

# Mining and Learning with Graphs *at Scale*

<https://gm-neurips-2020.github.io/>



# Welcome + Agenda

## Introduction to Graphs + Application Stories

What are graphs? Why are they important?

## Graph Mining: Basic tools and algorithms

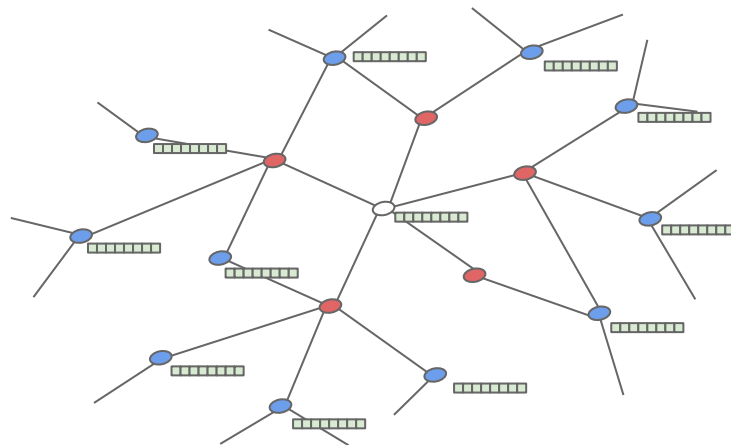
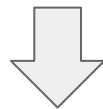
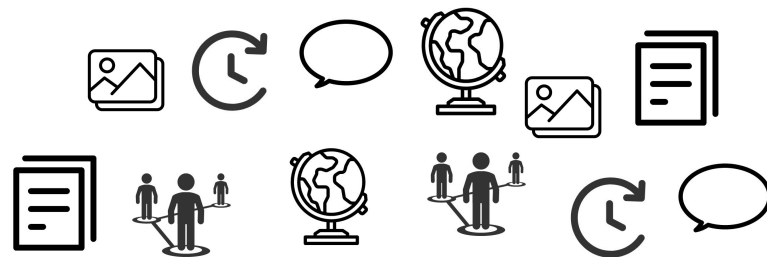
How do we build, cluster, and use graphs at scale?

## Graph Neural Networks

How can we use deep learning on graphs? How can we use graphs in deep learning?

## Systems, Algorithms and Scalability

How do we deal with massive graphs? How can graphs help us organize Google-scale data?





# An Introduction to Mining and Learning with Graphs

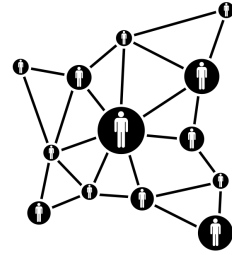
Vahab Mirrokni

# What are graphs?

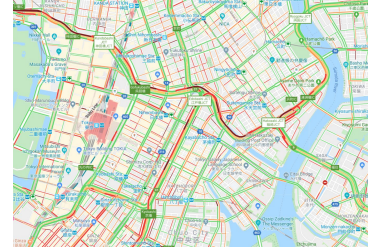
Graphs are representations of relationships (edges) between entities (nodes).

In the most general case, graphs have:

- varying numbers of edges...
- with different edge types going to different node types...
- with a highly complex structure.



Social Networks



Traffic, maps (Google Maps)

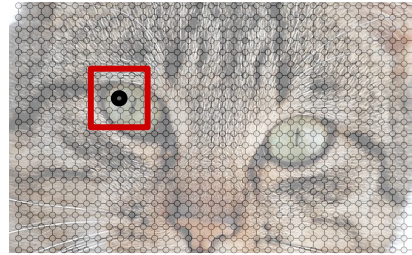
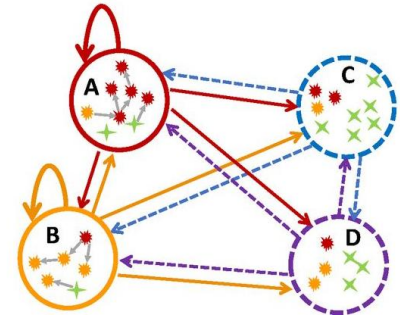


Image Pixels



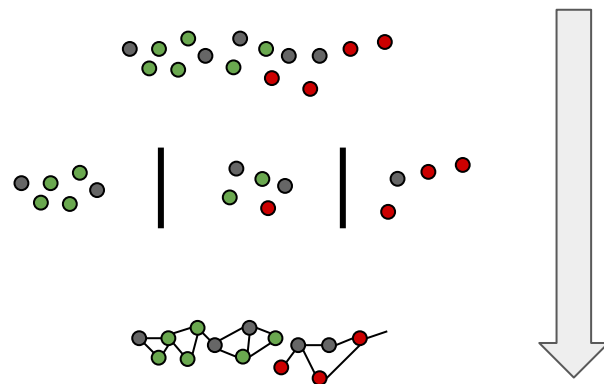
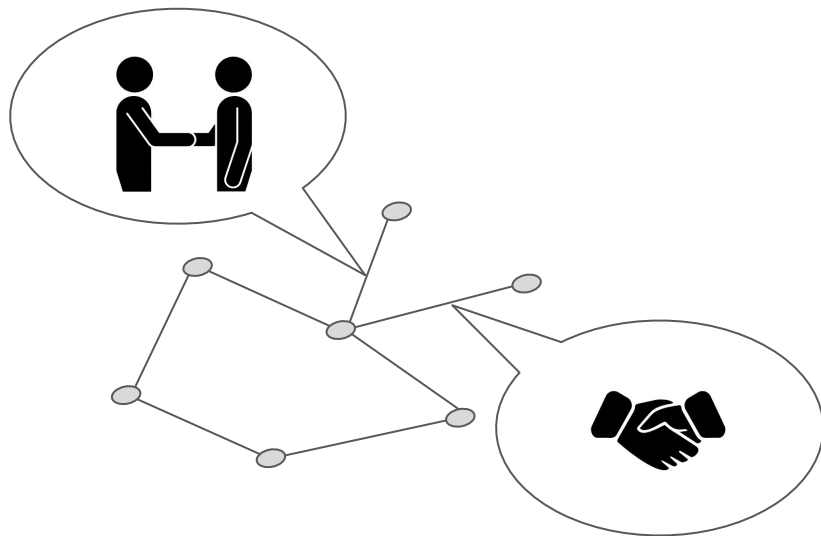
Disease Spread

<https://www.pnas.org/content/116/2/401>

# Types of Graphs

**Natural graphs** are graphs in which the edge relationship comes from an *external source*. Think: payments, social networks, roadways, coclick/cowatch.

By contrast, **similarity graphs** are graphs in which the edge relationship is based on some measure of similarity/distance between nodes. In these cases, we start with a blob of (meta-)data and attempt to give that blob structure via graph representation.



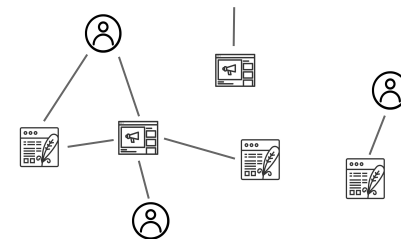
# Why Graphs?

## Computation on abstract concepts

Most data is fundamentally about relationships, and graphs can help us. Graphs can also help us abstract *local information* and use it to extract useful *global information* from data.

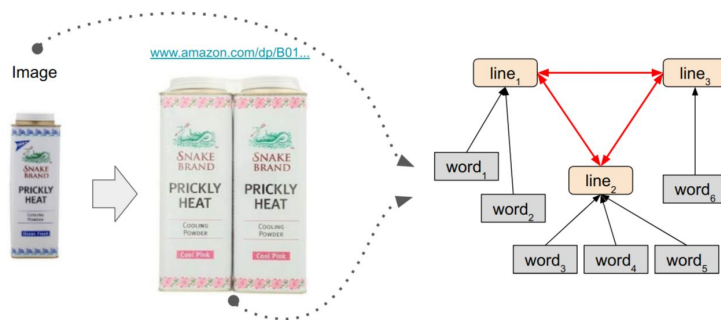


Social Network Analysis, Wikimedia Commons



## Computation on different data types

We constantly deal with visual, textual, and semantic information, and all of this data relates to each other. Graphs provide a natural way to handle *multi-modal data*.



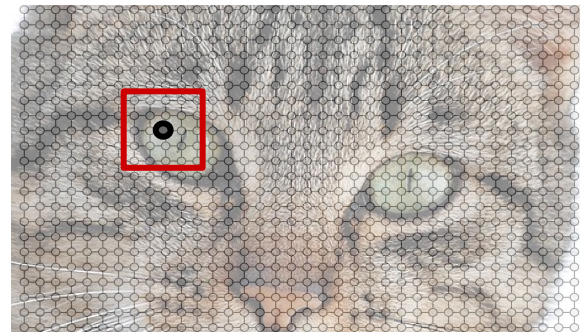
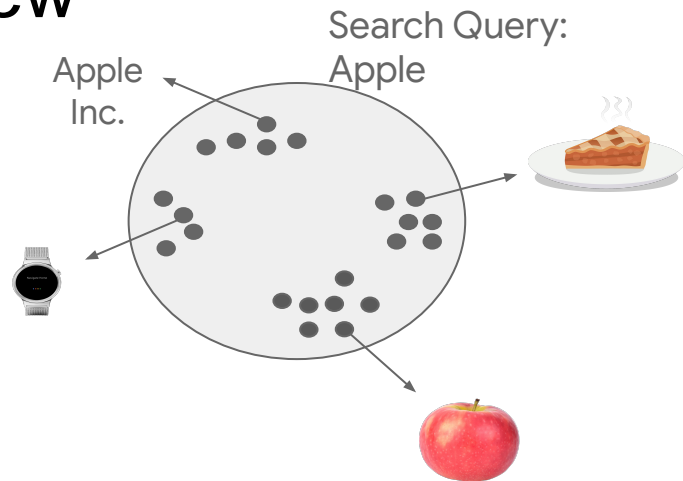
# Why Graphs? Global and Local View

## Global view:

Graph structure/topology can tell us a lot about our data such as uncovering clusters of data points, or providing distance measures for otherwise intangible concepts.

## Local view:

Local edges to and from a node can tell us something *useful* about a node -- something that is difficult to express with a single element.



*The black center pixel is part of an eye, but that is only apparent when you can see nearby pixels.*

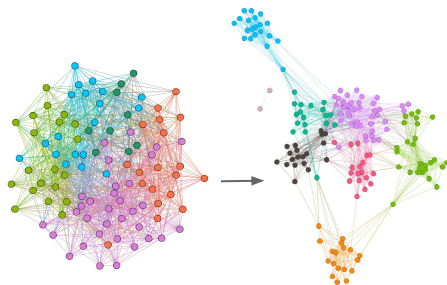
# Graphs at Scale: Algorithms, Learning, & Systems for Impact

Because graph representations are so flexible, we often want to use them on Google-scale data.

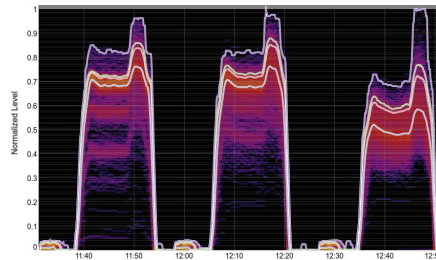
We are often dealing with billions of nodes and many more edges. To work with data at this scale, we have to combine algorithmic ideas with the right systems and ML models.

*This can be very hard, and the devil is in details.*

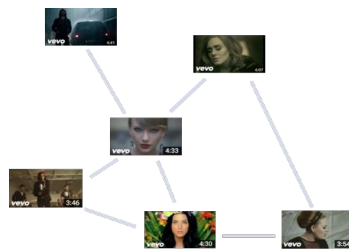
These tools power hundreds of projects at Google in Search, Ads, Youtube, Play, Cloud, Maps, Payments, and more.



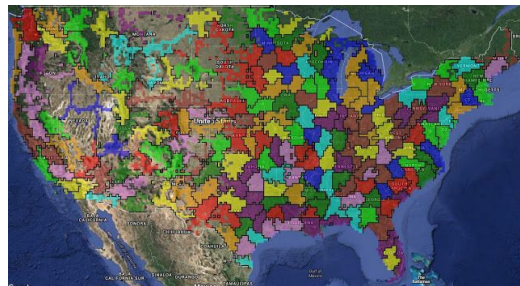
Same-meaning queries for Keyword matching systems



Better Caching for saving 32% Flash I/O for Search Infra([VLDB'19](#)).



Collaborative Filtering for YouTube Recommendations



Finding micro-markets in designing A/B experiments ([KDD'19](#), [NeurIPS'19](#))



# A bit of History: Graph Mining Team

<https://research.google/teams/algorithms-optimization/graph-mining/>

**Mission:** Develop the most scalable & reliable *graph-based mining and learning library*, and make it universally accessible (XT edges)

Started ~11 years ago from scalable graph mining → graph-based learning and graph neural networks.

**Team Skills:** Algorithms, Systems, and ML. Research + Engineering.

**Publish in a variety of venues:** NeurIPS, ICML, SODA, FOCS, STOC, VLDB, KDD, WWW, WSDM, AAI, and ...

**Lessons Learned:**

*Algorithms+System Research:* Important to combine right algorithms and distributed systems

- Tried Pregel first, but then it was not suitable for some large-scale applications (e.g., fault-tolerance)
- Then, we built infrastructure on top of MapReduce/Flume and Distributed Hash Table Service (DHT).
- Had to rethink the systems for Tensorflow and GNN training via Graph Sampling...

**More popular tools vs. less commonly tools in our library:**

- Widely Used: Graph Building and Clustering, Semi-supervised Learning, GNNs and Embedding.
- Less Used: Shortest Paths, Matchings, Graph Similarity, Graph-based centrality scores.

# Computation Frameworks: System+Algorithms+ML

Many popular frameworks for big data/ML analysis:

- MapReduce / Hadoop [DeanG, OSDI'04]
- Pregel / Giraph [e.g., MalewiczABDHLC, SIGMOD'10]
- TensorFlow/Pytorch/Keras [e.g., Abadi et al]
- Beam / Flume / Cloud Dataflow [e.g., AkidauBCC+, VLDB'15]

Our library has four main parts:

1. **Distributed Algorithms:** Mapreduce/Flume/DHT
  - graphs with XT edges in hours
2. **Multi-core In-memory:** GBBS graph-based
  - XXB (XXM\*) edges in minutes (seconds\*)
3. **GraphTensor:** Graph Neural Networks in Tensorflow
  - Graph analysis integrated w/ deep learning tools
4. **Dynamic Graph Mining** (*not covered here*)
  - Handling online requests very fast



# The Graph Mining Toolbox: Overview

## Graph Building

Nearest Neighbors,  
Locally Sensitive Hashing,  
Local Neighborhood Search,  
Graph Learning

## Clustering

Hierarchical Clustering,  
Connected Components  
Semi-supervised Clustering,  
Community Detection

## Topology Analysis and Similarity Ranking

Personalized PageRank,  
ego-Net Mining,  
Sampling like Coverage,

## Information Propagation

Label Propagation,  
Cluster Propagation,  
Iterative Classification,  
Semi-supervised Learning

## Graph Signals

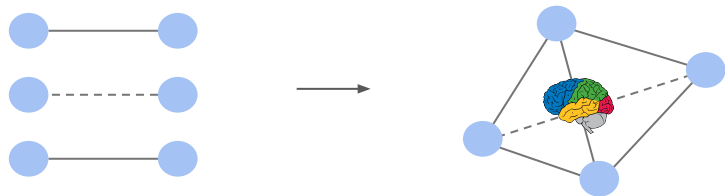
Local Density, PageRank,  
Centrality, Oddball Score,  
Clustering Coefficients,  
Graph Embeddings

## Graph Neural Networks

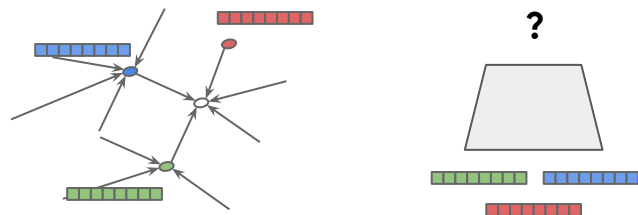
GNNs: Graph Convolutions,  
Message-Passing Neural Nets,  
Neural-based Embedding

# Tools for Learning with Graphs

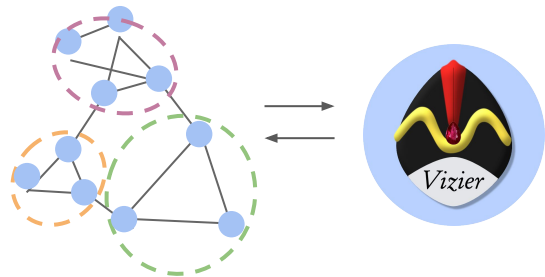
Grale (Learning Graphs)  
Learn graph structure from data.



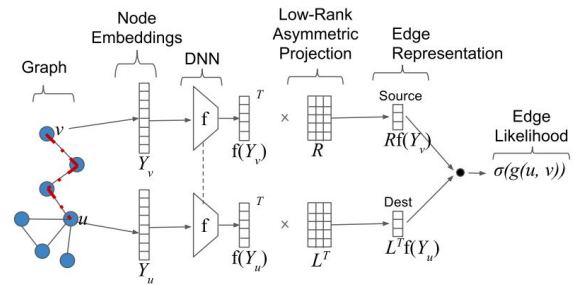
Graph-based Semi-supervised Learning (SSL)  
Learn a label propagation function from training data.



Semi-supervised Clustering  
Optimized Clustering for labeled training data



Neural Graph Embedding & Graph Convolutions  
Apply deep learning to arbitrary graph structures

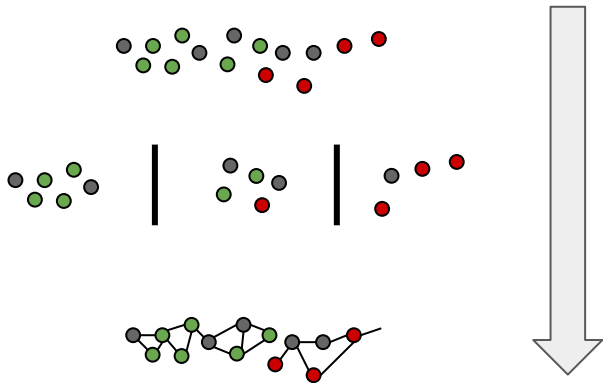


# The Graph Mining Toolbox

## Graph Building and Graph Learning

Graph Building answers two questions: what is the optimal graph for a given dataset; and how can we create that graph in a scalable way.

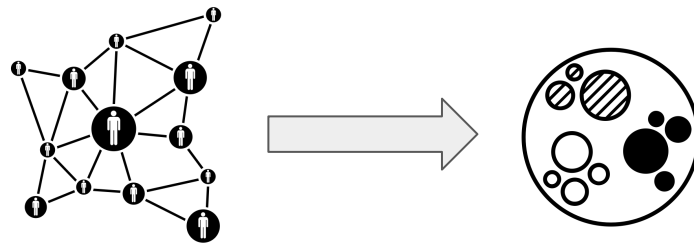
*Techniques:* Locally sensitive hashing(LSH), Local Search, Auto-encoders, ...



## Clustering

Clustering tools allow us to identify important patterns in data, aka clusters.

*Techniques:* LSH/sketching, random walks, message-passing, Composable Core-sets for hierarchical, overlapping, spectral, balanced clustering, ...

