A Transformational Approach to Graph Learning

Daniel Gonzalez Cedre

University of Notre Dame

25th of April, 2023

Introduction

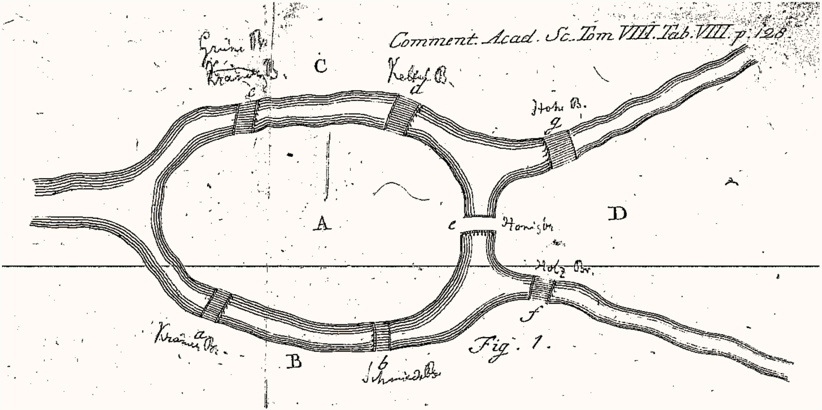
- Born in Havana, Cuba
- Education
    - B.Sc. in Comp. Sci. (FIU)
    - B.Sc. in Math (FIU)
    - M.Sc. in Math (FSU)
- Teaching
    - Precalculus Algebra at FSU
    - Discrete Math at ND
        - Spring 2022
        - Spring 2023
        - Fall 2023
    - Summer 2023 with William Theisen at ND

1. The Infinity Mirror Test for Graph Models[1]
   - Satyaki Sikdar, Daniel Gonzalez Cedre, Trenton W. Ford, Tim Weninger
2. Subgraph-to-Subgraph Transitions[2]
   - Justus Hibshman, Daniel Gonzalez Cedre, Satyaki Sikdar, Tim Weninger
3. Motif Mining[3]
   - William Theisen, Daniel Gonzalez Cedre, Zachariah Carmichael, Daniel Moreira, Tim Weninger, Walter Scheirer

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

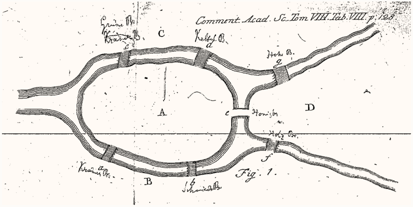
[2] Hibshman, J. I. et al. "Joint Subgraph-to-Subgraph Transitions: Generalizing Triadic Closure for Powerful and Interpretable Graph Modeling". 2021.

[3] Theisen, W. et al. "Motif Mining: Finding and Summarizing Remixed Image Content". 2023.

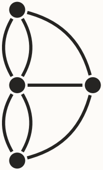


The 7 bridges of Königsberg, Prussia.[4]

[4] Euler, L. "Solutio problematis ad geometriam situs pertinentis". 1736.



The 7 bridges of Königsberg.

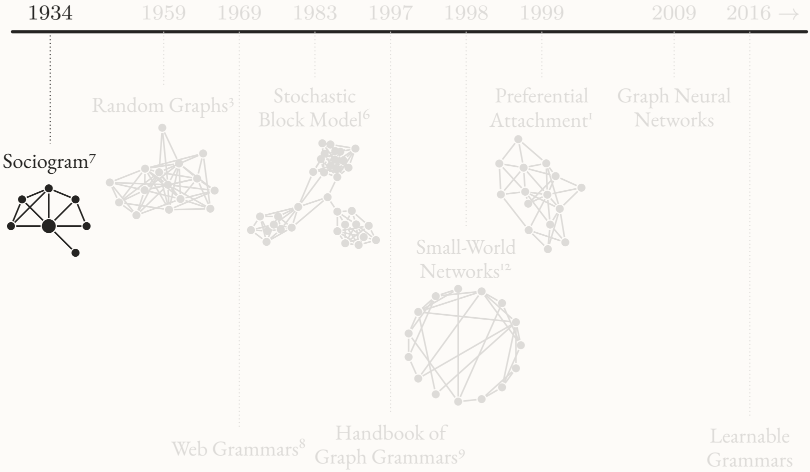


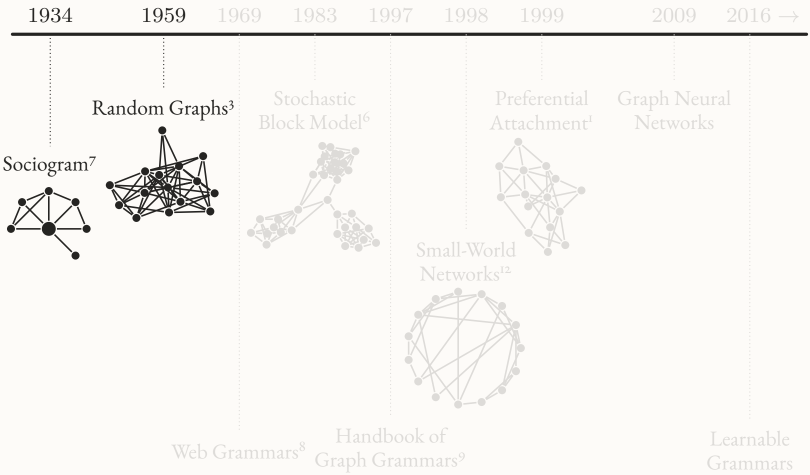
A multigraph representing the bridges of Königsberg.

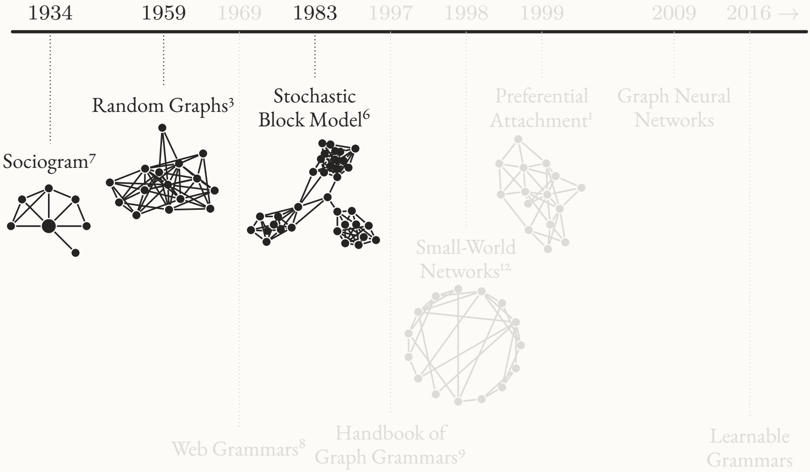
Graphs are abstractions of connectivity structures.

