" 2019.

[19] Since $\varphi_\theta(\mathbf{X})$ is a stochastic tensor, $p(x_u, x_v)$ can be treated like probabilities.

We jointly optimize an affine combination of two loss functions

$$\mathcal{L}(\mathfrak{G}, \hat{y}, y) = \lambda\mathcal{L}(\mathfrak{G}) + (1 - \lambda)\mathcal{L}(\hat{y}, y)$$

with a *loss for the classification task* given by

$$\mathcal{L}(\hat{y}, y) = \text{CrossEntropy}(\hat{y}, y)$$

The *loss for the grammar* should be inversely proportional to the number of patterns the rules describe, which we call *compressibility* here

$$\mathcal{L}(\mathfrak{G}) \propto 1 - \text{Compressibility}(\mathfrak{G}) = \frac{\text{\# of rules up-to-isomorphism}}{\text{\# of distinct rules}}$$

For gradient optimization, we would like to compute $\nabla_\theta \mathcal{L}(\mathfrak{G}, \hat{y}, y)$ but we can't because $\mathcal{L}(\mathfrak{G})$ is not differentiable in θ.

Treat $\mathfrak{G}$ like a *random variable*[20] and apply the policy-gradient theorem[21]

$$\nabla_\theta \mathbb{E}\left[\mathcal{L}(\mathfrak{G})\right] = \mathbb{E}\left[\mathcal{L}(\mathfrak{G})\left(\nabla_\theta \log p(\mathfrak{G})\right)\right]$$

and estimate the expectation using Monte Carlo sampling

$$\mathbb{E}\left[\mathcal{L}(\mathfrak{G})\left(\nabla_\theta \log p(\mathfrak{G})\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\mathfrak{G}_i)\left(\nabla_\theta \log p(\mathfrak{G}_i)\right)$$

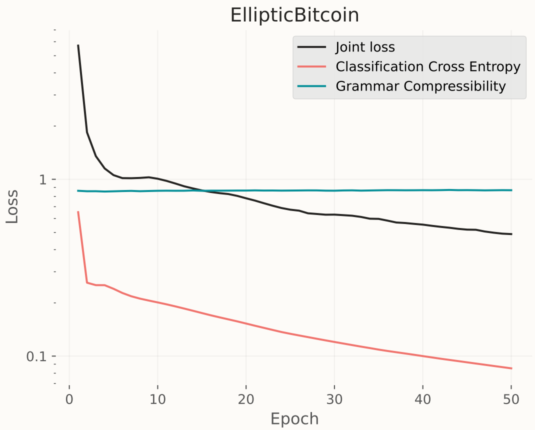
[20] Recall that we construct $\mathfrak{G}$ by randomly sampling edges.

[21] Williams, R. J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". 1992.

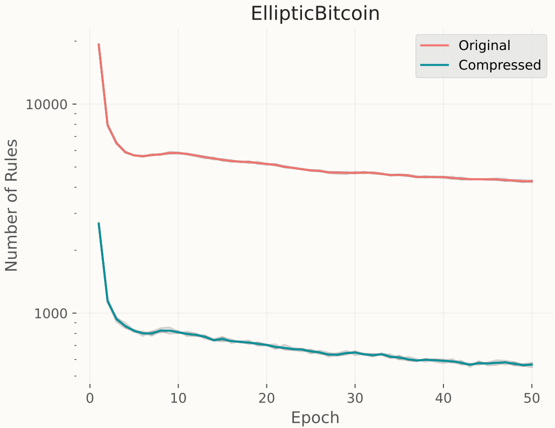
Results

Description of the EllipticBitcoin dataset.

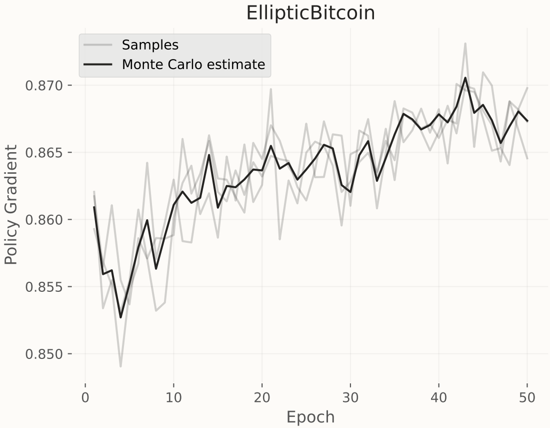
	Total	Labeled	# Licit	# Illicit	# Features
Nodes	203 769	46 564	42 019	4 545	165
Edges	234 355	36 624	–	–	–



A comparison of the three different loss functions over 50 epochs.
We have clearly yet to converge the node classifier.

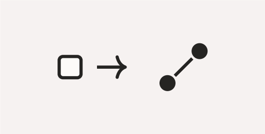


Average sizes of the grammars before and after compressing isomorphic rules.



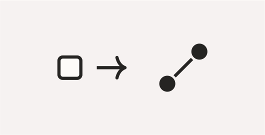
Average of four Monte Carlo samples estimating $\mathbb{E}\left[\mathcal{L}(\mathfrak{G})\left(\nabla_\theta \log p(\mathfrak{G})\right)\right]$.