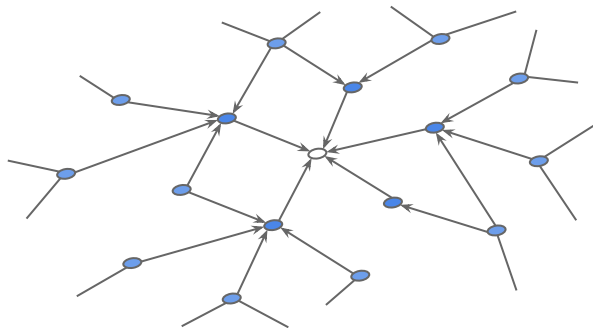


# The Graph Mining Toolbox

## Information Propagation & graph-based SSL

Information sparsity is a common problem in big data. We use graphs for semi-supervised learning(SSL) to spread information predicting missing data and correcting misinformation.

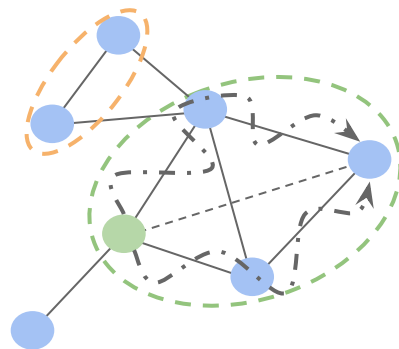
*Techniques: graph-based semi-supervised learning, spectral theory, iterative classification.*



## Graph Signals and Topology Analysis

Graph structure allows computing multi-hop similarity and graph signals, e.g., egoNets and Personalized PageRank. *In a multimodal world*, we can use graph information to extract useful signals, e.g., edge density and graph embedding.

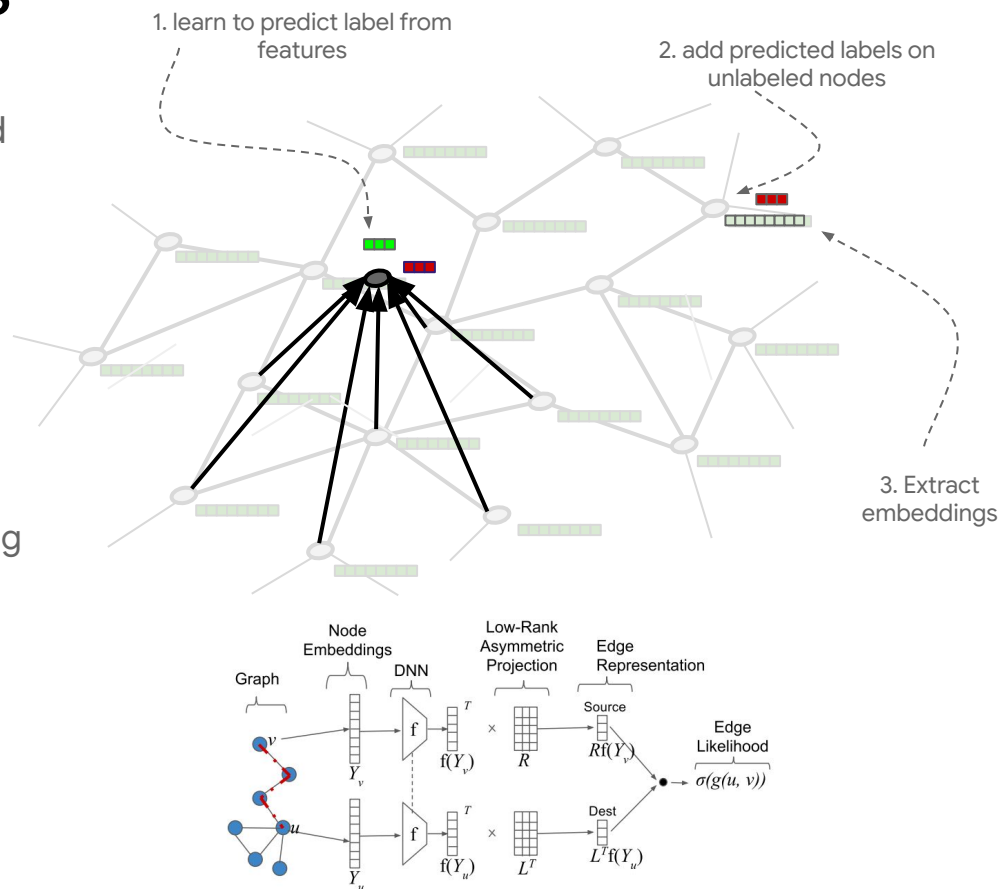
*Techniques: random walks, clustering, embedding.*



# Graph Neural Networks

Advances in deep learning have helped us build and deploy novel graph building and label propagation techniques. We've also been developing a scalable Graph Convolutional Network system that promises to completely upend how we think about graph data.

*Techniques:* Graph Convolutions, Message-Passing Neural Nets, Neural-based Embedding, Graph Attention Models, PPR for GNNs, Self-supervised Learning.



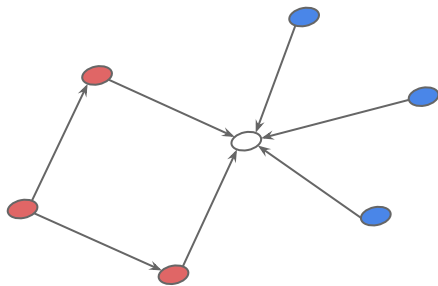
# Canonical Uses: Spam, Fraud and Abuse Detection

Graph Mining tools let us tackle Trust&Safety problems in many ways.

Preventing spam, fraud, and abuse is central for many products, e.g., YouTube and Ads.

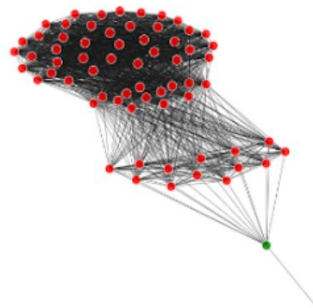
## Label Propagation

Core intuition: start with known bad actors, and use the graph structure to identify nearby neighbors that may also be suspicious.



## Anomaly Detection via Density Clustering

Core intuition: statistically unlikely dense clusters correlate highly with malicious behavior.





# Canonical Uses: Improving ML Models


## Relationship Discovery

Ever wonder how Social Networks find “People you may know”? The famous ‘social graph’ is represented as a real graph. We can use graph information to discover connections that may not appear naturally.



Image size:  
881 × 657

Find other sizes of this image:  
All sizes - Small - Medium - Large

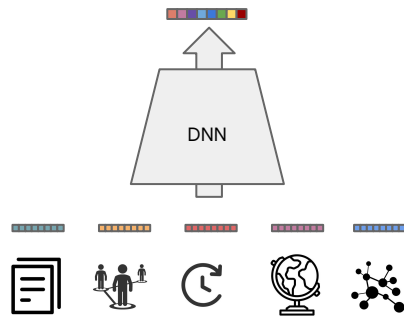
 Visually similar images



Google Images ‘Visually Similar Images’ is powered in part through Graph Mining technologies.

## Feature Extraction

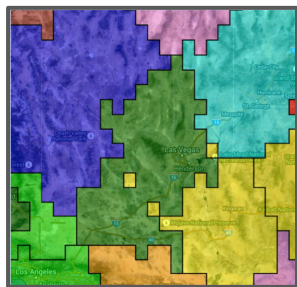
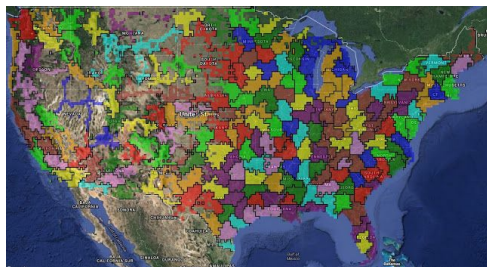
Generated graph signals like clusters, PPR vectors, and graph embeddings are useful as training signals to upstream ML models. In multi-modal models, graph data can be seen as another modality, part of the larger whole.



# Canonical Uses: Efficient Computing

## Resource Efficiency: Communication Overhead

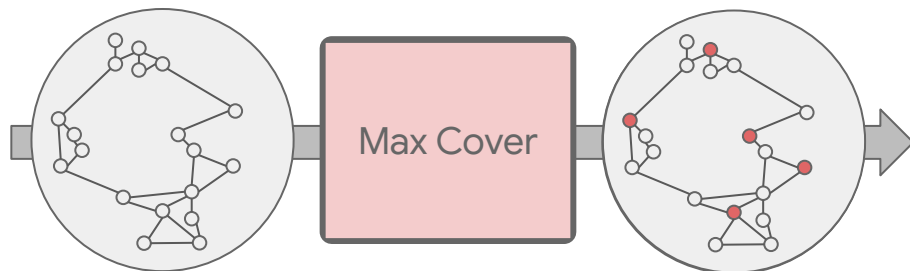
We can use graph partitioning algorithms like balanced partitioning to intelligently split large datasets, reducing overhead for distributed computing, e.g., in Google Driving Directions.



Graph clustering is used by Maps to help optimize backend of driving directions [[WSDM'16](#)].

## Data Efficiency and Active Learning

We can utilize graphs to answer queries like ‘what are the most diverse points in my dataset’, which can power active learning loops. *Graph-based semi-supervised learning* can also be used in data-sparse models with less data.



Graph-based clustering [[NeurIPS'17](#)] and coverage [[KDD'18](#)] is applied for active learning and feature engineering.

# The end of the beginning

The Graph Mining team combines research with application to create a powerful suite of tools. We're excited to share what we've been working on!

Hopefully, the last twenty minutes or so have provided a useful backdrop for why graph-based learning at scale is so important.

During the rest of this workshop, you'll hear from experts in the field discussing our work in much more depth, with a general focus on distributed algorithms and scalable graph-based learning.



GoogleAI Graph Mining

<https://research.google/teams/algorithms-optimization/graph-mining/>

Part of Algorithms and Optimization

<https://research.google/teams/algorithms-optimization/>

# Presenters



## GraphAI Graph Mining

[Graph Mining – Google Research](#)

Part of [Algorithms & Optimization – Google Research](#)



Vahab Mirrokni



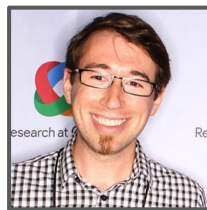
Alessandro Epasto



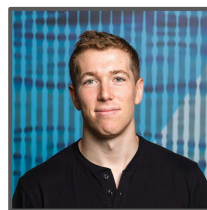
Allan Heydon



Amol Kapoor



Bryan Perozzi



Jean Pouget-Abadie



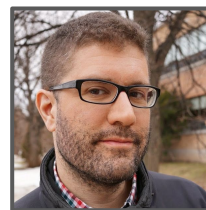
John Palowitch



Jakub Łącki



Jonathan Halcrow



Martin Blais

André Linhares,  
Andrew Tomkins,  
Arjun Gopalan  
Ashkan Fard,  
CJ Carey,  
David Eisenstat,  
Dustin Zelle,  
Filipe Almeida,  
Hossein Esfandiari,

Kevin Aydin,  
Jason Lee,  
Matthew Fahrbach,  
MohammadHossein Bateni,  
Nikos Parotsidis,  
Reah Miyara,  
Sam Ruth,  
Silvio Lattanzi,  
Warren Schudy, and  
many collaborators

# Rest of the Workshop

**-Applications:** Covid Forecasting, Privacy, Causal Inference.

**-Graph Mining: Basic tools and algorithms**

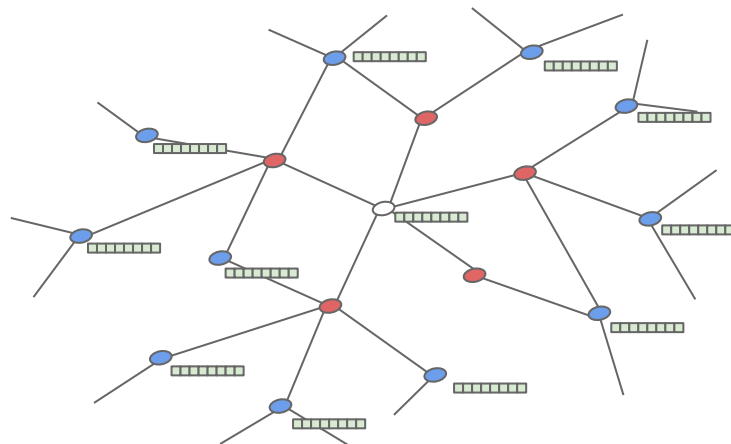
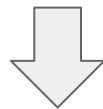
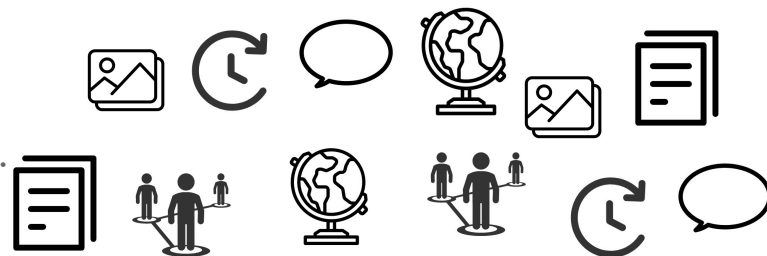
How do we learn, cluster, and use graphs at scale? Graph Learning, Similarity Ranking, Clustering, and Label Smearing.

**-Graph Neural Networks**

How can we use deep learning on graphs? How can we use graphs in deep learning?

**-Algorithms, Systems and Scalability**

How do we deal with massive graphs and use them to organize Google-scale data? TensorFlow, Flume, Multi-core.



# Citations

(excluding papers - which will be covered later)

## ICONS:

Document: <https://thenounproject.com/search/?q=documents&i=3594373>

Forward: <https://thenounproject.com/search/?q=time+forward&i=2596961>

Globe: <https://thenounproject.com/search/?q=globe&i=3119957>

Image: <https://thenounproject.com/search/?q=images&i=3593232>

Network: <https://thenounproject.com/search/?q=network&i=1350199>

Talk Bubble: <https://thenounproject.com/search/?q=talk+bubble&i=842574>

Handshake: <https://thenounproject.com/search/?q=handshake&i=983923>

Handshake: <https://thenounproject.com/search/?q=handshake&i=3592892>

Publisher: <https://thenounproject.com/search/?q=publisher&i=3048742>

Advertise: <https://thenounproject.com/search/?q=advertiser&i=2374780>

Account: <https://thenounproject.com/search/?q=account&i=1931153>

Network: <https://thenounproject.com/term/network/54119/>

Clustering: <https://thenounproject.com/search/?q=clusters&i=195949>

Network: <https://thenounproject.com/search/?q=network&i=1061260>

## GRAPHS

Disease Spread: <https://www.pnas.org/content/116/2/401>

### Social Network Analysis:

[https://commons.wikimedia.org/wiki/File:Social\\_Network\\_Analysis\\_Visualization.png](https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png)

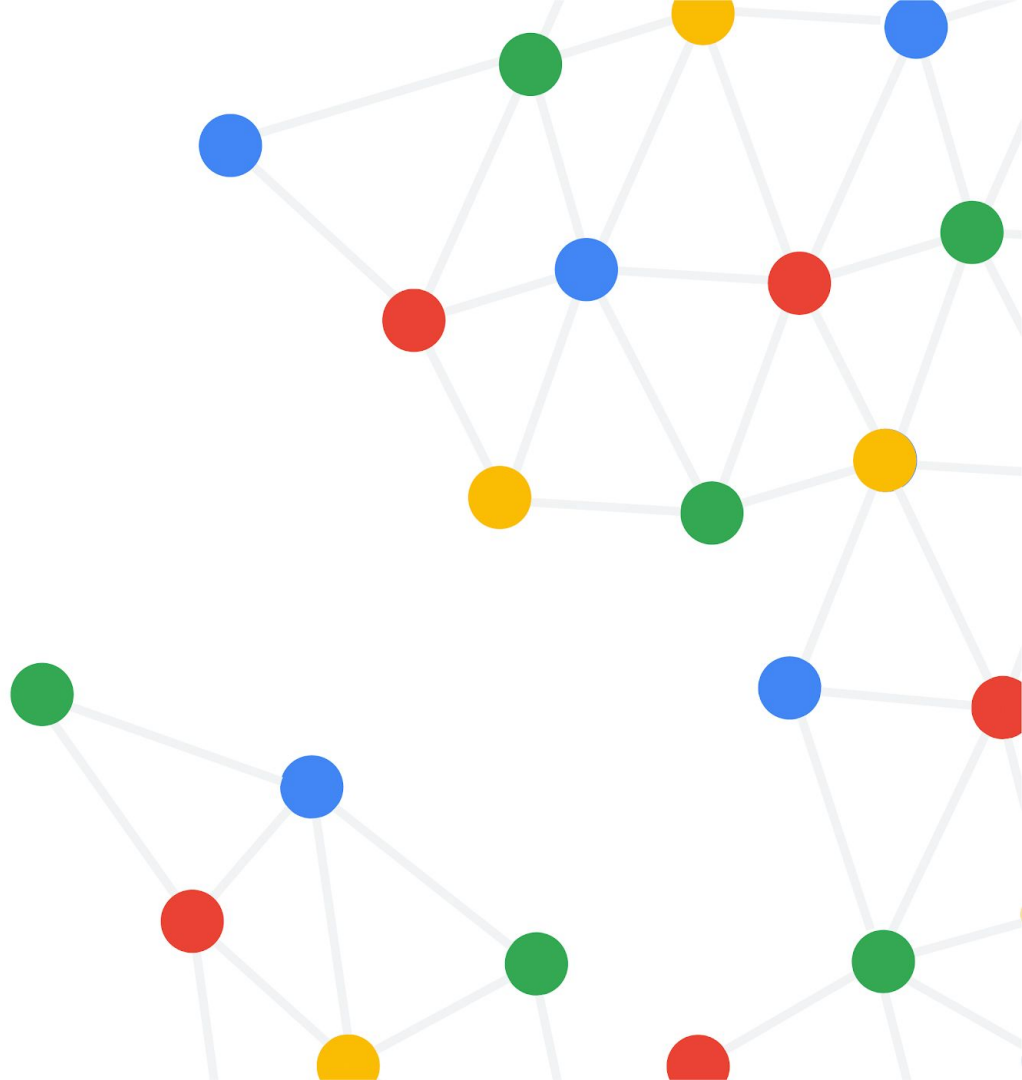
## OTHER:

Cat: [https://commons.wikimedia.org/wiki/File:Cat\\_March\\_2010-1.jpg](https://commons.wikimedia.org/wiki/File:Cat_March_2010-1.jpg)

Analyze Love: <https://xkcd.com/601/>

CC3.0: <https://creativecommons.org/licenses/by/3.0/us/legalcode>

CC2.5: <https://creativecommons.org/licenses/by-nc/2.5/>



Up next, we'll dive into a few  
Application Stories of Graph-based  
Learning, starting with Amol Kapoor  
discussing GNNs and COVID.