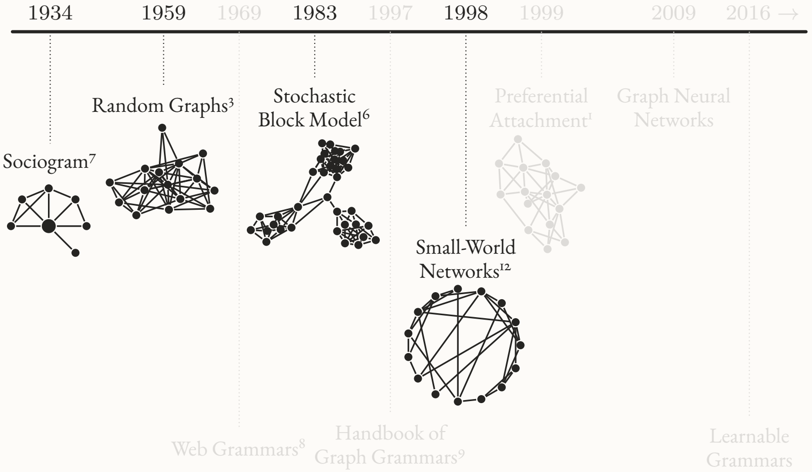
The Fundamental Question of Data Modeling

How do we find an *insightful abstraction*
that *accurately* characterizes some data?

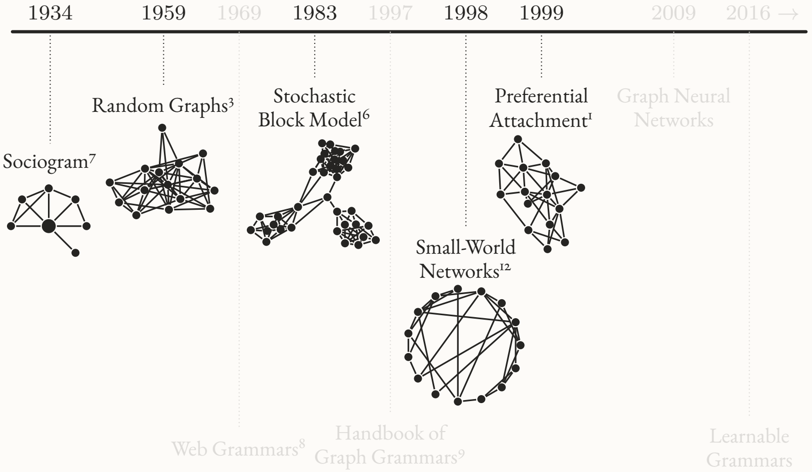

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Stochastic Block Model[6]
Preferential Attachment[1]
Graph Neural Networks
Small-World Networks[12]
Web Grammars[8]
Handbook of Graph Grammars[9]
Learnable Grammars

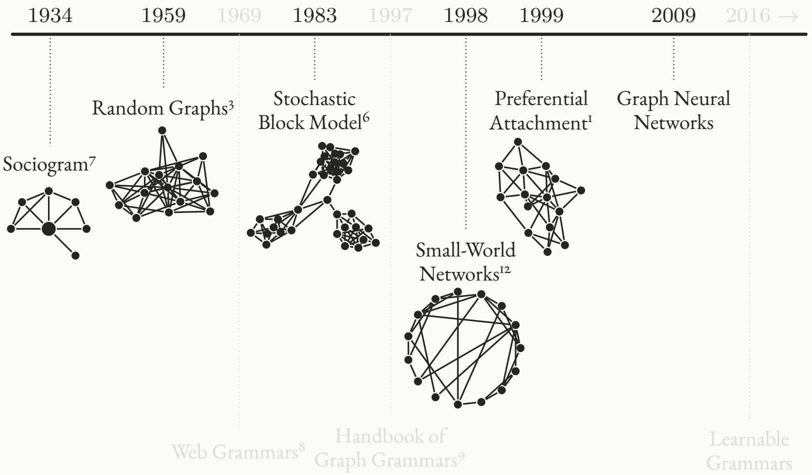

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Stochastic Block Model[6]
Preferential Attachment[1]
Graph Neural Networks
Small-World Networks[12]
Web Grammars[8]
Handbook of Graph Grammars[9]
Learnable Grammars

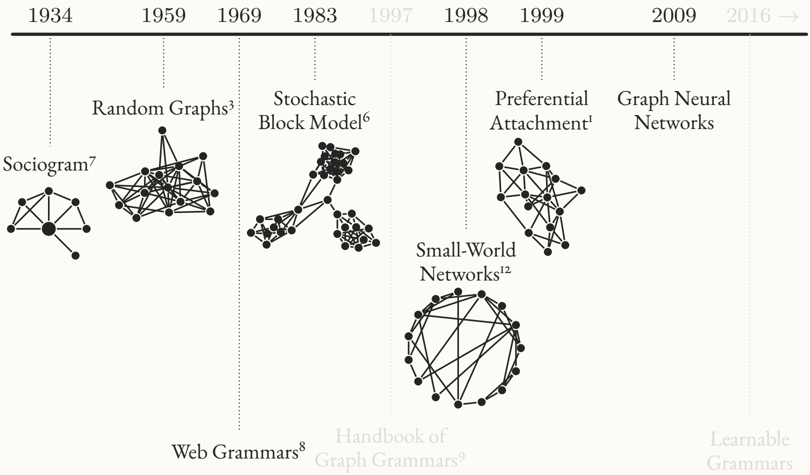

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Stochastic Block Model[6]
Preferential Attachment[1]
Graph Neural Networks
Small-World Networks[12]
Web Grammars[8]
Handbook of Graph Grammars[9]
Learnable Grammars

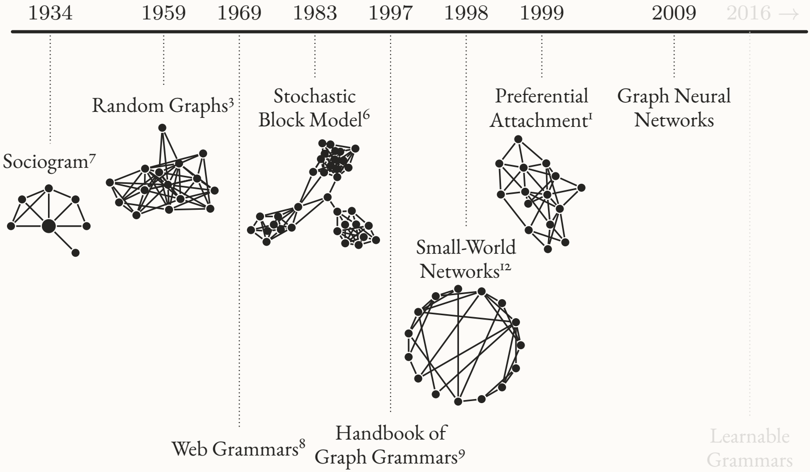

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Stochastic Block Model[6]
Small-World Networks[12]
Preferential Attachment[1]
Graph Neural Networks
Web Grammars[8]
Handbook of Graph Grammars[9]
Learnable Grammars