Epoch 1

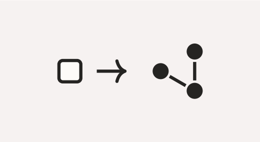


The most popular rule from one of the sampled grammars at epoch 1.
This rule occurred 2093 times.

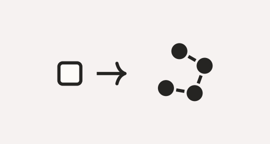
Epoch 50



×1208



×437



×174

The three most frequent rules from one of the grammars at epoch 50. Their respective frequencies are shown below.

Next steps on the way to publication:

1. Determine an adequate number of Monte Carlo samples per epoch.
2. Find good values for the hyperparameter λ that weighs the losses.
3. GINs may be *underparametrized*—consider GCNs or GATs.
4. Experiment with more sophisticated grammar losses.
5. Consider *parametrizing* the *edge embeddings*.
6. Find a better way to determine *nonterminal symbols*.

Acknowledgements

Sanmitra Bhattacharya

Tim Weninger

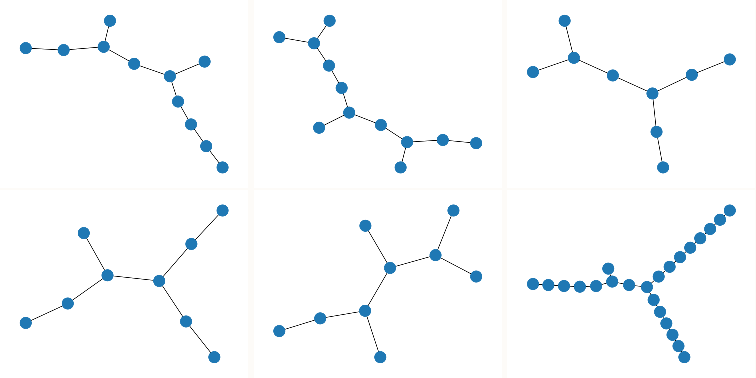
Sal Aguiñaga

Balaji Veeramani

Sunil Reddy Tiyyagura

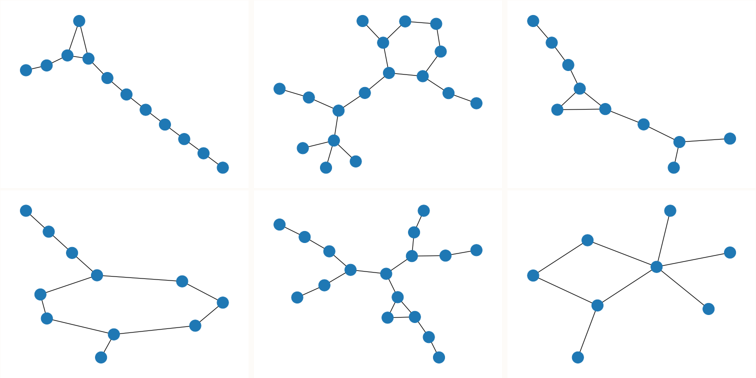
Thank you!

Epoch 50



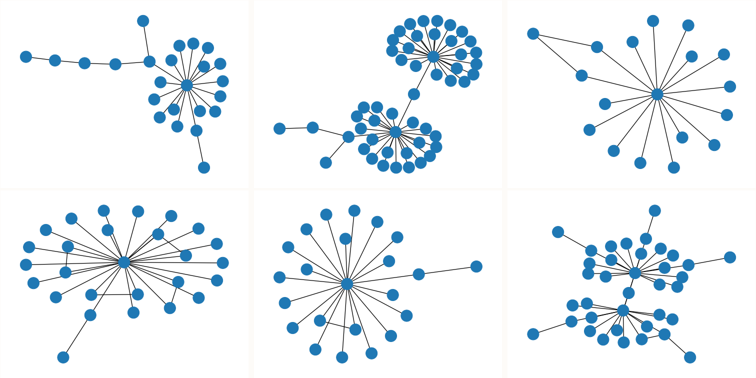
Examples of *(near)-combs* of various lengths.

Epoch 50



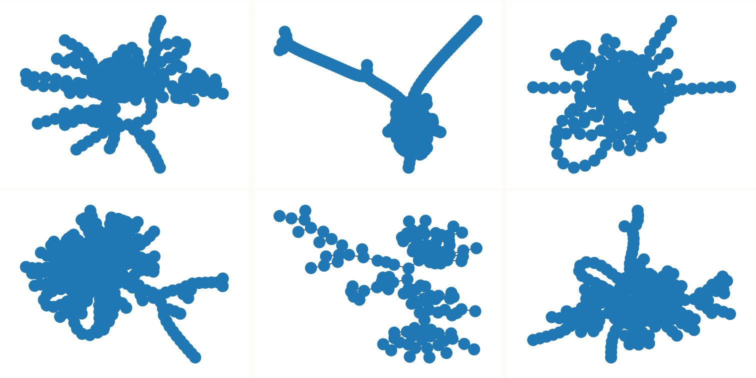
Examples of *narrow* subgraphs with one cycle.

Epoch 50



Examples of subgraphs with *large hubs*.

Epoch 50



Examples of *weird* subgraphs.