# Application Stories

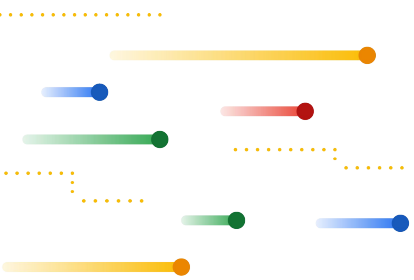
Amol Kapoor, Alessandro  
Epasto, Jean Pouget-Abadie

Modelling COVID with GNNs

Privacy

Experimental Design and  
Causal Inference





# Modeling COVID with Spatio-Temporal Graph Neural Networks

Amol Kapoor

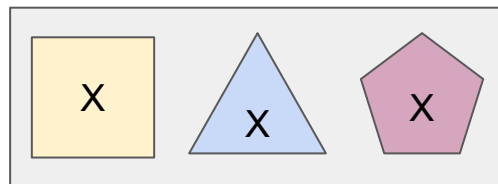


# The Basics



# The Basics

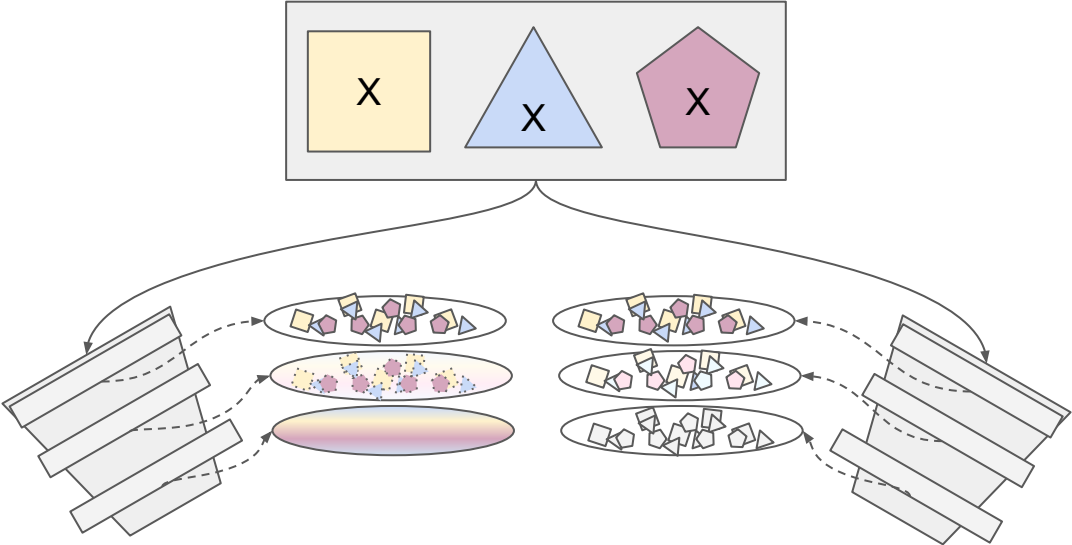
Deep ML models learn a function  $f(x)$ ,  
where  $x$  is some (curated) feature set.



# The Basics

Deep ML models learn a function  $f(x)$ , where  $x$  is some (curated) feature set.

Intermediate states -- embeddings -- in Deep ML models capture complex interactions between features in high dimensional space.

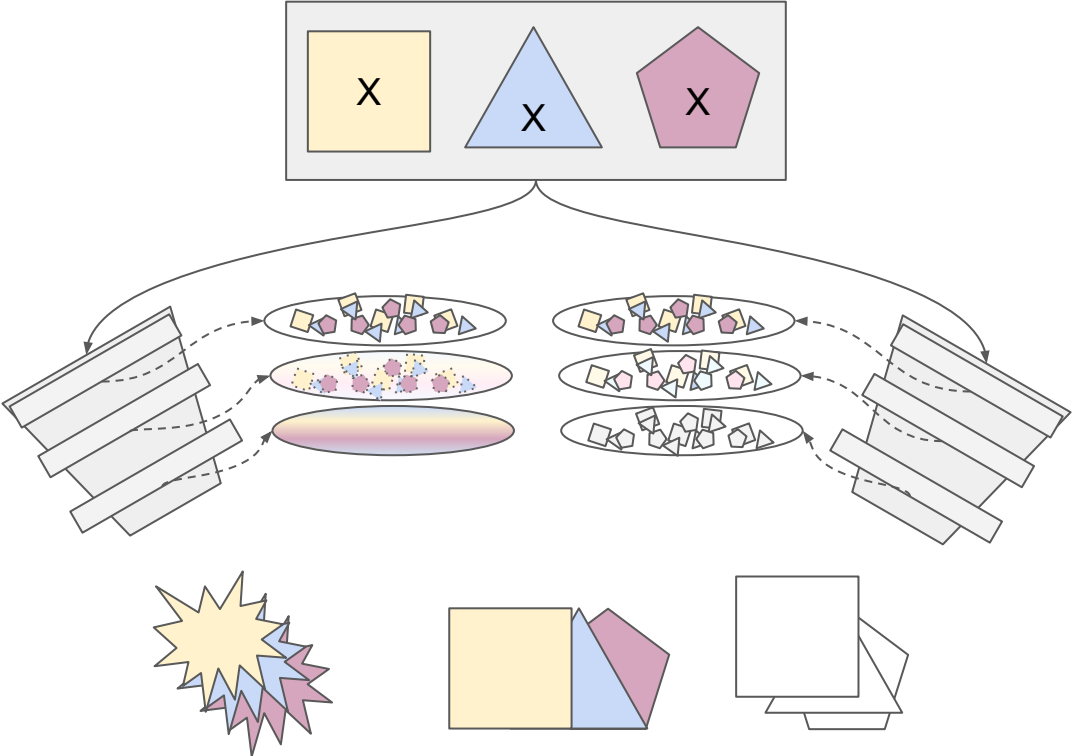


# The Basics

Deep ML models learn a function  $f(x)$ , where  $x$  is some (curated) feature set.

Intermediate states -- embeddings -- in Deep ML models capture complex interactions between features in high dimensional space.

Deep ML models are optimized for some loss, which in turn defines how each intermediate embedding is structured.

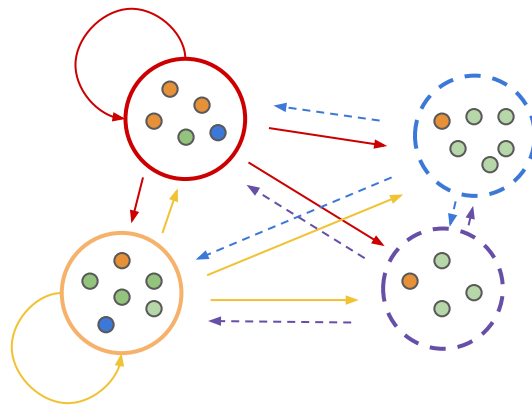
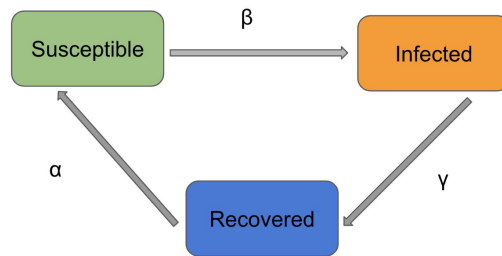


Deep ML models are powerful  
because you can put *anything* on the  
ends, and the intermediate state will  
fill in the blanks.

# DL in Epidemiology

SIR Models modelling disease spread by constraining the relationship between three groups: Susceptible, Infected, and Recovered. Accurately identifying these groupings (and the transition functions between them) is extremely difficult.

This is where deep learning comes in. DL shows ability in processing complex disease dynamics and multidimensional data that cannot be captured by traditional compartmental models and statistical models.





# Modelling COVID

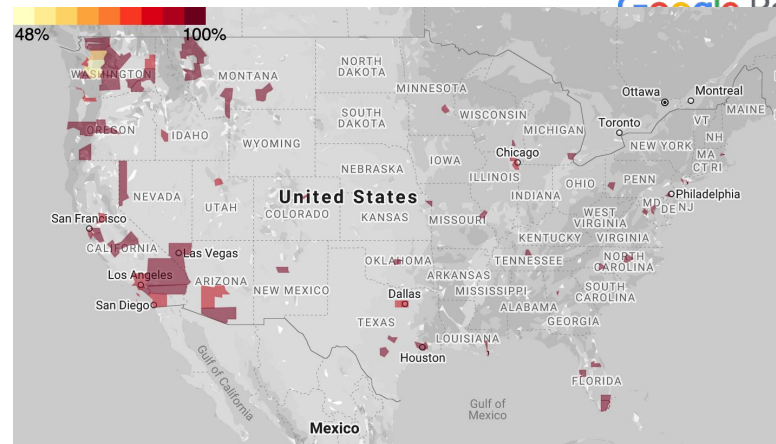


Intuition: epidemiological modelling depends on  
time and space. The number of cases I have  
tomorrow is a function of the cases I had yesterday,  
and the cases of my neighbors today.  
This is a multimodal problem.

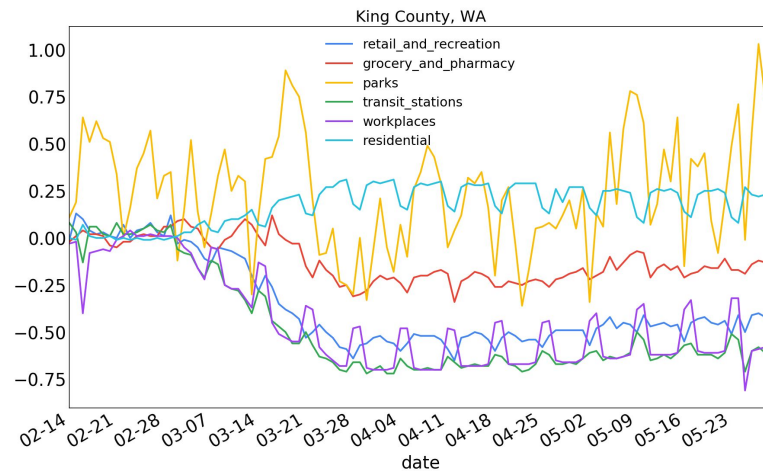
# Mobility Data

Can utilize mobility data to create temporal and spatial edges between nodes, in order to understand how people (and, by extension, COVID) move around.

Google has rich mobility info through aggregated GPS analysis. This allows us to answer questions like ‘how many people flew from King County, Washington, to Queens, NY’, or ‘how many people in LA used the subway today’.



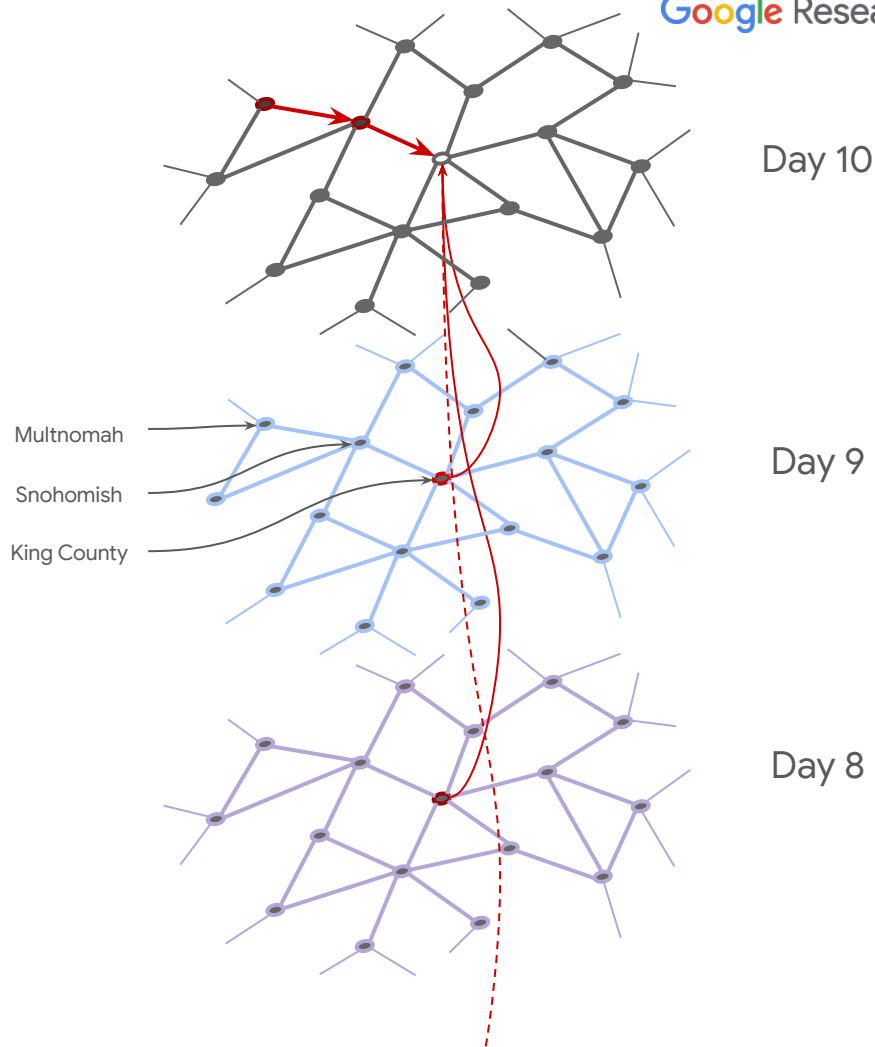
Top: Inter-county mobility data from King County.  
Bottom: Intra-county mobility data from King County.



# Modelling COVID

Using NYT COVID report data and Google mobility data at US county level, we created a spatio-temporal graph. Each node was a time + place, and had case counts and intra-mobility data as self-features.

The graph can be modeled as 150 slices. Edges within each slice are spatial, and are weighted based on mobility. Edges between slices are temporal, and are (inversely) weighted based on the amount of time passed between the edge.

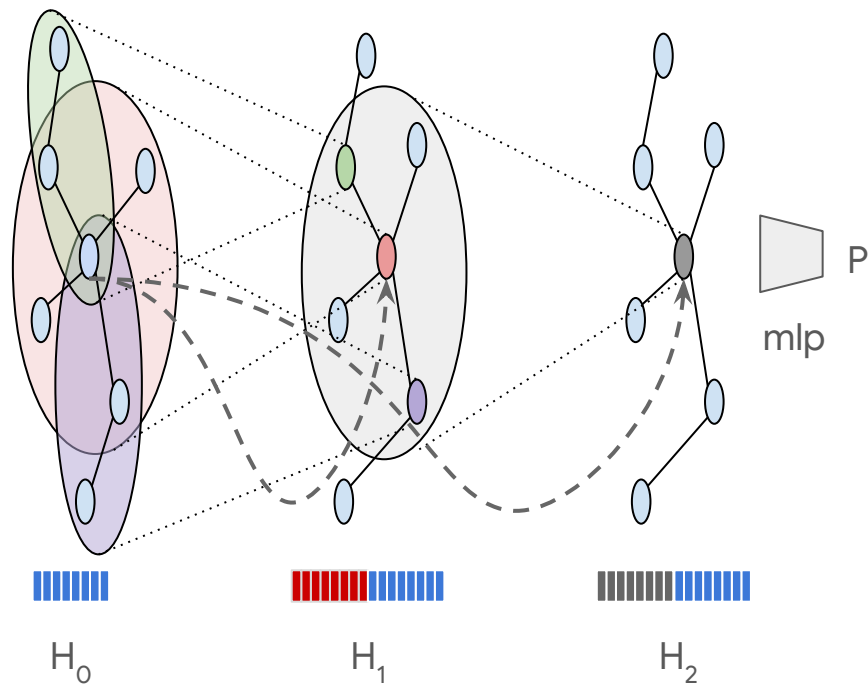


# Why GCNs?

One of the greatest benefits of graph data is that we can incorporate context into our analysis. When analyzing a node, we can surface its neighborhood as a source of information.

GCNs supercharge this principle by applying deep learning on top. We can use a GCN to build a learned hierarchical representation around a given node, allowing us to pull in contextual information that will help us predict node level features.

Like, say, COVID case counts.



# (Initial) Results

Initial results show that the GNN is able to successfully use the mobility data to better predict next-day-change in COVID caseload.

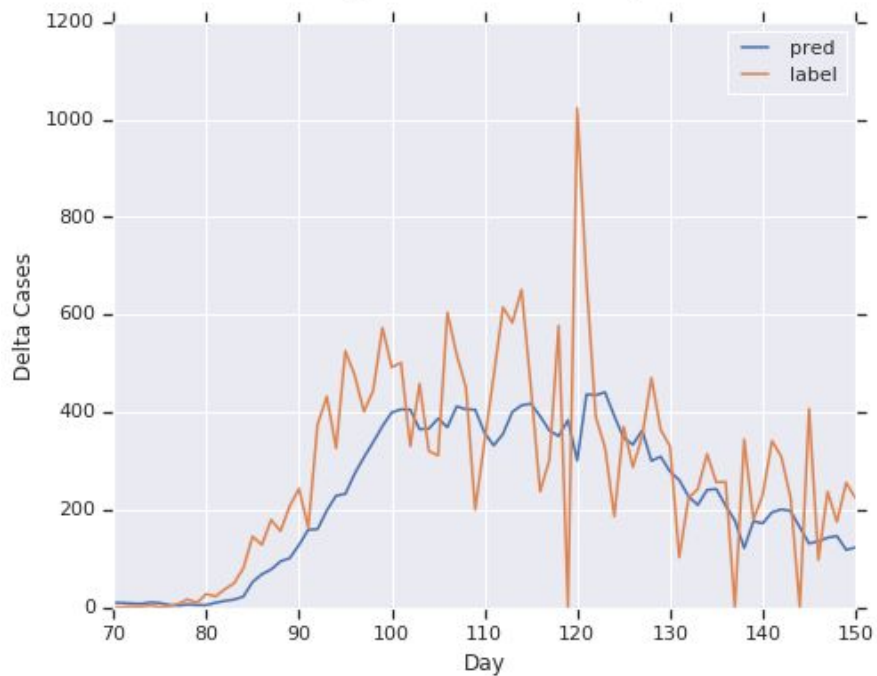
With virtually no hyperparameter tuning or feature engineering, we achieve as significant reduction in error on RMSLE and Pearson Correlation.