A Brief History of Graph Modeling

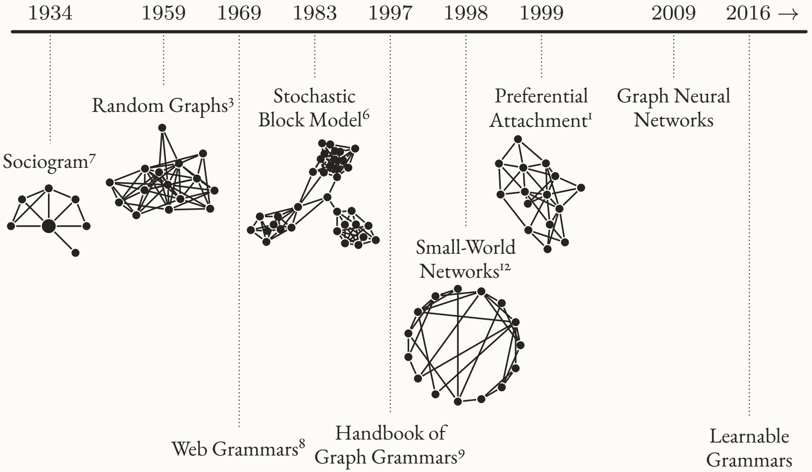


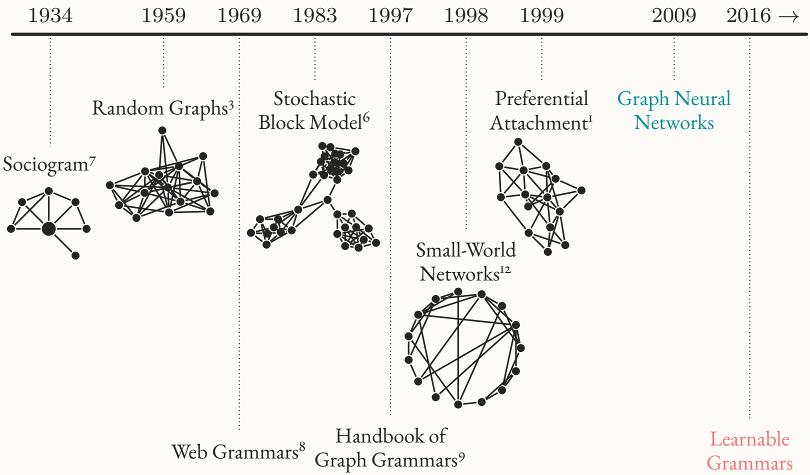

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Stochastic Block Model[6]
Small-World Networks[12]
Preferential Attachment[1]
Graph Neural Networks
Web Grammars[8]
Handbook of Graph Grammars[9]
Learnable Grammars

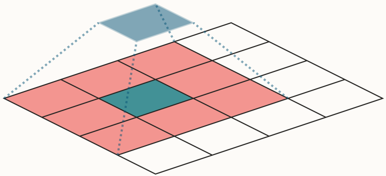

1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Web Grammars[8]
Stochastic Block Model[6]
Handbook of Graph Grammars[9]
Small-World Networks[12]
Preferential Attachment[1]
Graph Neural Networks
Learnable Grammars


1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Web Grammars[8]
Stochastic Block Model[6]
Handbook of Graph Grammars[9]
Small-World Networks[12]
Preferential Attachment[1]
Graph Neural Networks
Learnable Grammars

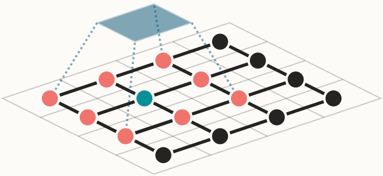
A Brief History of Graph Modeling




1934
1959
1969
1983
1997
1998
1999
2009
2016 →
Sociogram[7]
Random Graphs[3]
Web Grammars[8]
Stochastic Block Model[6]
Handbook of Graph Grammars[9]
Small-World Networks[12]
Preferential Attachment[1]
Graph Neural Networks
Learnable Grammars



(a) *Convolutions* aggregate neighborhoods around pixels to update embeddings.



(b) *Message passing* aggregates neighborhoods around nodes to update embeddings.

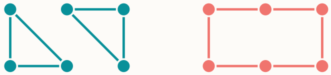
Graph Neural Networks

Benefits

- Highly performant on a variety of graph learning tasks
- *Equivariance* and *permutation invariance* are good modeling assumptions for graphs

Drawbacks

- Difficult to interpret
- Hidden inductive biases[5]
- Limited by message passing

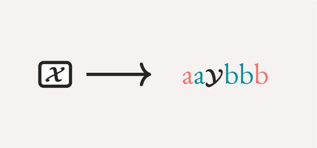


1-hop message passing can not distinguish *two 3-cycles* from a *6-cycle*.[6]

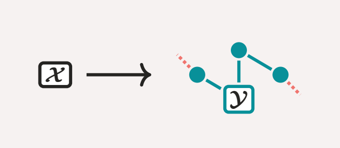
[5] Sikdar et al. [10]

[6] Chen et al. [2]

Graph neural nets are accurate, but not always insightful.



(a) A string rule with *left-hand* $\mathcal{X}$ and right-hand string with boundary characters.



(b) A graph rule with *left-hand* $\mathcal{X}$ and right-hand graph with boundary edges.

Graph Grammars

Benefits

- Rules are interpretable
- Derivations are explainable
- Rules carry semantic meaning
- Relatively easy to analyze
- Highly performant on constrained generative tasks
  - molecule generation
  - robot design

Drawbacks

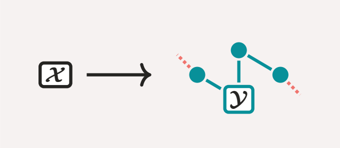
- Not competitive at unconstrained generation
  - social networks
  - temporal processes
  - stochastic modeling
- Modeling assumptions may be inappropriate for data
- Reliance on heuristics for determining hyperparameters
  - size of rules
  - boundary conditions

Graph grammars are insightful, but not always accurate.

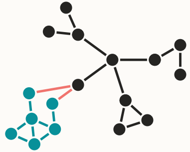
Dynamic Vertex Replacement Grammars

A vertex replacement grammar $\mathcal{G}$ is specified by a set $\mathcal{R}$ of rules

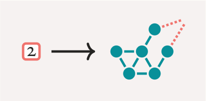
- rules' left-hand sides consist of a single *nonterminal* node
- rules' right-hand sides are graphs with terminal, *nonterminal*, and boundary structures



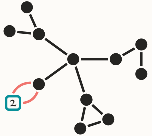
How do we *learn* rules?



(a) Select a subgraph and compute its boundary.

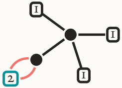


(b) Create a rule corresponding to the subgraph and boundary.

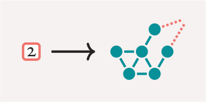


(c) Compress the subgraph.

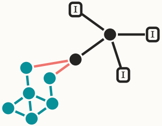
How do we *apply* rules?



(a) Select a nonterminal with its boundary edges.

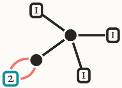


(b) Choose a rule with matching *left-hand* side.

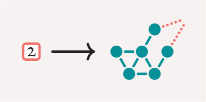


(c) Replace the nonterminal with the subgraph and rewire boundary edges randomly.

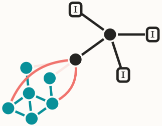
How do we *apply* rules?



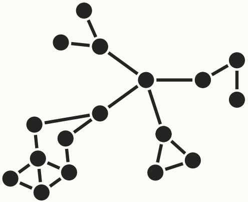
(a) Select a nonterminal with its boundary edges.

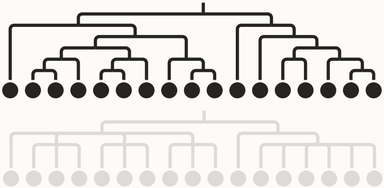


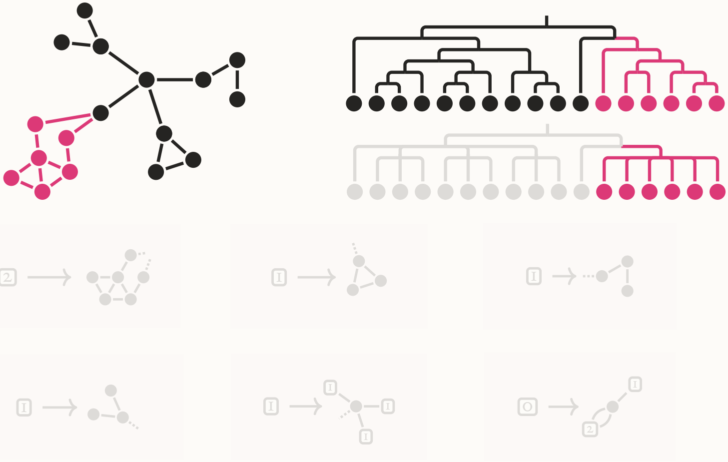
(b) Choose a rule with matching *left-hand* side.

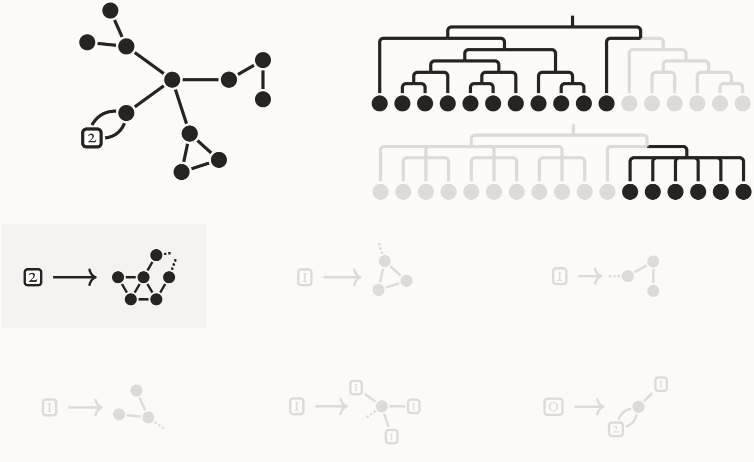


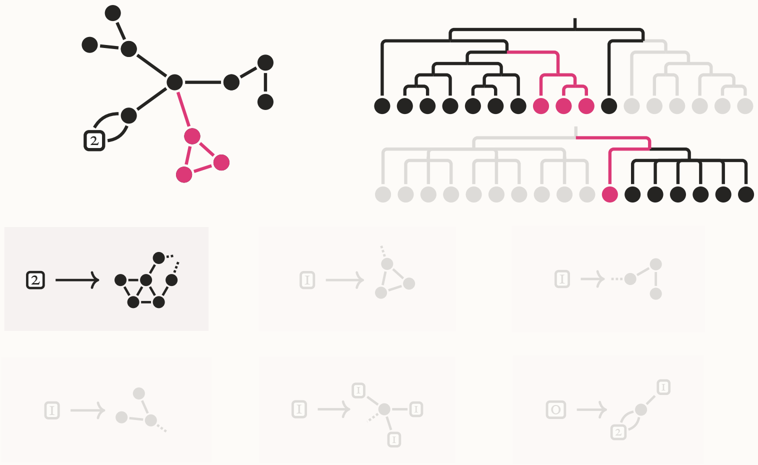
(c) Replace the nonterminal with the subgraph and rewire boundary edges randomly.