Model	RMSLE	Corr	$\Delta$ RMSLE	$\Delta$ Corr
Previous Cases	0.0226	<b>0.9981</b>	4.7879	NaN
Previous Delta	0.0127	0.9965	0.9697	0.1854
No Mob ARIMA	0.0124	0.9968	0.9217	0.1449
ARIMA	0.0144	0.9952	0.9624	0.0966
No Mob LSTM	0.0125	0.9978	0.9172	0.1540
LSTM	0.0121	0.9978	0.9163	0.1863
No Mob Seq2Seq	0.0118	0.9976	<u>0.8467</u>	0.1020
Seq2Seq	0.0116	0.9977	0.8634	<b>0.2802</b>
GNN	<b>0.0109</b>	<u>0.9980</u>	<b>0.7983</b>	<u>0.2230</u>

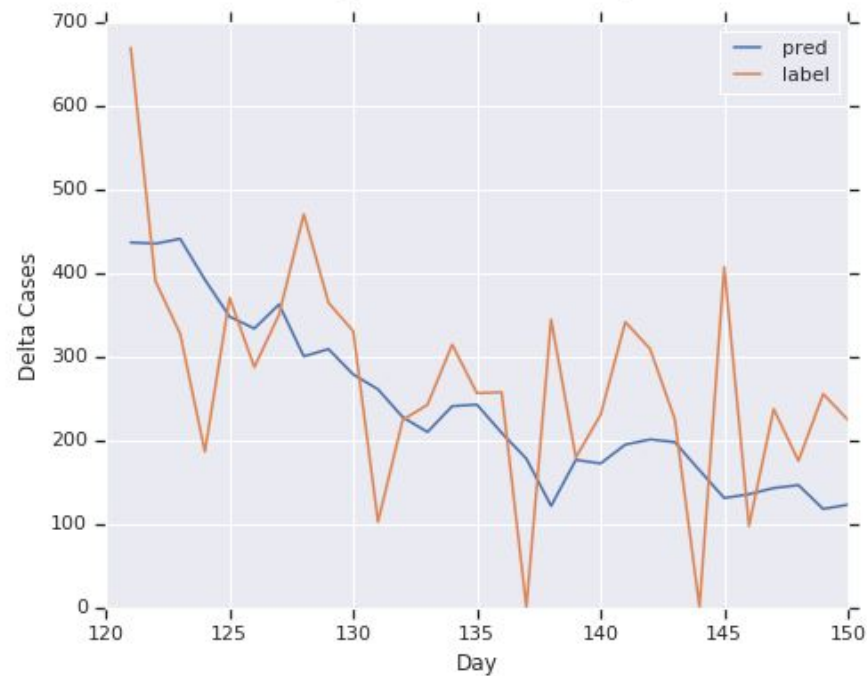
**Table 1: Summary of model performances.**

# (Initial) Results

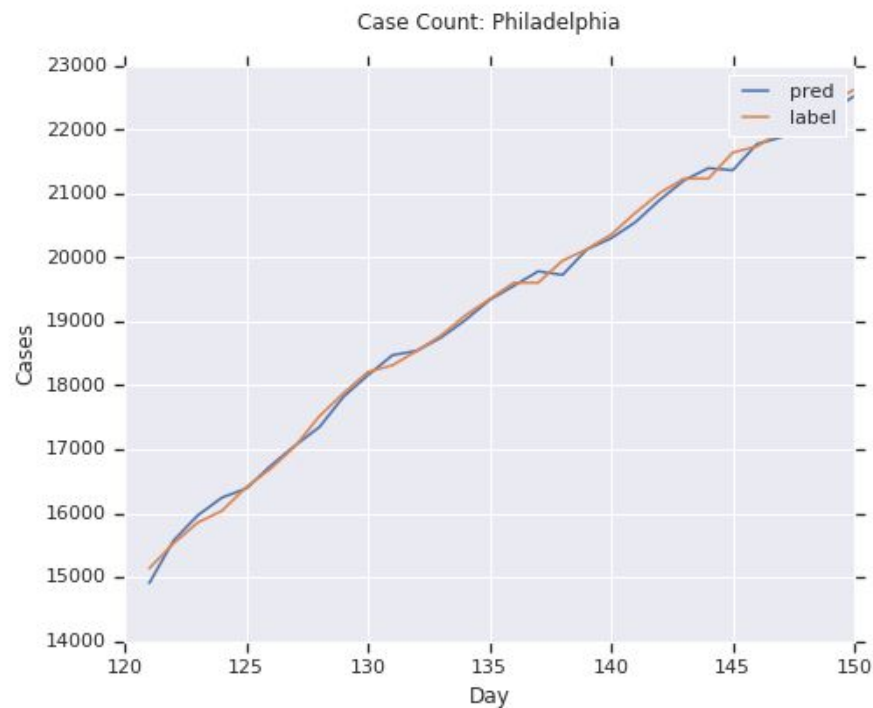
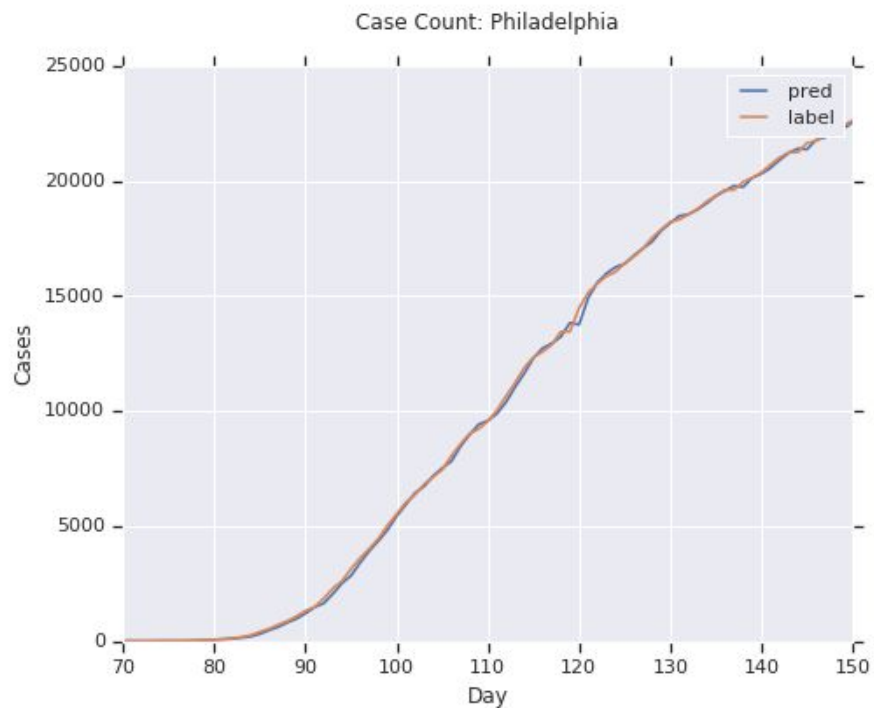
Change in Case Count: Philadelphia



Change in Case Count: Philadelphia



# (Initial) Results





# Conclusions





Deep ML models are powerful because they can take arbitrary inputs and learn mappings to requested outputs.

Graphs provide a means to incorporate *context*, which is a powerful source of information. GCNs build this context into a unifying deep-ml framework.

GCNs can be used to model COVID (and other things!),  
and represent a powerful tool-in-the-toolbox to tackle  
all sorts of epidemiological problems.

# Citations

## Based on Work By:

Amol Kapoor, Xue Ben, Luyang Liu, Bryan Perozzi, Matt Barnes, Martin Blais, Shawn O'Banion.

**Examining COVID-19 Forecasting using Spatio-Temporal Graph Neural Networks.**

MLG'20, epiDAMIK'20. <https://arxiv.org/abs/2007.03113>

New York Times. **Coronavirus (Covid-19) Data in the United States.**

<https://github.com/nytimes/covid-19-data>

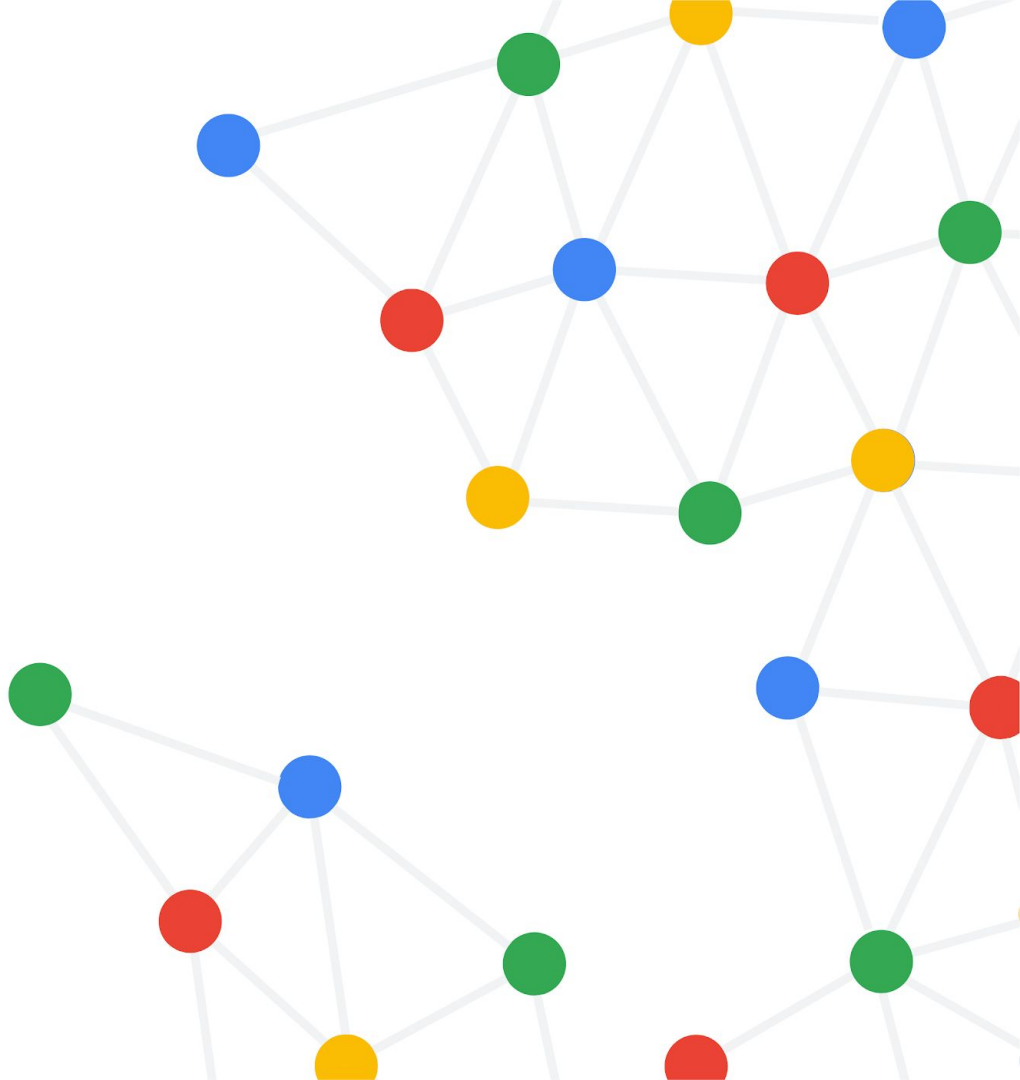
## ICONS:

Mind: <https://thenounproject.com/search/?q=deep+learning+graph&i=1705433>

## OTHER:

CC3.0: <https://creativecommons.org/licenses/by/3.0/us/legalcode>

CC2.5: <https://creativecommons.org/licenses/by-nc/2.5/>





# Application Story: Privacy

Alessandro Epasto

# Privacy for Graphs, Graphs for Privacy

Privacy is a fundamental concern in the analysis of user data and graphs are no exception.

Two application stories for privacy in graphs:

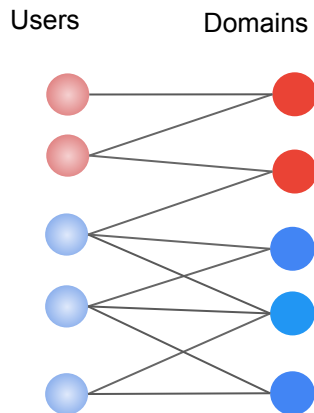
1. Can we use graph mining (graph clustering) to improve user privacy?
2. How can we protect the users' privacy in a social network application?

# FLoC -- Federated Learning of Cohorts

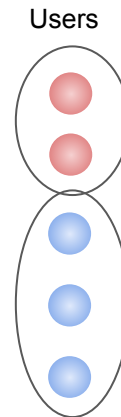
As part of the Chrome Privacy Sandbox effort to deprecate the use of third-party cookies.

FLoC aim at replacing identifying third-party cookies with anonymous cookies *shared by many users*.

**Input:** User x Browsing history data.



**Output:** FLoCs with  $\geq k$  users with similar browsing interests

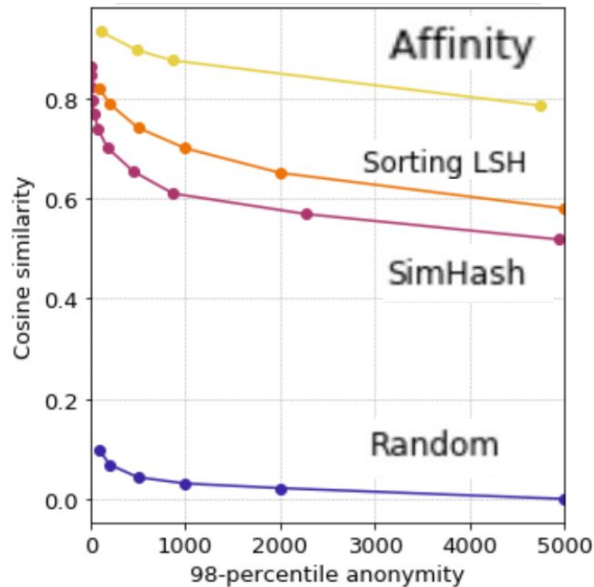


# Clustering for Privacy

The essence of the FLoC is a **size-constrained clustering** problem where clusters must respect minimum sizes.

We evaluated many clustering algorithms including Hierarchical Graph Clustering algorithms like Affinity (discussed in a later talk).

Affinity outperformed all variants tested in our experiment reported in the public FLoC white paper.



Results on the public MSD dataset.

More details: [bit.ly/3ngKcrK](https://bit.ly/3ngKcrK)

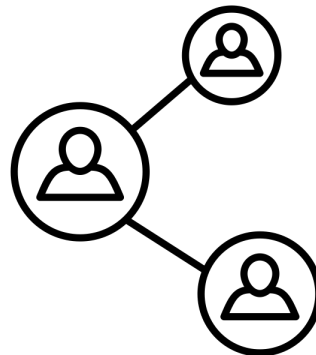


# On-Device Public-Private Graph Model

Consider a social-network-based recommender system.

Can the user receive suggestions based on their social contact, **without sharing** their **local private device data** or **even their private social contacts** with the central recommendation system?

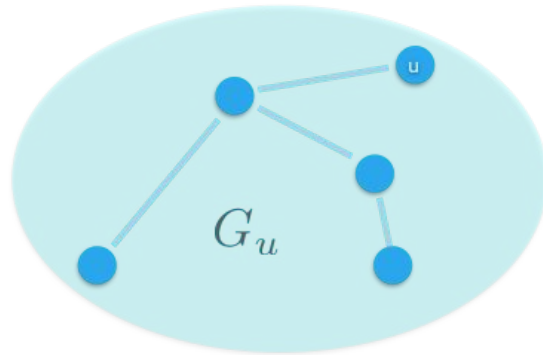
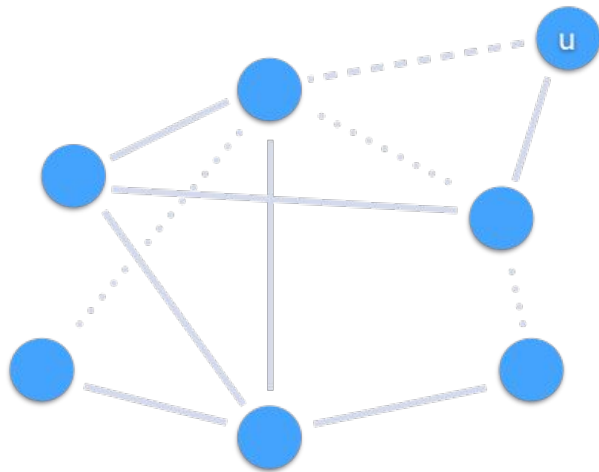
Based on work Chierichetti, Epasto Kumar, Lattanzi, Mirrokni, KDD'15 (best paper award) and Epasto, Esfandiari, Mirrokni, WWW'19.



Created by Med Marki  
from Noun Project

# Public-Private Graphs

There is a public graph  $G$  in addition every node  $u$  has access to a local graph  $G_u$



# On Device Public-Private Computation

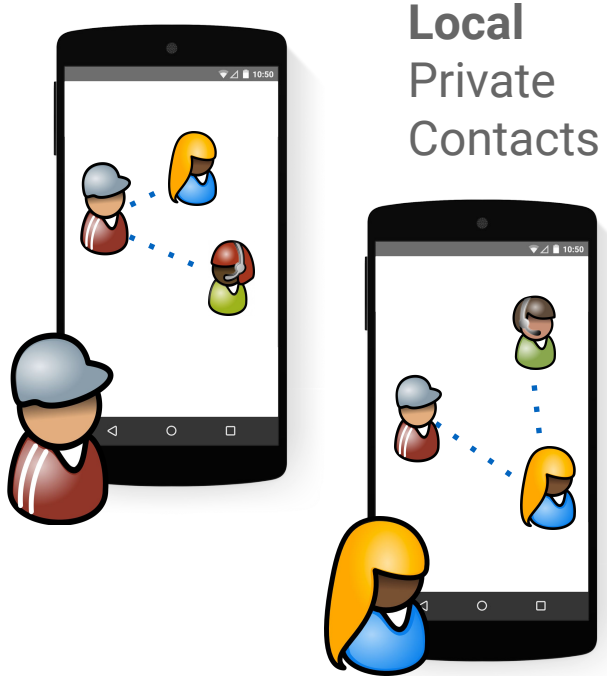
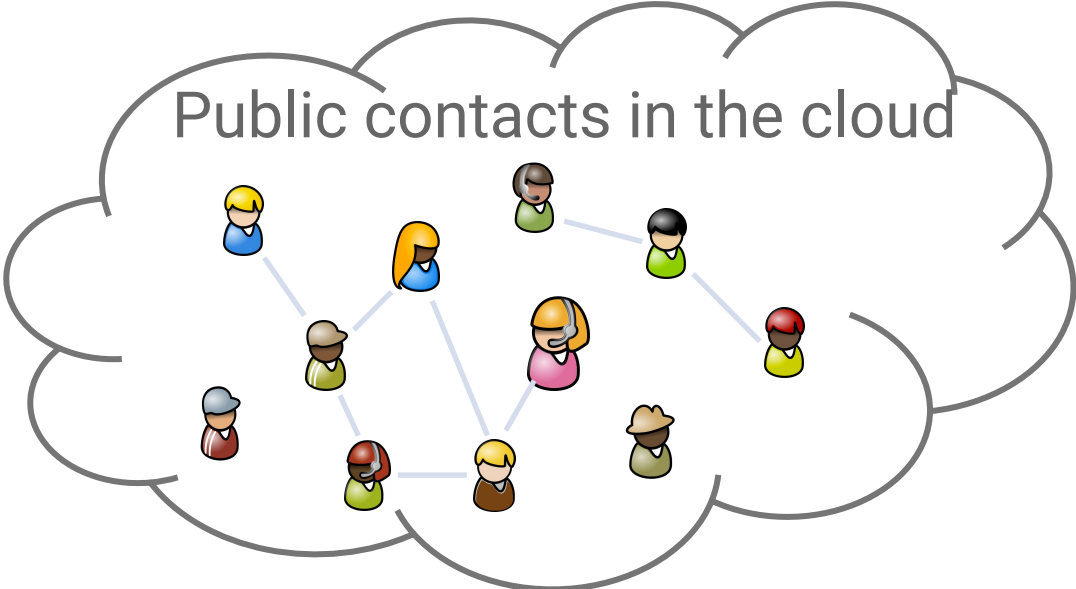
Can we keep all private data and private contacts on the users' devices and solve important machine learning problems without any privacy loss?

**Yes!** We provide algorithms for the following problems:

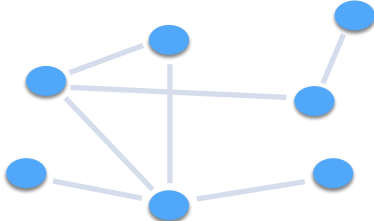
- K-clustering (k-center, k-means);
- Personalized social-network based recommendations: heavy hitters, linear suggestions.

*On-Device Algorithms for Public-Private Data with Absolute Privacy*, Epasto, Esfandiari, Mirrokni, WWW'19

# Data Separation



# Two Step Process



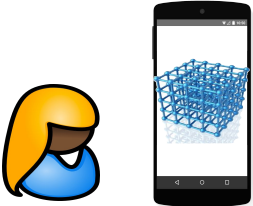
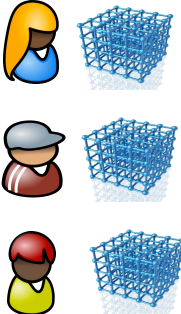
Public Graph

Preprocessed by the cloud.

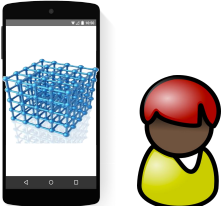
Sends the sketches to the individual users.



Synopsis of public data



Private Contacts exchange their public sketches



# Thank you for your attention!

Please check out our next sessions.

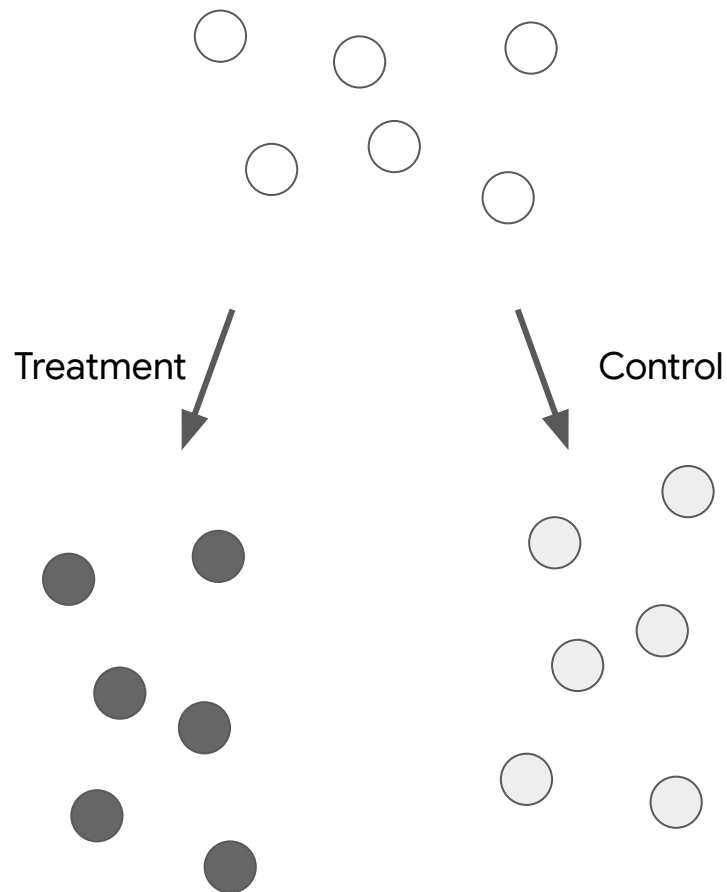


# Clustering and Causal Inference

Jean Pouget-Abadie

# What is causal inference?

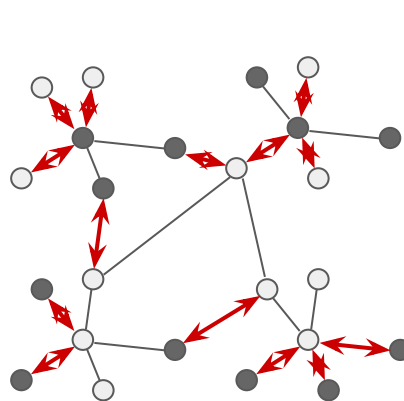
- Causal Inference is a branch of statistics that tries to establish the link between cause and effect.
- Randomized trials (e.g., clinical trials, A/B tests) assign units (e.g., patients, users) to a treatment condition or control condition.



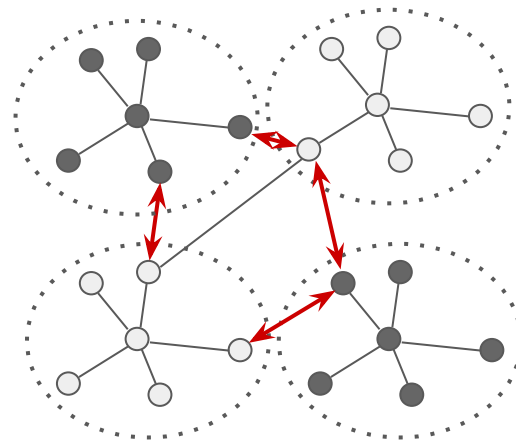


# Where does clustering come in?

- Randomized trials can suffer from interference if the treatment of one unit affects another.
- To place units in conditions as close to the “all treated” and “all control” world, cluster-randomized trials assign units to treatment/control in clusters.



Randomized Assignment



Cluster randomized trial

○ Control unit

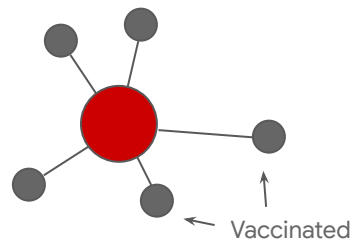
● Treated unit

↔ Treatment-control interaction

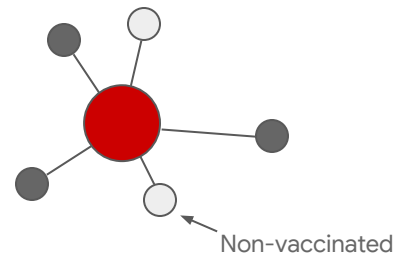
# Example 1: vaccination trials

- **Community clustering** has been studied for vaccination trials. Such a clustering can be as simple as individual vs household [1].

A unit surrounded by vaccinated units is less likely to get sick than...



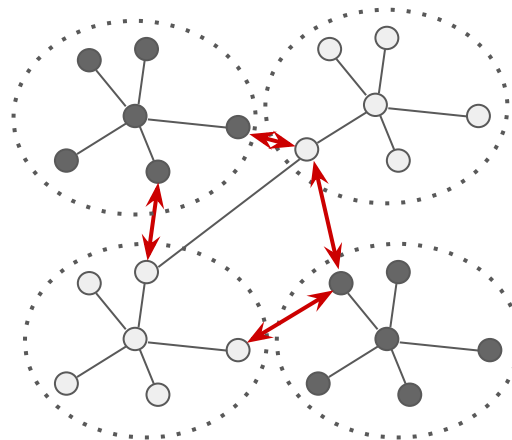
... a unit partially surrounded by vaccinated units.



[1] Datta, Susmita, M. Elizabeth Halloran, and Ira M. Longini Jr. "Efficiency of estimating vaccine efficacy for susceptibility and infectiousness: randomization by individual versus household." *Biometrics* 55.3 (1999): 792-798.

## Example 2: social networks

- **Balanced partitioning** is a popular method to split a social network into buckets and avoid too many interactions between the edges of a social network [1, 2].



[1] Eckles, Dean, Brian Karrer, and Johan Ugander. "Design and analysis of experiments in networks: Reducing bias from interference." *Journal of Causal Inference* 5.1 (2016).

[2] Gui, Huan, Ya Xu, Anmol Bhasin, and Jiawei Han. "Network a/b testing: From sampling to estimation." In Proceedings of the 24th International Conference on World Wide Web, pp. 399-409. 2015.

Word of mouth can easily spread through a social network

## Example 3: online marketplaces

- Clustering has also been studied for experimentation of online marketplaces [1, 2]
- Different methods have been used. **Geographical partitioning** [3], and more recently **correlation clustering** [4].

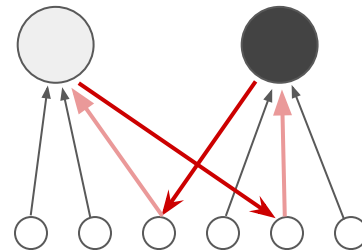
[1] Pouget-Abadie, Jean, Vahab Mirrokni, David C. Parkes, and Edoardo M. Airoldi. "Optimizing cluster-based randomized experiments under monotonicity." In KDD, pp. 2090-2099. 2018.

[2] Holtz, David, Ruben Lobel, Inessa Liskovich, and Sinan Aral. "Reducing Interference Bias in Online Marketplace Pricing Experiments." arXiv:2004.12489 (2020).

[3] Rolnick, David, Kevin Aydin, Jean Pouget-Abadie, Shahab Kamali, Vahab Mirrokni, and Amir Najmi. "Randomized Experimental Design via Geographic Clustering." In KDD, pp. 2745-2753. 2019.

[4] Pouget-Abadie, Jean, Kevin Aydin, Warren Schudy, Kay Brodersen, and Vahab Mirrokni. "Variance Reduction in Bipartite Experiments through Correlation Clustering." In NeurIPS, pp. 13309-13319. 2019.

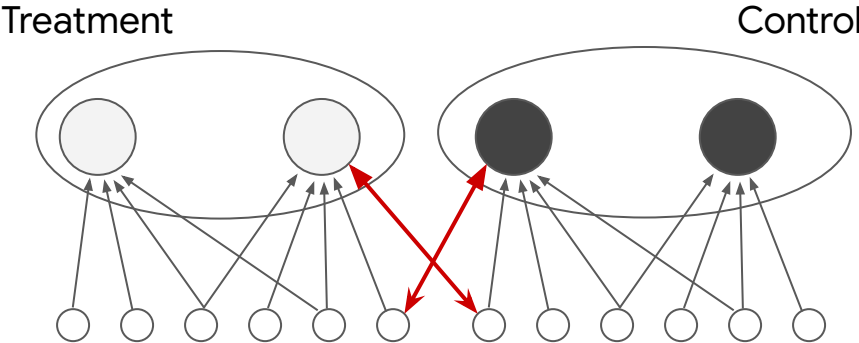
Buyers in online marketplaces compete with one another. Changing one market cluster..



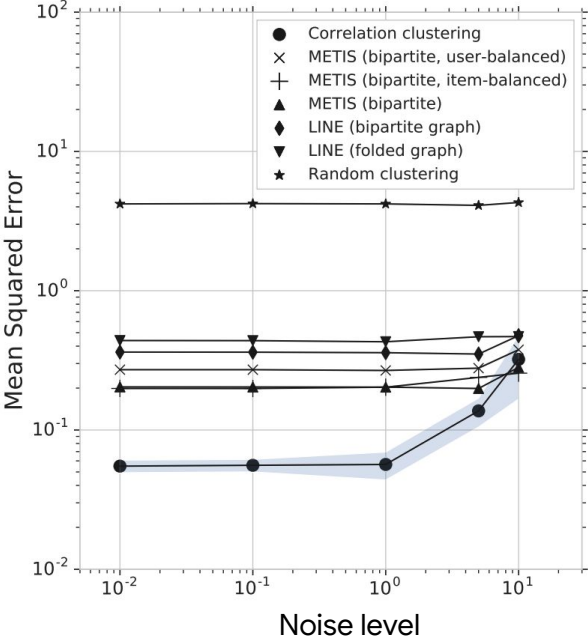
...affects all buyers in it and all buyers competing against them.

# Deep Dive: Correlation Clustering for Marketplace Experiments

- In [1], the authors show that a specific instance of **correlation clustering** is optimal for maximizing the power of item-diverted user-focused marketplace experiments.

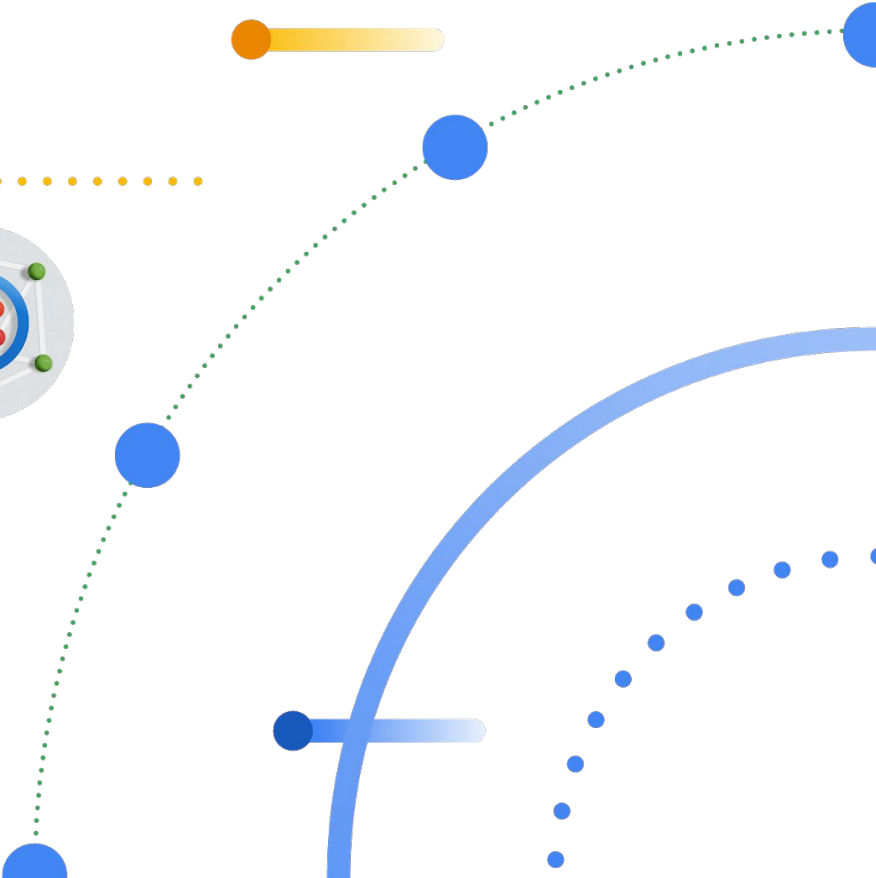


Correlation Clustering outperforms other clustering methods [1]



[1] Pouget-Abadie, Jean, et al. "Variance Reduction in Bipartite Experiments through Correlation Clustering." *NeurIPS*'2019.

# Graph Mining and Learning at Scale



# Agenda

- Grale: Learning Graphs
- Similarity Ranking
- Clustering At Scale
- Community Detection
- Label Propagation



# GrALE: Learning Graphs

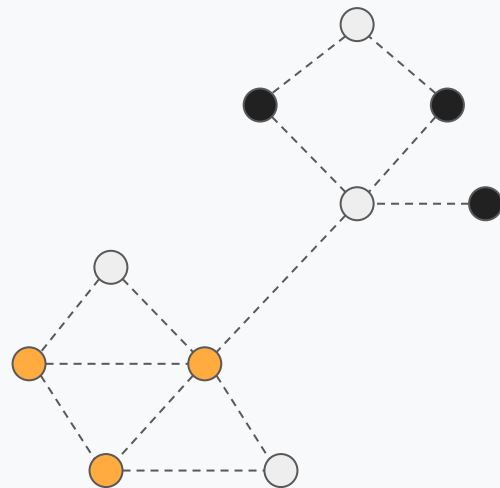
Jonathan Halcrow

Paper: ["GrALE: Designing Networks for Graph Learning"](#) KDD'20



# A Cartoon Example

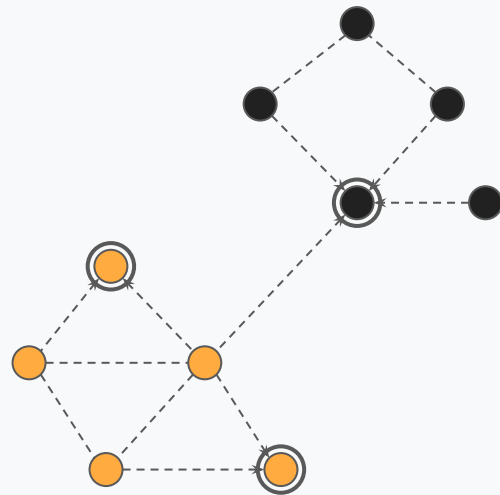
In a toy example, we may be given a partially labeled set of nodes and a graph indicating some similarity relation on the nodes.



# A Cartoon Example

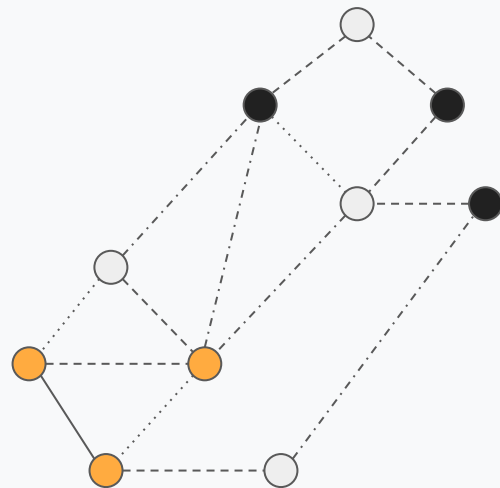
In a toy example, we may be given a partially labeled set of nodes and a graph indicating some similarity relation on the nodes.

We use the graph to infer labels for the unlabeled set, by spreading from the labeled nodes.



# A "Real-World" Example

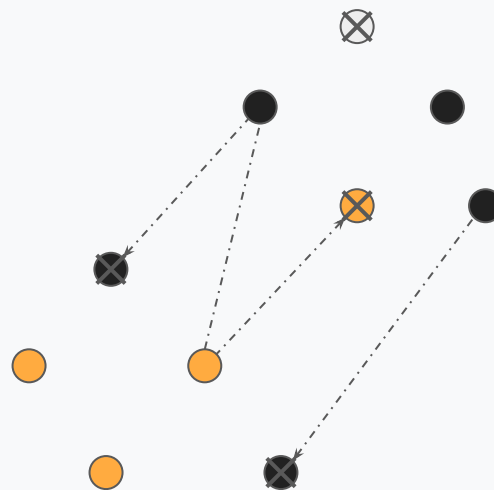
In real world examples, the picture is rarely this clear. Instead of a single set of relationships closely aligned with our target labels, we usually have many types of relationships to pick from, of varying quality.



# A "Real-World" Example

In real world examples, the picture is rarely this clear. Instead of a single set of relationships closely aligned with our target labels, we usually have many types of relationships to pick from, of varying quality.

A bad choice of graph will yield a poorly performing graph learning algorithm.

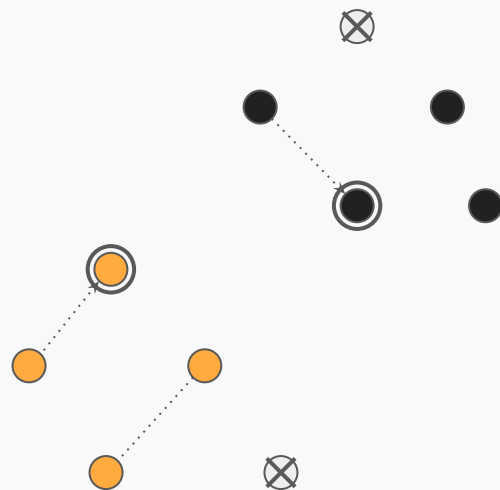


A Bad Choice

# A "Real-World" Example

In real world examples, the picture is rarely this clear. Instead of a single set of relationships closely aligned with our target labels, we usually have many types of relationships to pick from, of varying quality.

The choice of graph is critical for the performance of graph learning algorithms.