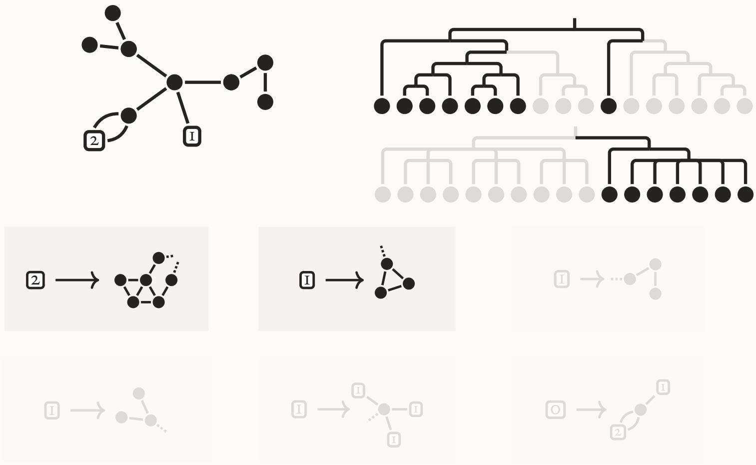


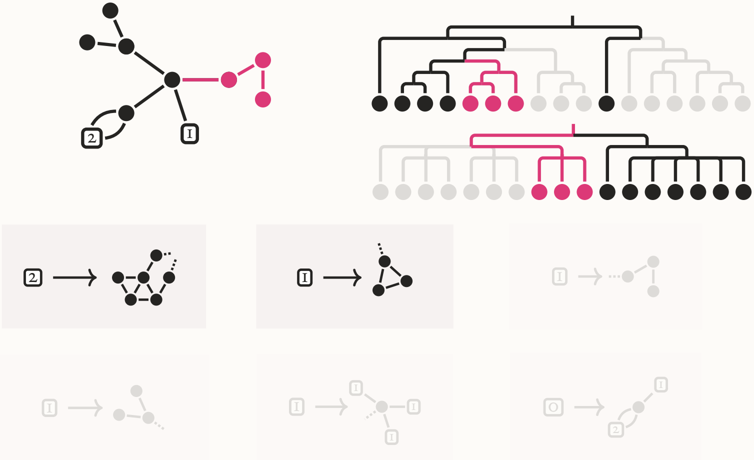


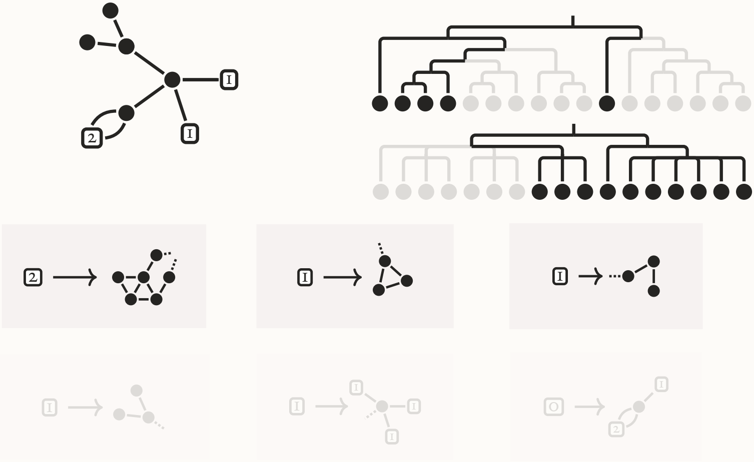


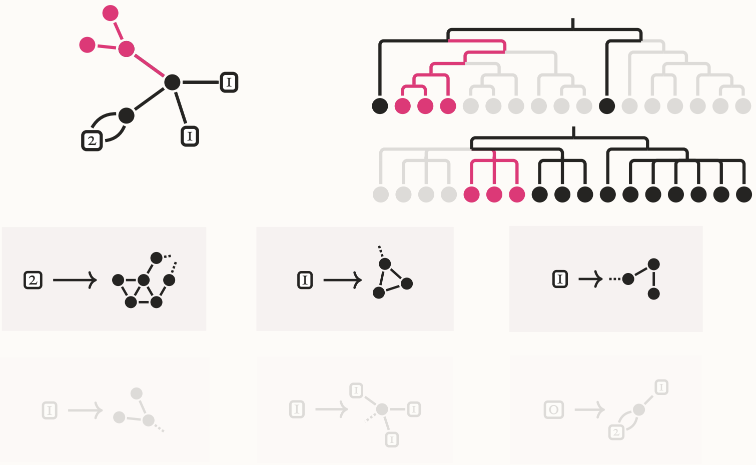


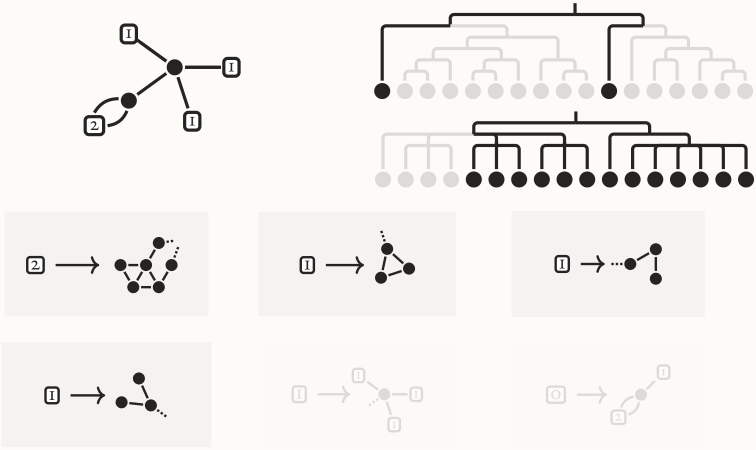

2
2
1
1
1
1
1
1
1
0
1
2

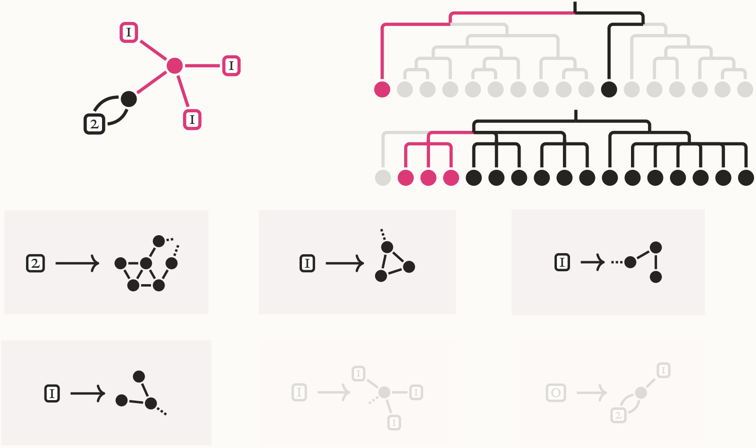

2
1
2
1
1
1
1
1
1
1
0
1
2

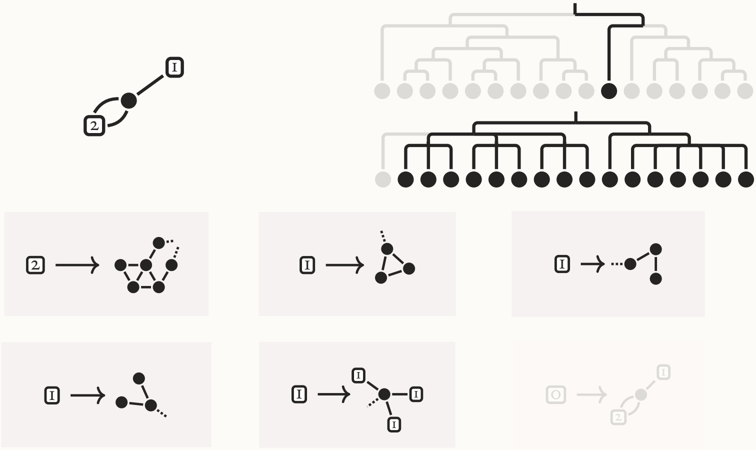


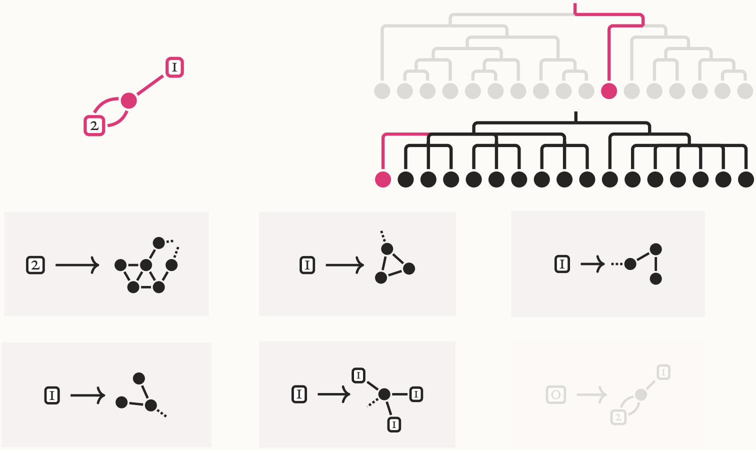


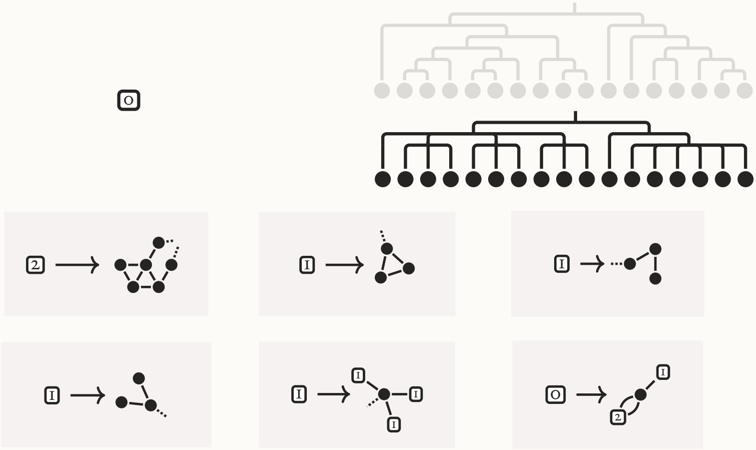








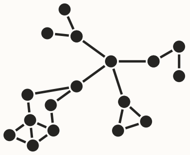




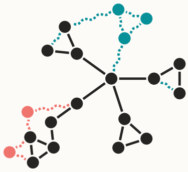
Rules are *static*, but real data is *dynamic*.