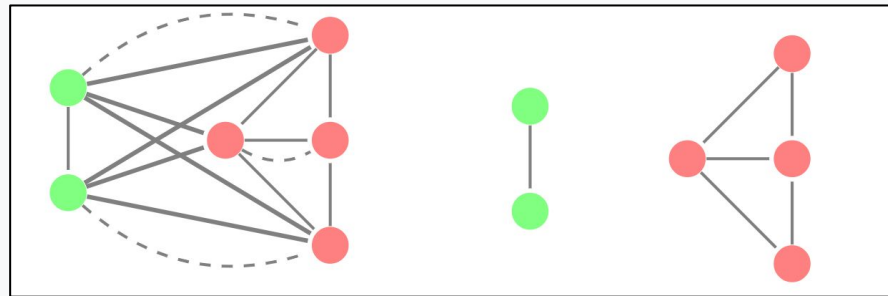


A Better Choice

# The Graph Design Problem

Given:

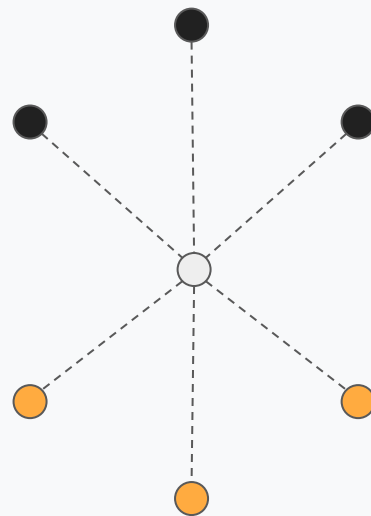
- A multi-modal feature space  $\mathbb{X}$ , each mode with a natural distance measure,  $\kappa_i$
- A partial labeling on this feature space
- A learning algorithm which is a function of some graph  $G$  having vertex set equal to the elements of  $\mathbb{X}$



Observed relationships  
vs. an ideal similarity  
measure

# Application to label propagation

In our paper, we focus on the designing graphs  
for a single hop of label propagation.

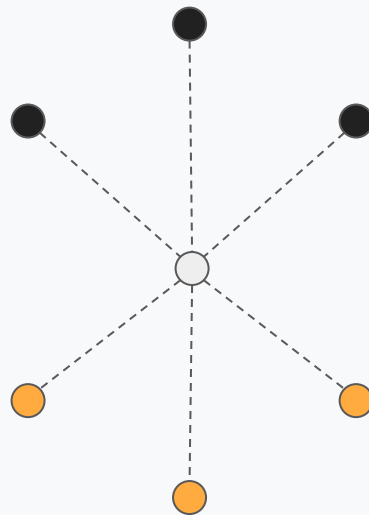


$$\hat{y}_{i,c} = \prod_{j \in \mathcal{N}_{i,c}} G(x_i, x_j)$$

# Application to label propagation

In our paper, we focus on the designing graphs for a single hop of label propagation.

We assume that the nodes in our graph have several different features associated with them, each with a natural distance. We learn the edge weights as functions of these distances.



$$G(x_i, x_j) = f(\kappa_1(x_i, x_j), \kappa_2(x_i, x_j), \dots, \kappa_d(x_i, x_j)).$$

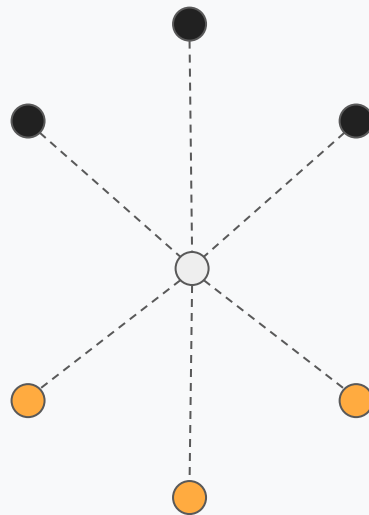


# Application to label propagation

In our paper, we focus on the designing graphs for a single hop of label propagation.

We assume that the nodes in our graph have several different features associated with them, each with a natural distance. We learn the edge weights as functions of these distances.

We show that in this setting, minimizing the log-loss for the multi-class label propagation classifier is equivalent to minimizing the log-loss for the binary prediction that two nodes are in the same class



$$\mathcal{L} = - \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{X}} \sum_{j \in \mathcal{X}} y_{i,c} y_{j,c} \log G(x_i, x_j).$$

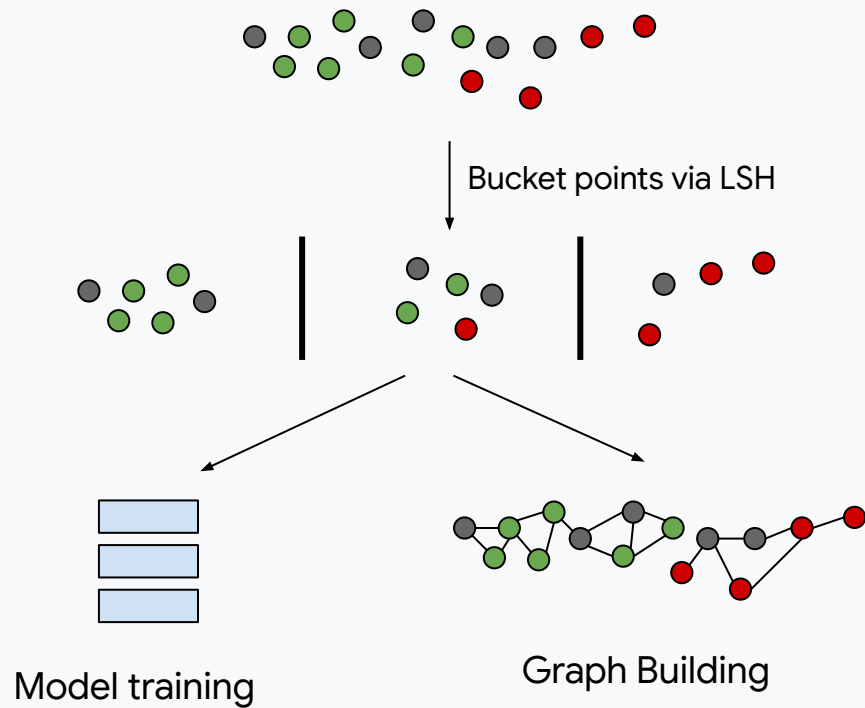
# Grale: A Scalable Solution

## Step 1:

Generate candidate pairs via locality sensitive hashing

## Step 2:

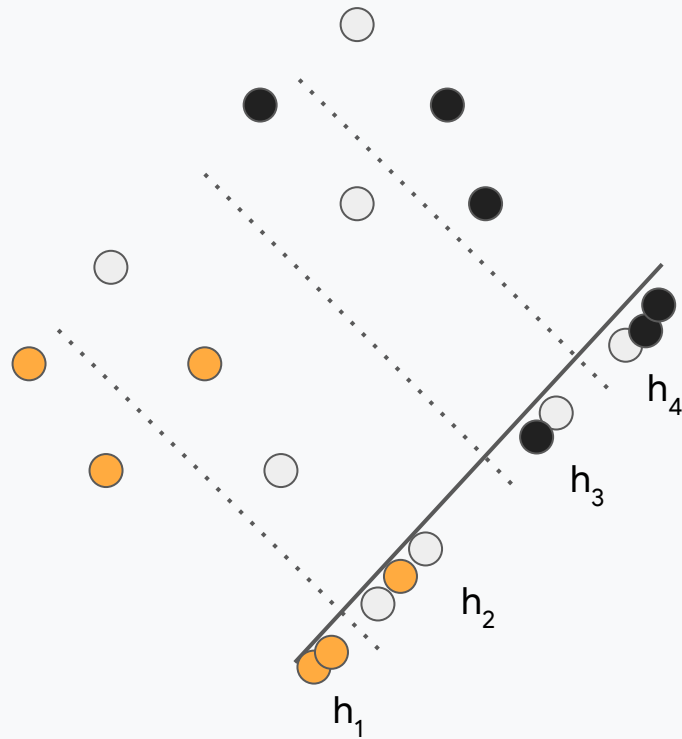
Train a pairwise model to predict same class membership, or apply the model to infer similarity on pairs



# Locality Sensitive Hashing

A key requirement for Grale is that it must scale to datasets containing billions of nodes, making an all-pairs search infeasible. Instead we rely on approximate similarity search using locality sensitive hashing.

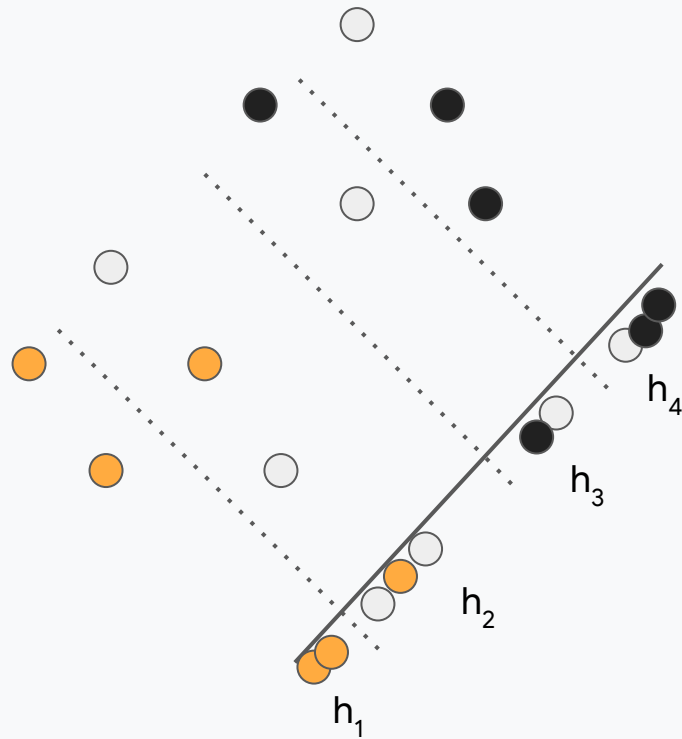
An LSH function is a hash function with the property that points which are 'close' are likely to hash to the same value, while points which are 'far' are unlikely to.



# Locality Sensitive Hashing

In our case, 'close' and 'far' depend on what our model learns. We show that combinations of LSH functions for our simpler per-feature distances can serve as LSH functions for the model.

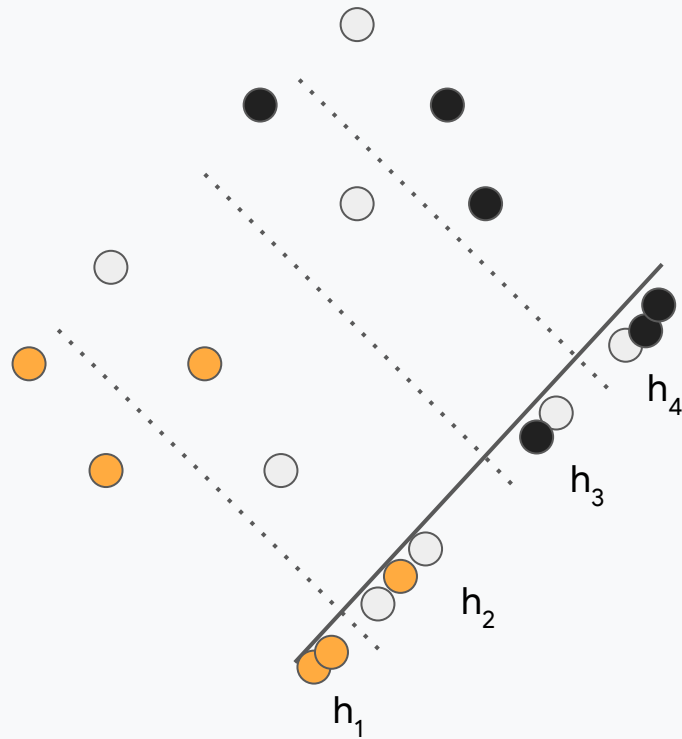
Points which are 'close' in feature values, should also be 'close' according to our learned similarity (with some basic continuity assumptions).



# Locality Sensitive Hashing

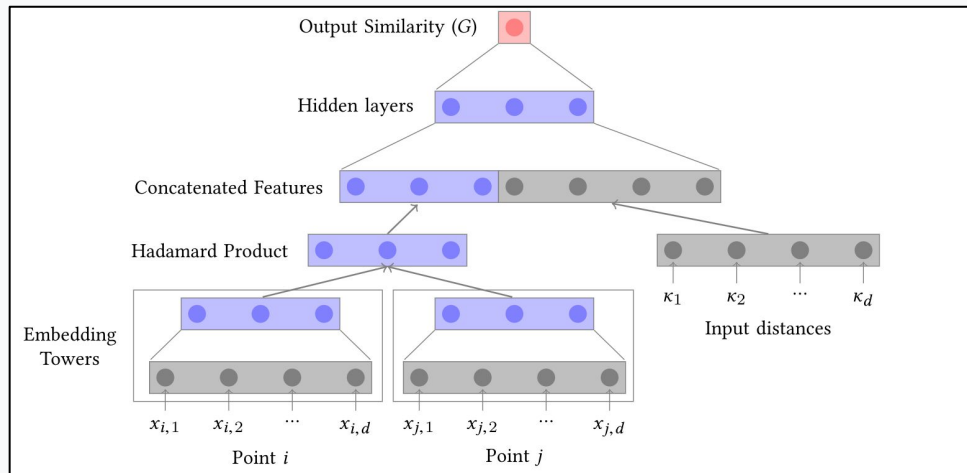
Further Reading (from literature and our team):

- Gionis et al - [Similarity Search in High Dimensions via Hashing](#)
- Charikar [Similarity estimation techniques from rounding algorithms](#),
- Broder et al - [Syntactic Clustering of the Web](#)
- Indyk, Motwani - [Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality](#)
- [Locality-Sensitive Hashing Scheme Based on p-Stable Distributions](#) (Andoni et al)
- Chen et al - [Locality-Sensitive Hashing for f-Divergences and Krein Kernels: Mutual Information Loss and Beyond](#)



# Model Structure

The specific choice of model structure may vary depending on the application, but most commonly we use a neural net which combines a two-tower structure to learn embeddings over the nodes and combines it with the 'natural' distances in the data.



# Evaluation on small datasets

We compare Grale against other techniques which set out to learn task specific similarities for label propagation.

USPS is a handwritten digit set, scanned from envelopes by the U.S. postal service and represented as numeric pixel values.

MNIST is an-other popular handwritten digit dataset, where the images have been size-normalized and centered.

Dataset	Grale	PGLrn	MinEnt	AEW	Grid	Rand <sub>d</sub>
USPS	0.892	0.906	<b>0.908</b>	0.895	0.873	0.816
MNIST	<b>0.927</b>	0.824	0.816	0.782	0.755	0.732

Name	# points	# dimension	# classes
USPS	1000	256	10
MNIST	70000	784	10

# Evaluation on small datasets

We compare Grale against other techniques which set out to learn task specific similarities for label propagation.

We compare to other approaches tested in the paper: "A Quest for Structure: Jointly Learning the Graph Structure and Semi-Supervised Classification" by Wu et al.

The other methods all focus on tuning per-dimension bandwidths in:

$$K(x_i, x_j) = \exp\left(-\sum_m \frac{(x_{im} - x_{jm})^2}{\sigma_m}\right)$$

Dataset	Grale	PGLrn	MinEnt	AEW	Grid	Rand <sub>d</sub>
USPS	0.892	0.906	<b>0.908</b>	0.895	0.873	0.816
MNIST	<b>0.927</b>	0.824	0.816	0.782	0.755	0.732

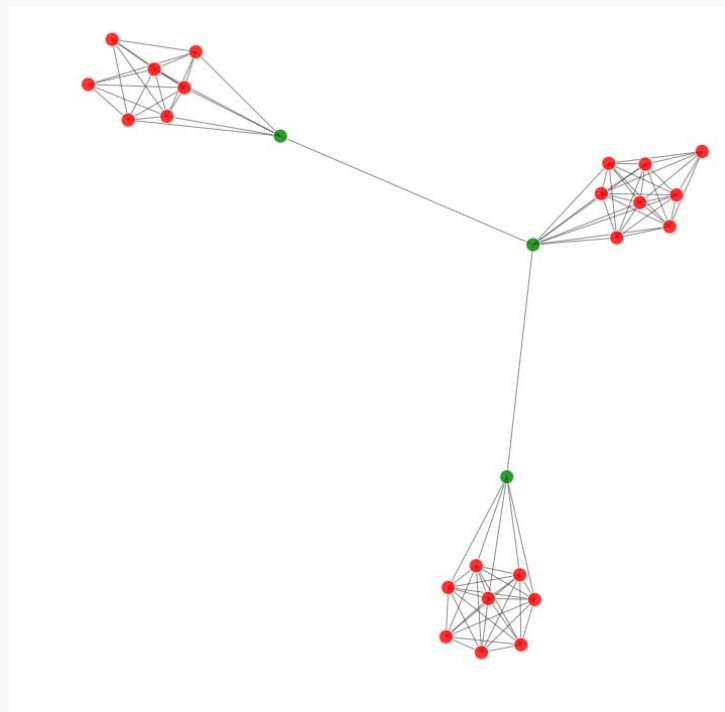
Name	# points	# dimension	# classes
USPS	1000	256	10
MNIST	70000	784	10



# Deployment for YouTube

Grale has been deployed in many different settings within Google. In particular it is used by YouTube to detect malicious actors.

We train the Grale model in this case to differentiate pairs of abusive items from pairs where at least one item is non-abusive.



A subgraph of related items on YouTube found by Grale

Google Research

# LSH Efficiency

A comparison of the LSH function used for YouTube and a naive baseline.

"Strong ties" are the fraction of pairs returned by LSH that are closer than a distance useful for high precision decisions.

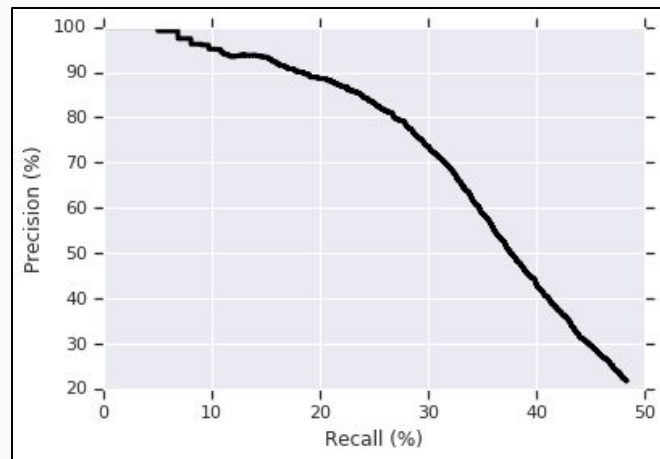
"Weak ties" are those with worse than a moderate precision threshold.

	% strong ties (p)	% weak ties (q)
baseline (random pairs)	0.0653%	77.4%
tuned LSH	52.3%	22.7%

# Single Nearest Neighbor performance

In practice, Grale is used as an input to a more complex abuse fighting system. To illustrate the performance of the graph alone, we compute the precision and recall of a single nearest neighbor classifier.

In this evaluation, we select the oldest 25% of known abusive nodes as seeds, and evaluate against the newest 75%.



# Comparison to other approaches

The Grale+Label Propagation system is deployed alongside various heuristics and content based classifiers.

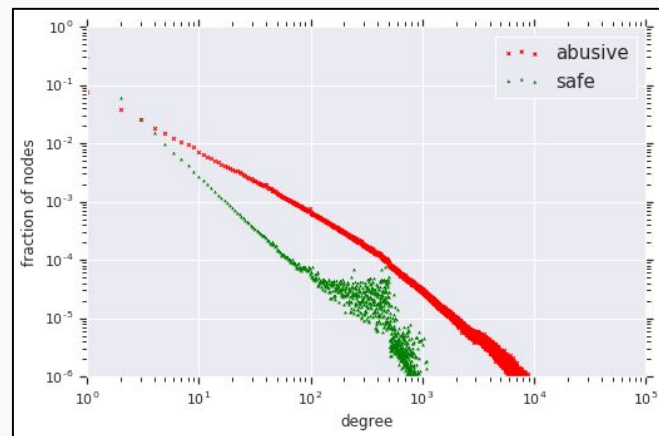
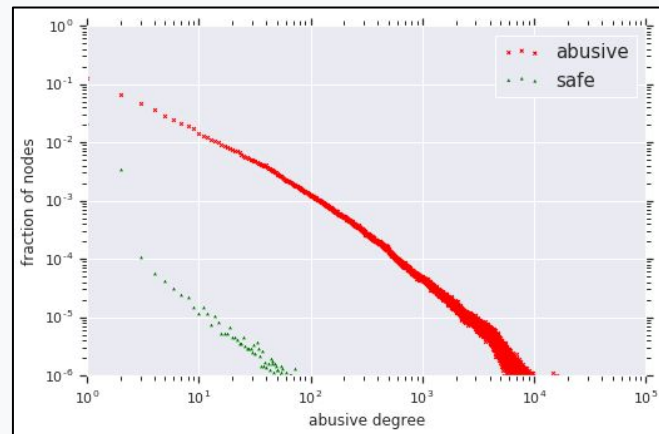
For the type of items that we target here, we increase recall by 89% vs these other approaches alone. In particular we find many items that are missed by a first pass by purely content based classifiers.

Algorithm	% of items
Content Classifiers	47.7%
Heuristics	5.3%
Total (Heuristics + Content Classifiers)	53%
Grale+Label Propagation	47.0% (+89%)

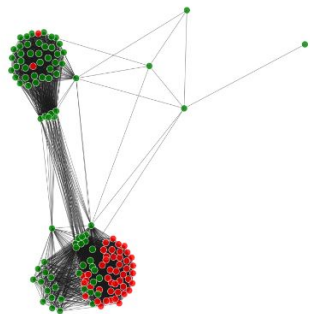
Algorithm	New items	Old items
Grale+Label Propagation	25.8%	74.2%
Content Classifiers	71.6%	28.4%
Heuristics	33.7%	66.3%

# YouTube Graph Structure

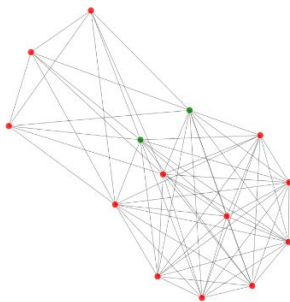
Sorting the degree distribution in the graph by abuse status. We see that abusive nodes have much higher degree on average and are particularly strongly connected to other abusive nodes. This is precisely what we are hoping to achieve.



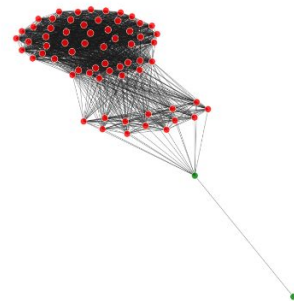
# Example Clusters Found on YouTube



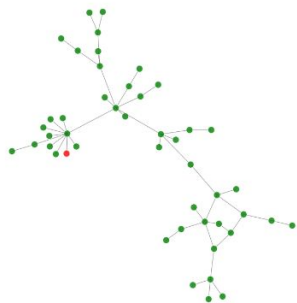
(a) Intermixed dense clusters.



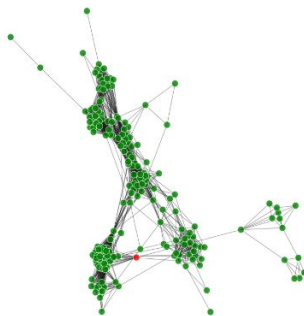
(b) Sparse abusive cluster.



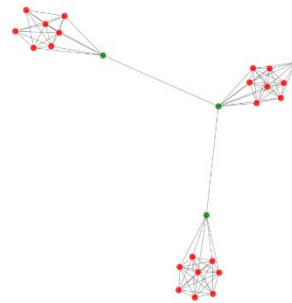
(c) Dense abusive cluster.



(d) Sparse non-abusive subgraph.



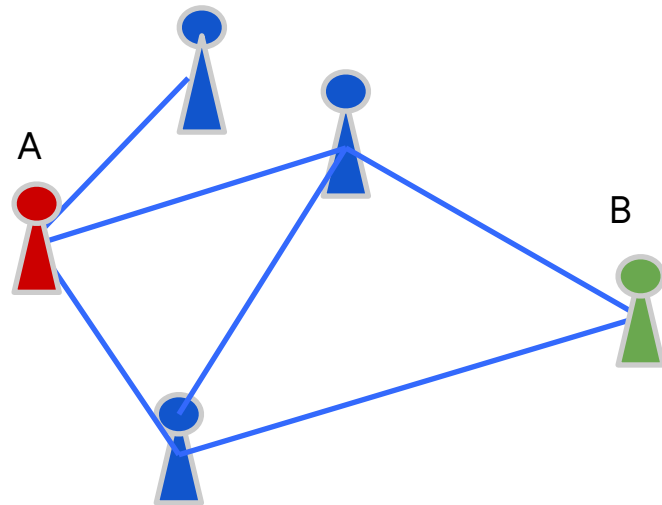
(e) Dense non-abusive clusters.



(f) Small abusive clusters.

# What else can we do?

- Unsupervised structural similarity measures
  - Personalized PageRank and more (see next section!)
- Graph Embeddings / Graph Neural Network methods
  - DeepWalk (later)
  - Deep Graph InfoMax
- Variational Autoencoders
  - Usefully captures the key information with a natural similarity measure
- Can also include any of the above as signals to Gated style models!





# Similarity Ranking

Alessandro Epasto



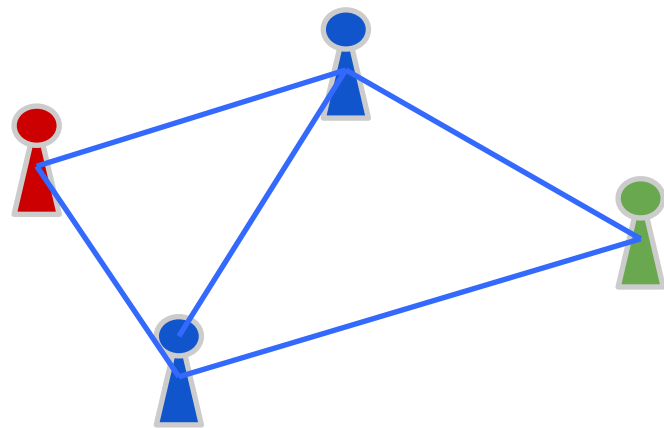
# Similarity Ranking

Before we have seen how to build a **similarity graph** given **non-graph data**.

Here we will address the question: Given a graph, how **similar are two nodes in the graph**? Can we predict missing edges from the graph?

Classical graph problem with applications in:

- Link prediction;
- Recommender systems, Collaborative filtering;
- Spam & Abuse detection, Anonymity detection;
- Graph embeddings;
- Clustering;
- Feature engineering in graph-based learning.



# Classical Unsupervised Similarity Scores

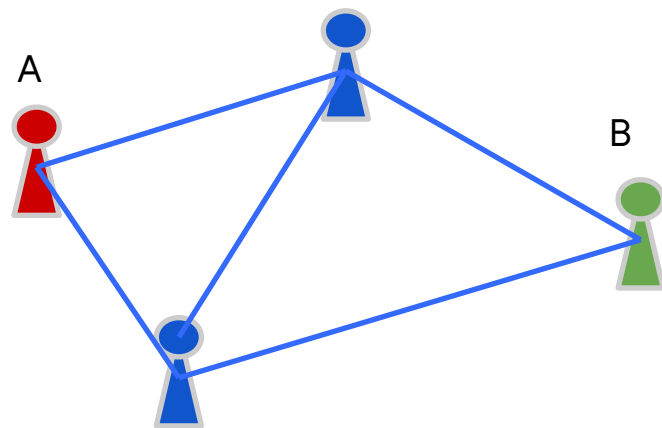
The unsupervised version of the problem has a long stream of work starting from Liben-Nowell & Kleinberg in 2004.

Input: a weighted graph (no additional side information).

Output: a score for a given pair of nodes in the graph.

Similarity scores between pairs of nodes are defined by multi-hop neighborhood measures, e.g. number of direct, or indirect connections between users.

Extensions include using side information, heterogeneous graphs, etc.



# Classical Unsupervised Similarity Scores

Some classic examples, similarity of A and B:

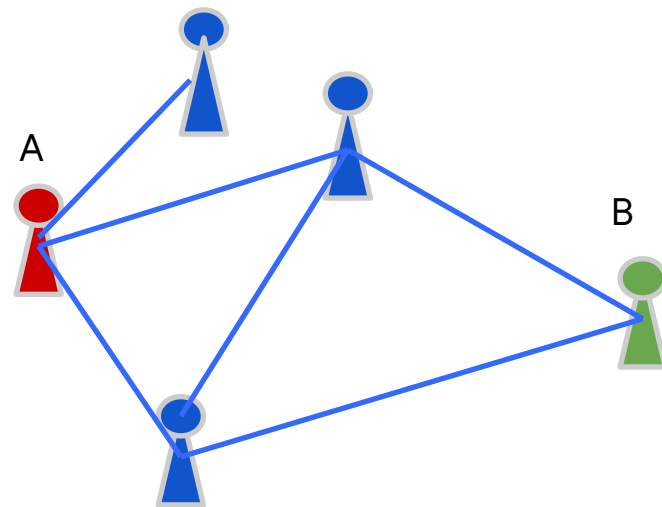
- Single hop:

- Common Neighbors  $|N(A) \cap N(B)| = 2$
- Jaccard coefficient  $|N(A) \cap N(B)| / |N(A) \cup N(B)| = 2/3$
- Adamic Adar

$$\sum_{z \in N(u) \cap N(v)} \frac{1}{\log(d(z))}$$

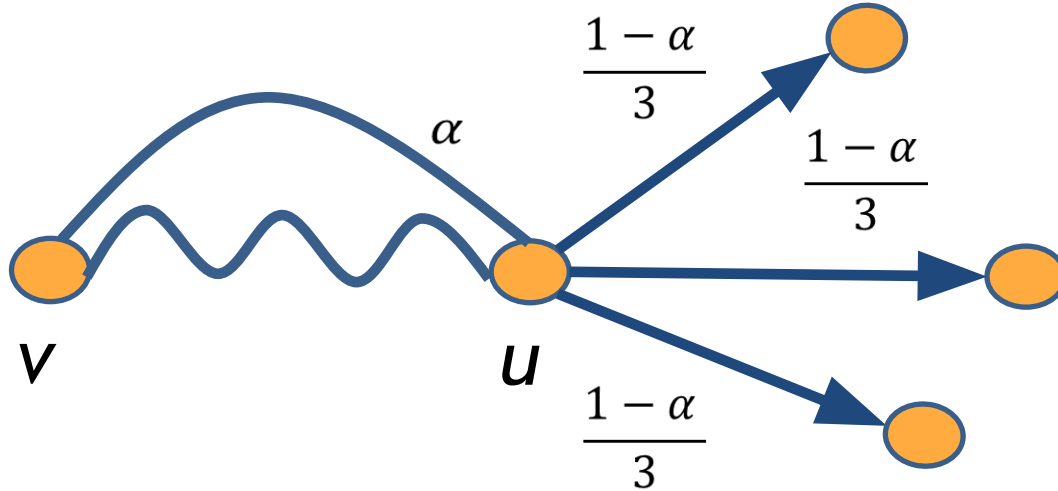
- Multi-hop:

- Katz score
- Personalized PageRank.



# Personalized PageRank (PPR)

For a node  $v$  (the seed) and a probability alpha



The stationary distribution assigns a similarity score to each node in the graph w.r.t. node  $v$ .

# PPR is fast

- Extensive algorithmic literature on efficient approximation methods (Andersen Chung and Lang, 2007).
- Efficient MapReduce / Distributed algorithm scaling to *large* graphs (billions of edges).
- Very good accuracy in our experimental evaluation compared to other similarities (Jaccard, Intersection, etc.).

# Applications of PPR and Similarity Ranking

- Clustering (*Andersen Chung and Lang, 2007, A Local Algorithm for Finding Well-Connected Clusters Zhu et al. ICML'13*) → Graph Clustering session
- Efficient GNN (*PPRGo: GNNs at Scale KDD'20 Perozzi et al.*) → Graph Neural Network session.
- Efficient Graph Embeddings (*VERSE: Versatile Graph Embeddings from Similarity Measures, Tsitsulin et al. WWW'18*)
- Spam & Abuse Detection (*Robust PageRank and locally computable spam detection features, Mirrokni, AirWeb08. Local Computation of PageRank Contributions Andersen et al. WAW'07*)
- Heterogenous graph ranking (*Reduce and Aggregate: Similarity Ranking in Multi-Categorical Bipartite Graphs, Epasto et al. WWW'14*) → This talk
- Suggestions in social networks (*Improved friend suggestion via Ego-net clustering, Epasto et al. VLDB16*) → This talk

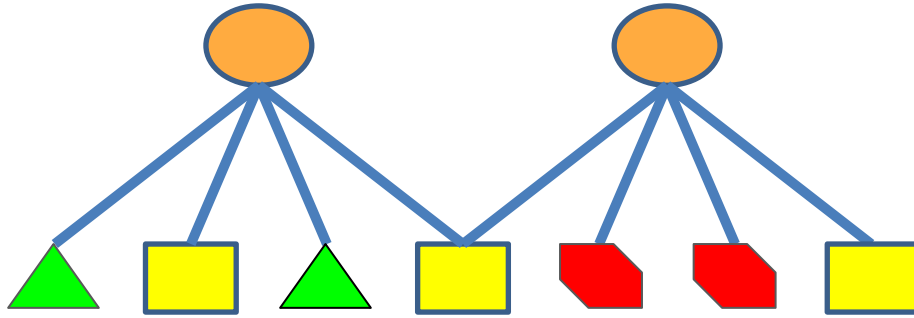


# Reduce and Aggregate: Similarity Ranking in Multi-Categorical Bipartite Graphs

Based on work by: Epasto, Feldman, Lattanzi, Leonardi and Mirrokni, WWW2014

# Heterogenous bipartite graph ranking

Users



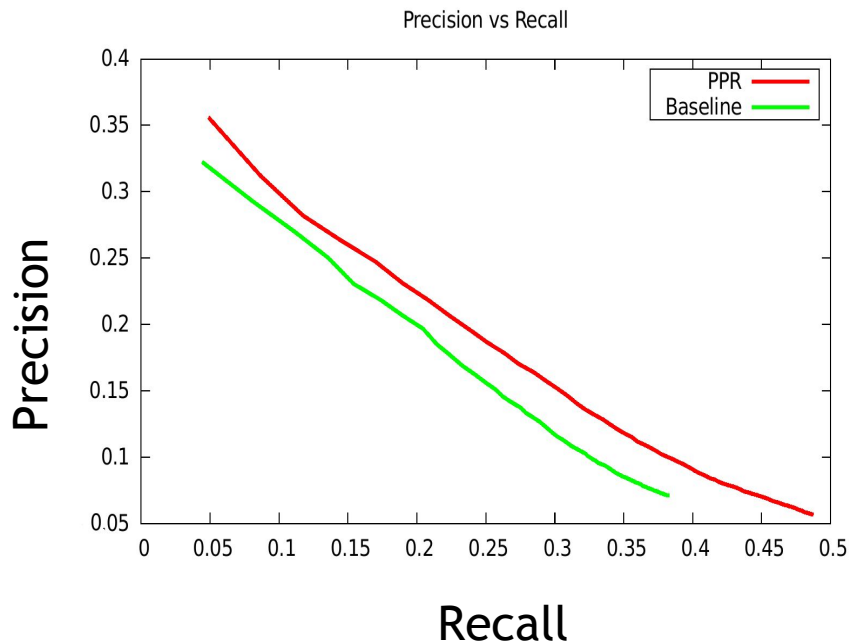
Items of different types

Given a subset of categories of interest determine a similarity ranking for the users.

We provide efficient distributed and real-time algorithms for the problem.



# Heterogenous bipartite graph ranking



We provide results for computing rankings based on PPR, and other 2-hop similarities.

Experiments on a AdWords recommendation problem.

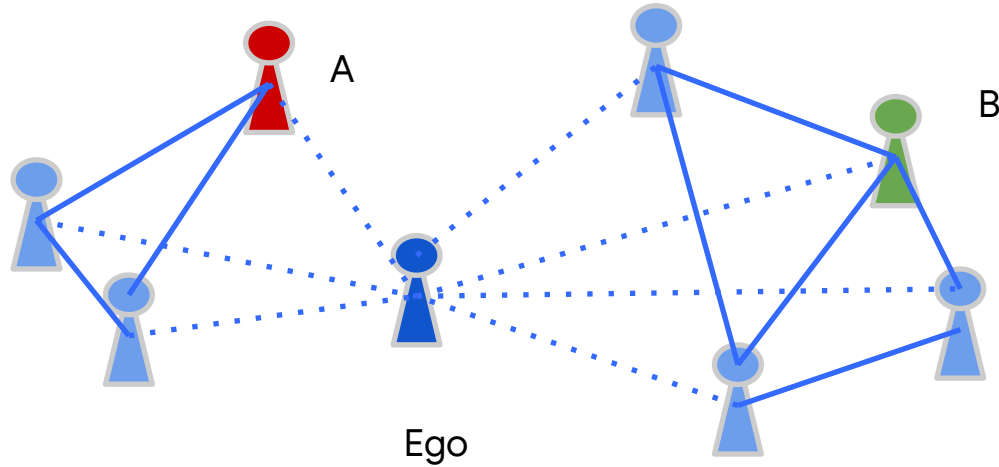


# Improved friend suggestion via Ego-net clustering

Based on work by: Epasto, Lattanzi, Mirrokni, Sebe, Tai, Verma VLDB'16

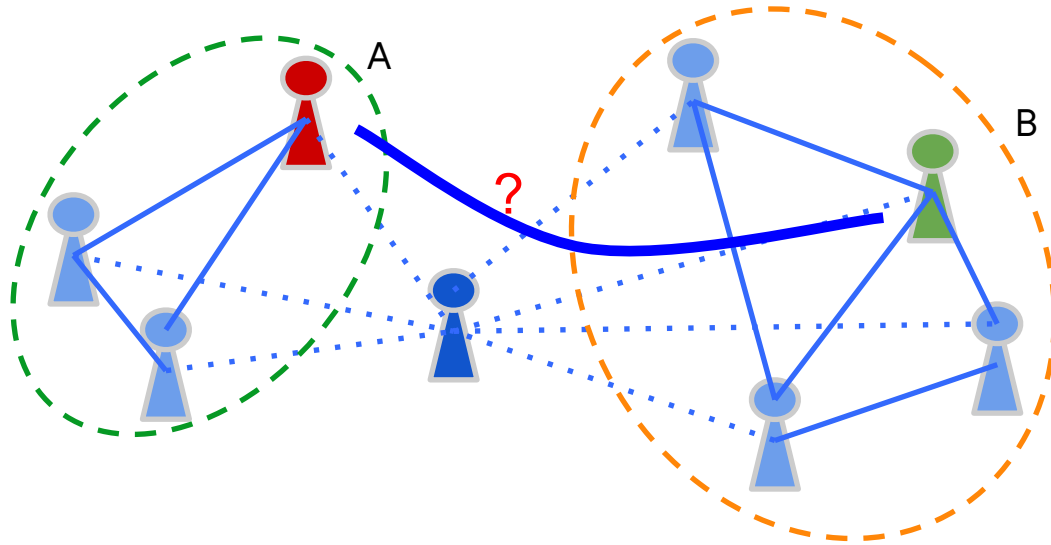
# A problematic case for graph similarity

The ego node part of many community.



# Improving Collaborative Filtering

Should we suggest A to B as similar?