How do we make vertex replacement grammars more insightful and accurate when a graph changes?

A *temporal graph* in discrete time is a sequence of graphs $\langle G_1, \dots G_T \rangle$.



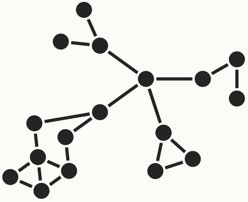
(a) Time $t = 1$

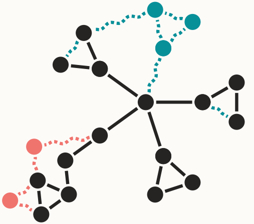


(b) Time $t = 2$

(c) Time $t = 3$

Dynamically Updating Rules





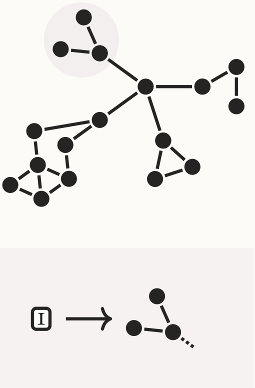
Time $t = 1$

Time $t = 2$

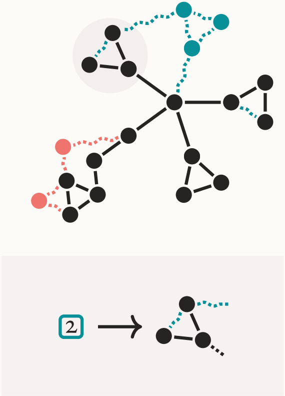
The *edit distance* between these rules is 3.

edge added

edge deleted



Time $t = 1$

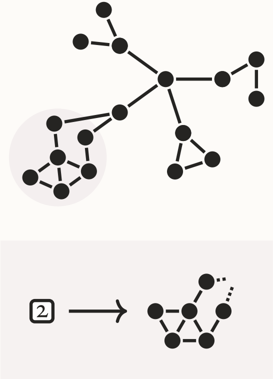


Time $t = 2$

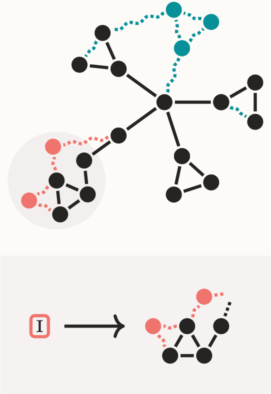
The *edit distance* between these rules is 3.

edge added

edge deleted



Time $t = 1$



Time $t = 2$

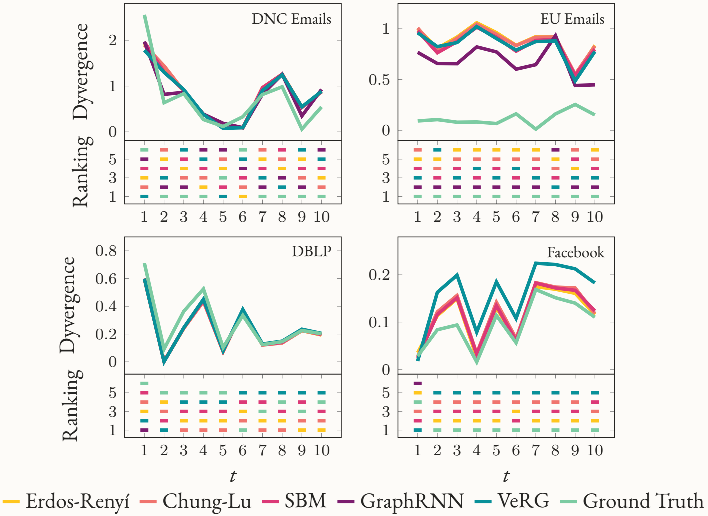
The *edit distance* between these rules is 7.

edge added

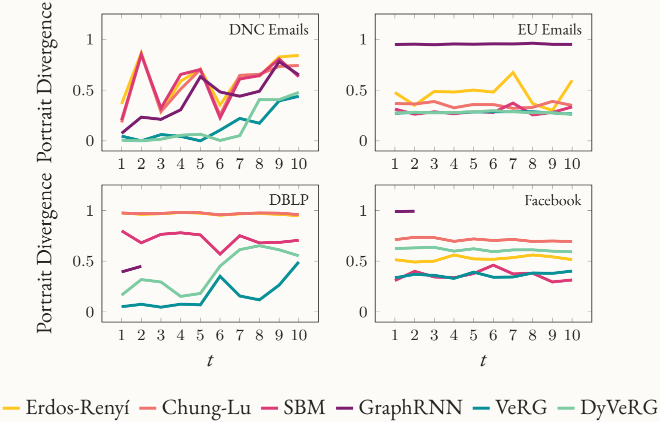
edge deleted

We introduce the grammar edit distance for measuring temporal change.

Averaging this quantity over time gives a notion of the expected amount of change in a temporal graph.

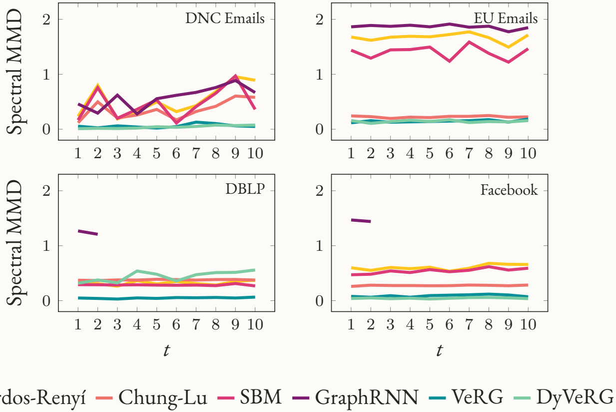


Measuring deviation from the historical trend. Lower is better.

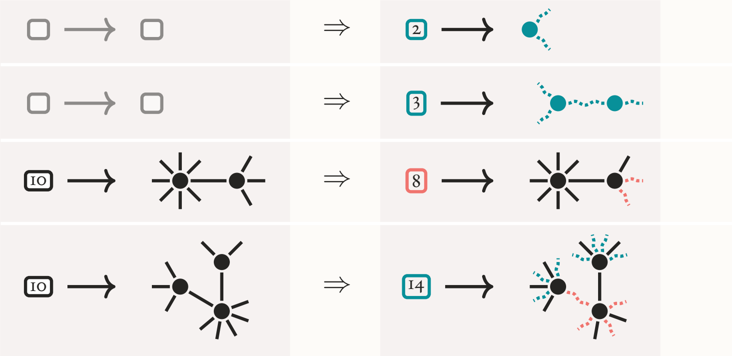


Measuring dissimilarity of generated graphs. Lower is better.

Results: Spectral MMD



Measuring dissimilarity of generated graphs. Lower is better.

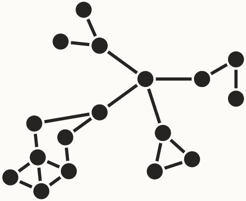


Some of the most frequent rule transitions from the EU Emails dataset.

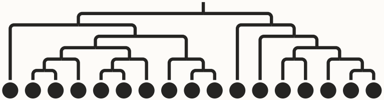
1. Vertex replacement rules *look like* transformations, but are not; we introduce *temporal rule transitions* that
    - make grammars more temporally insightful
    - provide a measurement of temporal model accuracy
2. Hierarchical clustering *constrains* rules that can be learned. How do we fix this?

1. Vertex replacement rules *look like* transformations, but are not; we introduce *temporal rule transitions* that
   - make grammars more temporally insightful
   - provide a measurement of temporal model accuracy
2. Hierarchical clustering *constrains* rules that can be learned. How do we fix this?

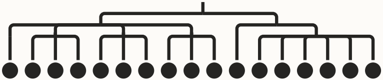
Better Priors for Graph Grammars



(a) Input graph



(b) Dendrogram determined by clustering



(c) Dendrogram determined by grammar

Better filtrations lead to better rules.

How do we learn filtrations that fit our data?

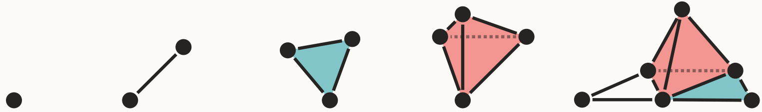
(a) $d = 0$ (b) $d = 1$ (c) $d = 2$ (d) $d = 3$ (e) $d = 3$

Some examples of simplicial complexes in various dimensions.

(a) $f(v) \leqslant \ell_1$ (b) $f(v) \leqslant \ell_2$ (c) $f(v) \leqslant \ell_3$ (d) $\ell_3 < f(v)$

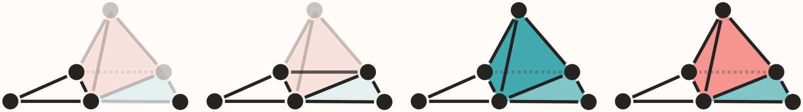
A filtration of length 4 on a simplicial complex.

Filtrations are the *level sets* of a node valuation function $f : V(G) \to \mathbb{R}$.



(a) $d = 0$ (b) $d = 1$ (c) $d = 2$ (d) $d = 3$ (e) $d = 3$

Some examples of simplicial complexes in various dimensions.

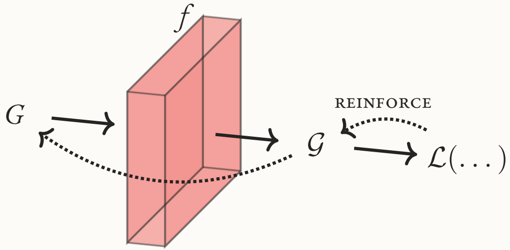


(a) $f(v) \leqslant \ell_1$ (b) $f(v) \leqslant \ell_2$ (c) $f(v) \leqslant \ell_3$ (d) $\ell_3 < f(v)$

A filtration of length 4 on a simplicial complex.

Filtrations are the *level sets* of a node valuation function $f : V(G) \to \mathbb{R}$.

We can learn $f: V(G) \to \mathbb{R}$ with a graph neural network whose loss optimizes a *relevant objective* for the data and the problem being solved.



Gradients are propagated back to the grammar $\mathcal{G}$ from the loss function $\mathcal{L}$ using the REINFORCE[7] algorithm, and backpropagation goes the rest of the way.

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

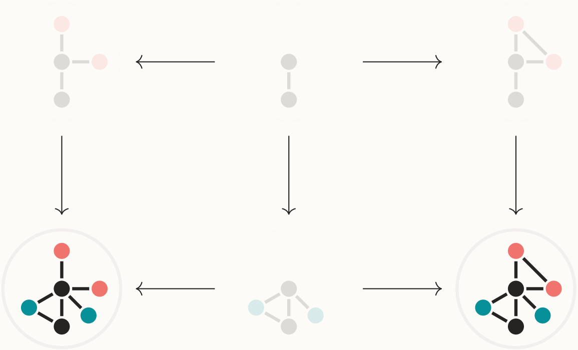
1. This will produce grammars whose rules are learned by optimizing an objective related to the data and the task.

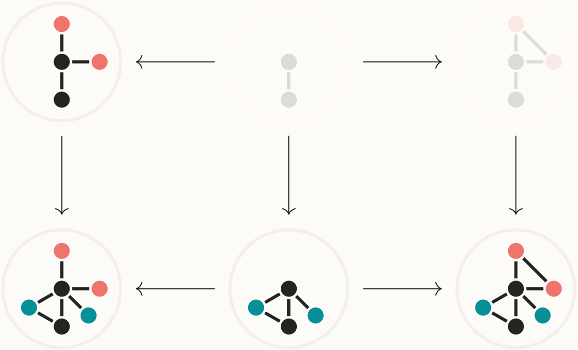
2. Grammar rules are still not *transformational*.
   How can we fix this?

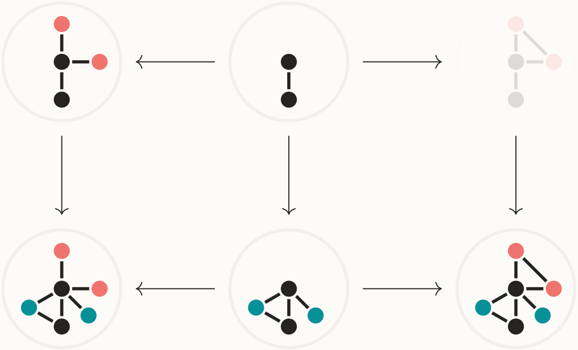
1. This will produce grammars whose rules are learned by optimizing an objective related to the data and the task.
2. Grammar rules are still not *transformational*.
   How can we fix this?

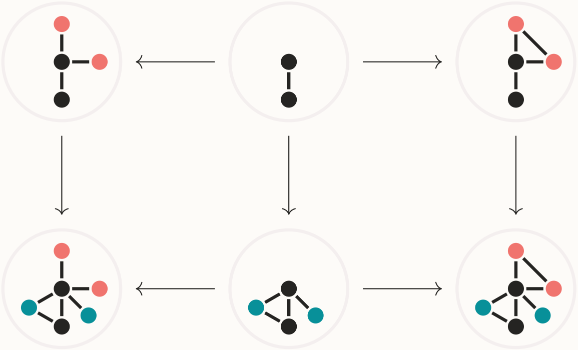
Learning Pushout Grammars

Rules should represent transformations between graph structures.



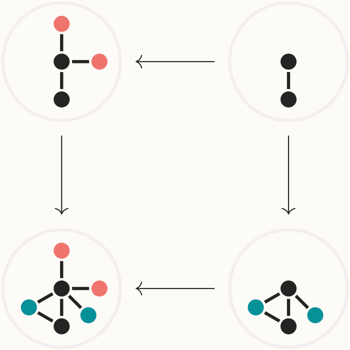






A span is a way of associating a common interface between two graphs.

A pushout describes a way of *gluing* two graphs together along the common interface described by a span.

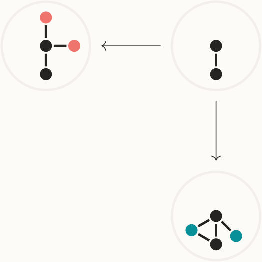


A graph homomorphism is a function $f : G \to H$ between the vertices of two graphs that preserves adjacency:

$$x \sim_G y \;\Rightarrow\; f(x) \sim_H f(y)$$

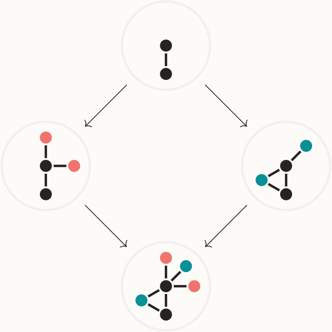
A span is a diagram consisting of three graphs A, B, C with homomorphisms between them as follows:

$$\begin{tikzcd} & C \arrow[dl] \arrow[dr] & \\ A & & B \end{tikzcd}$$



The pushout of a span is a *cospan* that is *homomorphically smaller* than any other candidate cospan

$$\begin{tikzcd}
& C \arrow[dl] \arrow[dr] & \\
A \arrow[dr] \arrow[ddr, bend right] & & B \arrow[dl] \arrow[ddl, bend left] \\
& P \arrow[d, dashed] \arrow[u, phantom, "\ulcorner" rotate=45] & \\
& Q &
\end{tikzcd}$$



A *double-pushout* diagram is one that looks like two pushout diagrams joined together

$$\begin{tikzcd} A_\ell \arrow[d] & C \arrow[l] \arrow[r] \arrow[d] & A_r \arrow[d] \\ P_\ell \arrow[ur, phantom, very near start, "\ulcorner"] & B \arrow[l] \arrow[r] & P_r \arrow[ul, phantom, very near start, "\urcorner"] \end{tikzcd}$$

In a double-pushout grammar, the rules look like spans

$$L \xleftarrow{\ell} I \xrightarrow{r} R$$

The application of a rule that transforms a subgraph H_ℓ into a new graph H_r is given by a double-pushout diagram

$$\begin{tikzcd}
L \arrow[d, "m_\ell"'] \arrow[dr, phantom, "\urcorner"] & I \arrow[l, hook', "\ell"'] \arrow[r, hook, "r"] \arrow[d] \arrow[dr, phantom, "\ulcorner"] & R \arrow[d, "m_r"] \\
H_\ell & K \arrow[l] \arrow[r] & H_r
\end{tikzcd}$$

We can learn one of these rules by working backwards from the double-pushout. If we can learn a *parallel filtration* of two graphs, we can determine the match morphisms m_ℓ and m_r, and thereby the rule spans.

Conclusion

1. Submit DyVeRG paper: before this summer.
2. Better Priors for Graph Grammars: before end of Spring 2024
3. Learning Pushout Grammars: before end of Summer 2024

Thank you!

Dynamic rule updates provide *temporal transitions* between rules based on the observed changes between the snapshots $\langle G_0, G_1, \ldots G_T \rangle$.

A Dynamic Vertex Replacement Grammar is specified by:

- a sequence of rule sets $\langle \mathcal{R}_0, \mathcal{R}_1, \ldots \mathcal{R}_T \rangle$, one for each timestep
- a set of transition functions $\pi_t : \mathcal{R}_t \to \mathcal{R}_{t+1}$ that map the rules P_t at time t to their updated forms $P_{t+1} = \pi_t(P_t)$ at time $t + 1$

Temporal generation: to grow a graph *at time t*, apply rules from $\mathcal{R}_t$.

For simplicity, we will only consider DyVeRG grammars that result from updating $\mathcal{G}_t \mapsto \mathcal{G}_{t+1}$ one timestep at-a-time.

These *temporal rule transitions* help quantify the amount of change in a graph from time t to $t+1$. We call this the deviation:

$$\Delta_{t+1} = \ln\left(1 + \sum_{P_{t+1} \in R_{t+1}} \mathrm{GED}\left(\pi_t^{-1}\left(\left\{P_{t+1}\right\}\right), P_{t+1}\right)\right) \tag{1}$$

where P_t is a rule belonging to the rule set R_t at time t

P_{t+1} is the updated form of P_t in R_{t+1}

$\pi_t : R_t \hookrightarrow R_{t+1}$ is the projection $\pi_t(P_t) = P_{t+1}$

$\mathrm{GED}\left(\dot{P}, \ddot{P}\right)$ is the edit distance between $\dot{P}$ and $\ddot{P}$

The higher the minimum number of edits required to turn $\mathcal{G}_t$ into $\mathcal{G}_{t+1}$ is, the more deviation there must have been between the graphs G_t and G_{t+1}.

These *temporal rule transitions* help quantify the amount of change in a graph from time t to $t+1$. We call this the deviation:

$$\boldsymbol{\Delta}_{t+1} = \ln\left(1 + \sum_{P_{t+1} \in R_{t+1}} \mathrm{GED}\left(\pi_t^{-1}\left(\left\{P_{t+1}\right\}\right), P_{t+1}\right)\right) \tag{1}$$

where P_t is a rule belonging to the rule set R_t at time t

P_{t+1} is the updated form of P_t in R_{t+1}

$\pi_t : R_t \hookrightarrow R_{t+1}$ is the projection $\pi_t(P_t) = P_{t+1}$

$\mathrm{GED}\left(\dot{P}, \ddot{P}\right)$ is the edit distance between $\dot{P}$ and $\ddot{P}$

The higher the minimum number of edits required to turn $\mathcal{G}_t$ into $\mathcal{G}_{t+1}$ is, the more deviation there must have been between the graphs G_t and G_{t+1}.

We estimate the expected deviation for a temporal graph by averaging the sequential deviations over time:

$$A_{t-1} = \frac{1}{t-1} \sum_{i=0}^{t-1} \Delta_i \tag{2}$$

and use these running averages to estimate the next deviation:

$$\hat{\Delta}_t = A_{t-1} + \left(\Delta_{t-1} - A_{t-2}\right) \tag{3}$$

We then measure the *dyvergence* of the timestep $t \mapsto t+1$ by comparing the deviation at that step to our historical estimate from the data:

$$\text{dyvergence}\left(G_t, G_{t+1}\right) = |\Delta_t - \hat{\Delta}_t| \tag{4}$$

To evaluate the efficacy of *dyvergence* scores for modeling a temporal dataset, we can compare the *dyvergence* of the ground-truth graph G_{t+1} to that of an impostor graph $G_{t+1}^{\mathcal{M}}$ generated by a competing model $\mathcal{M}$.

If we assign a lower *devergence* to the ground truth than to an impostor graph, then our model is correctly capturing temporal and topological features of the dataset that $\mathcal{M}$ is not taking into account.