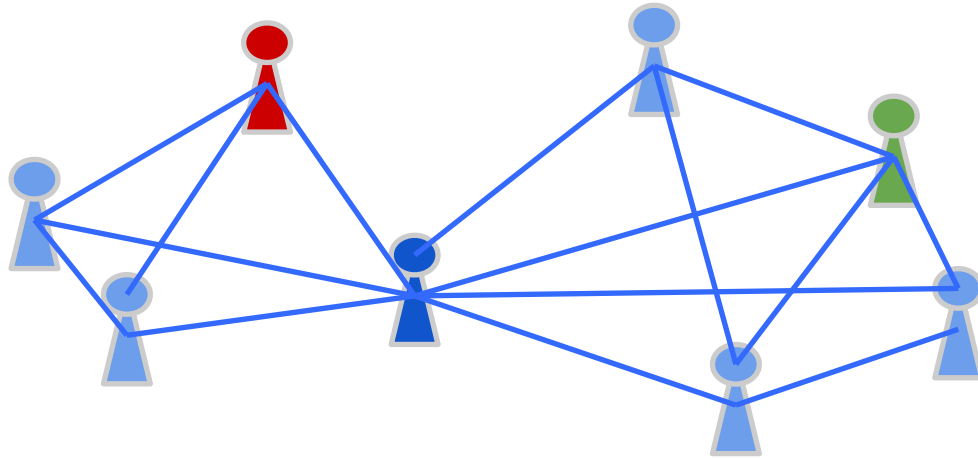


We should really cluster the ego-networks

# Ego-network score

Suppose we cluster all ego-networks.

Ego-network score 
$$\sum_{z \in N(u) \cap N(v)} \mathbb{I}\{C_z(u) = C_z(v)\}$$

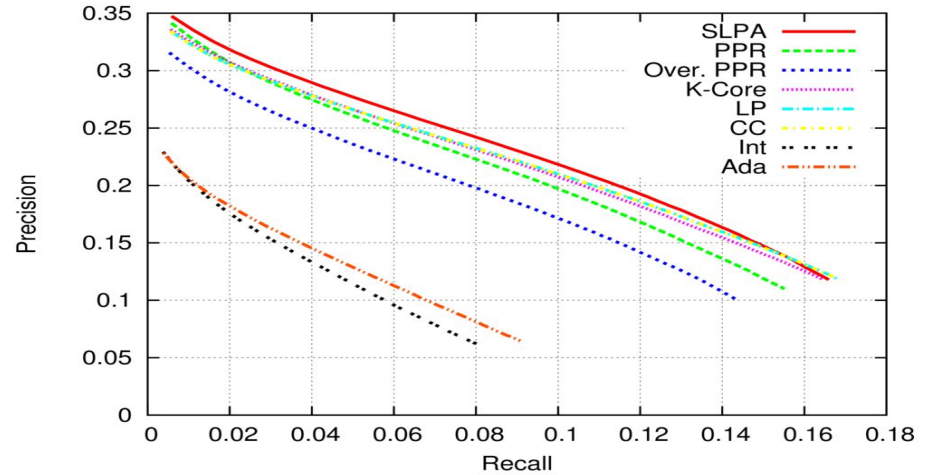


Ego-network score, e.g., # ego-net clusters two nodes belong to

# Results: Ego-network score

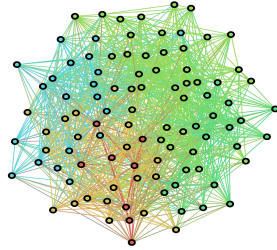
Evaluation: ablation analysis, where we remove some edges and then we try to predict them.

Live Experiments: 1.4% decrease in live rejection rate (see paper)

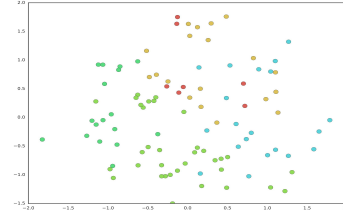


# Follow-up work: Persona Graph and Embeddings

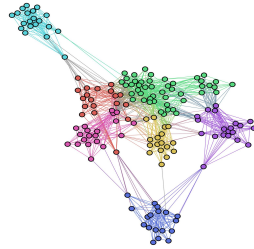
Original Graph



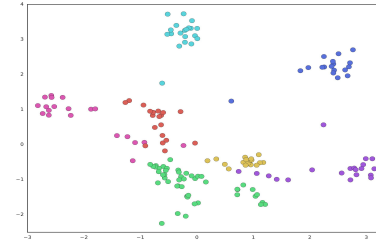
node2vec



Persona  
Graph



splitter



Epasto, Lattanzi, Leme - KDD'17.

Epasto Perozzi WWW'19 will be covered in the Graph Neural Learning Section.



# Clustering at Scale



Vahab Mirrokni

Based on several papers, e.g.,

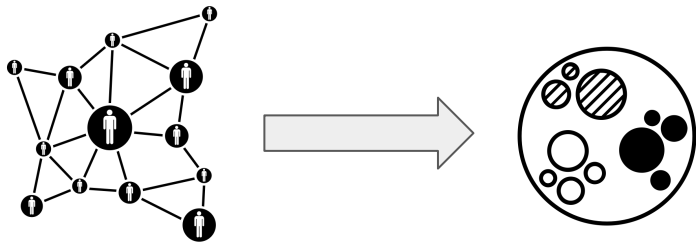
- Affinity Clustering: Hierarchical Clustering at Scale, Bateni, Behnezhad, Hajiaghayi, Kiveris, Lattanzi, Mirrokni, NeurIPS'17
- Distributed Balanced Partitioning via Linear Embedding: Aydin, Bateni, Mirrokni, WSDM'16
- Mapping Core-sets for Balanced Clustering, Bateni, Bhaskara, Lattanzi, Mirrokni, NeurIPS'14.
- Optimal Distributed Submodular Maximization via Sketching, Bateni, Esfandiari, Mirrokni, KDD'18.  
Applications in KDD'18, VLDB'19, NeurIPS'19



# Clustering: Motivation

## Clustering

Graph Mining is about pattern recognition. The most basic pattern is ‘these nodes are alike, group them together’. Clustering tools allow us to identify such patterns in data. With a wide array of clustering algorithms available, we have a lot of fine grained control over how clusters are created.



## Many Applications, e.g.,

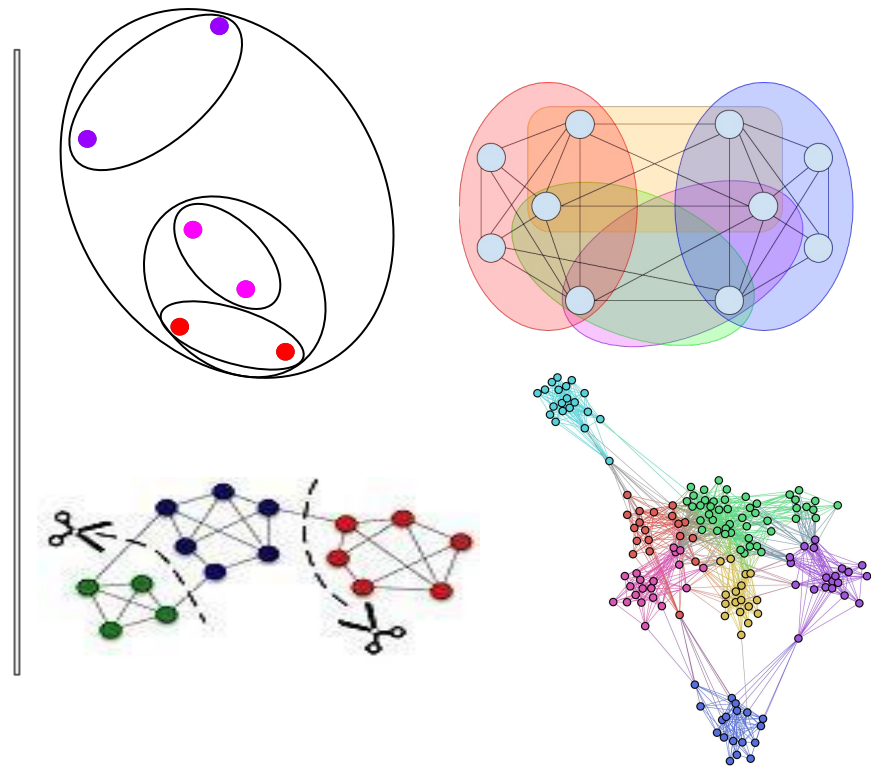
- Feature Engineering:
  - Clusters as features & input for ML models
- Preprocessing for other Graph-based Learning
  - ClusterGCN, ego-net clusters for GNNs
- Causal inference and Experimental Design
  - Minimize interference: Section 1
- Privacy and Anonymity
  - Min-size & anonymity clustering : Section 1
- Data Efficiency:
  - Diversified sampling: Coverage, K-means
- Many more applications in model efficiency,

# Clustering: Many Algorithms and Techniques

- Hierarchical Clustering: Affinity, HAC, pHAC, ...
- Metric Clustering: K-means, DBScan, K-center, ...
- Clustering w/ constraints: Balanced, Min-Size, ...
- Community Detection: Modularity, Local Random Walk, Ego-net and Correlation Clustering, ...

Big Challenge: Doing it at Scale!

Techniques: Random Walks, Sketching and Locally Sensitive Hashing(LSH), Composable Core-sets, ...

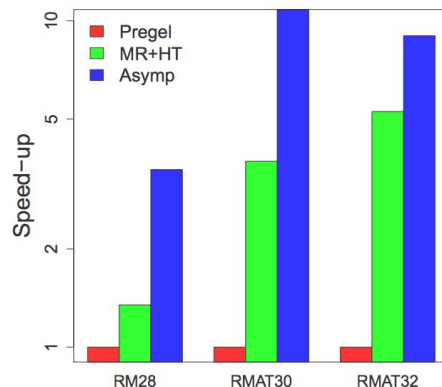
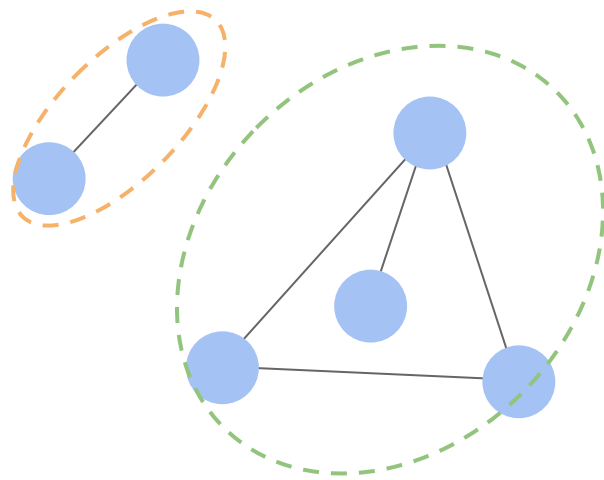


# Algorithms and Systems for Clustering

For each algorithm, we explore several combinations of systems+algorithms. For example, for connected components:

- [MapReduce+DHT paper](#)
- [Flume via Local Contractions](#)
- [ASYMP paper](#)
- Up to *50X speedup* over baselines

→ Will discuss in the last section

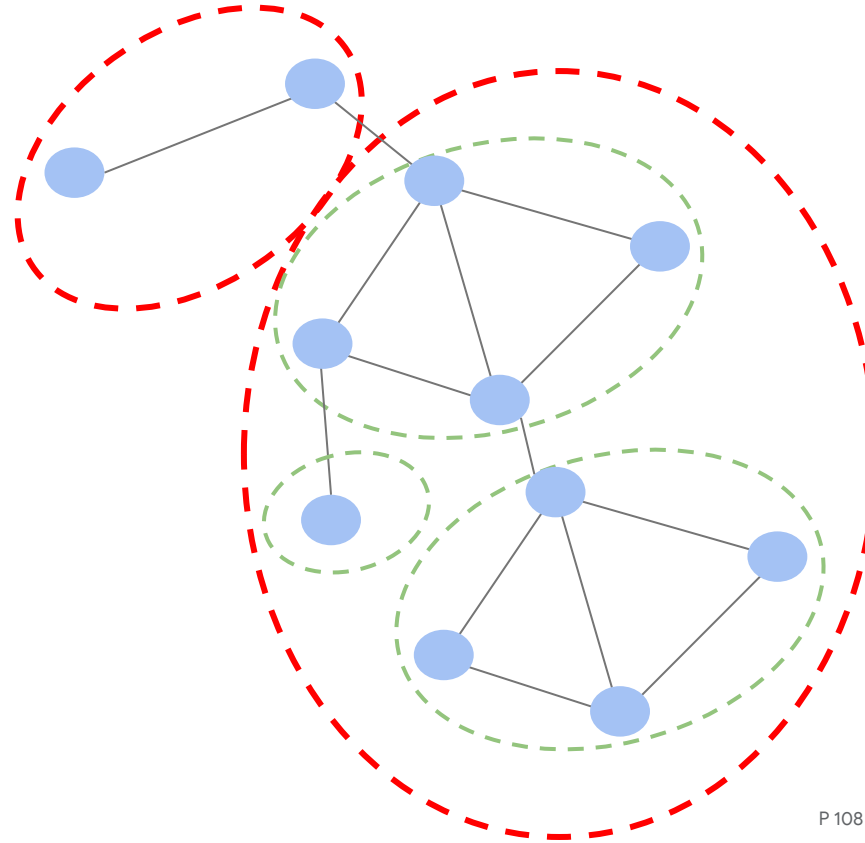


# Hierarchical Clustering

A Clustering method that seeks to build a hierarchy of clusters

Many sequential algorithms:  
HAC, Single Linkage, Avg Linkage.

Parallel Hierarchical Clustering:  
Affinity Clustering, Parallel HAC.



# Affinity Hierarchical Clustering

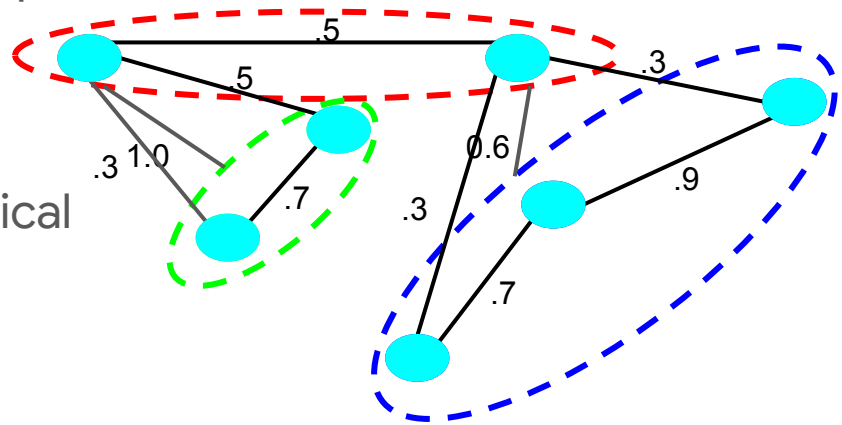
*Parallel Linkage Clustering via use of MapReduce & Distributed Hash Tables (DHT)*

- Keep heaviest edge above a threshold incident to each node
- Compute clusters and construct graph between clusters
- Iterate & Recluster

Subroutine in many applications

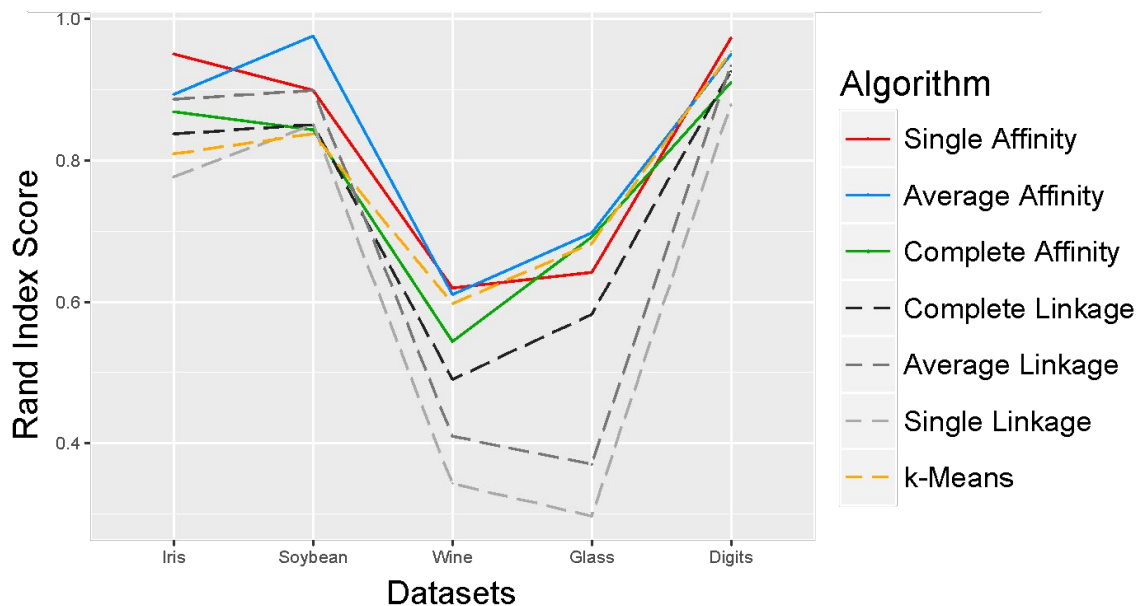
[Paper in NeuIPS'17](#): On Affinity Hierarchical Clustering, bateni et al [[Video](#)]

- Theoretical study
- Applied to graphs with Trillions of edges
- Better quality compared to HAC, k-means...



# Affinity Clustering: Empirical Study (Quality)

- Datasets are from the UCI database and we use Euclidean distance.



# Affinity Clustering: Empirical Study (Scalability)

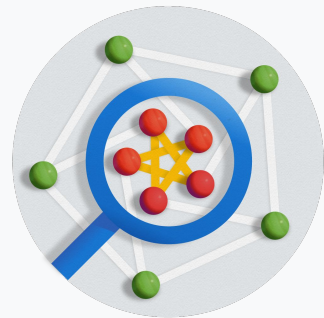
- The first three graphs in table 1 are based on public graphs and the last graph is based on an internal corpus of public images found on the web and their similarities.

Dataset	# nodes	# edges	max degree	running time	speedup
LiveJournal	4,846,609	7,861,383,690	444,522	1.0	4.3
Orkut	3,072,441	42,687,055,644	893,056	2.4	9.2
Friendster	65,608,366	1,092,793,541,014	2,151,462	54	5.9
ImageGraph	$2 \times 10^{10}$	$10^{12}$	14000	142	4.1



# Distributed Balanced Partitioning via Linear Embedding and Affinity Clustering

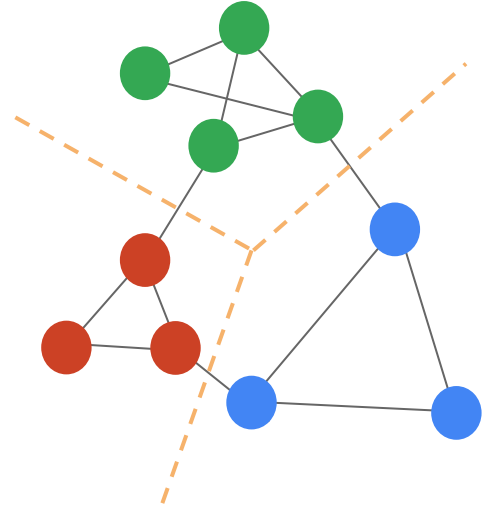
Distributed Balanced Partitioning via Linear Embedding: Aydin, Bateni, Mirrokni, WSDM'16,





# Balanced Partitioning Problem

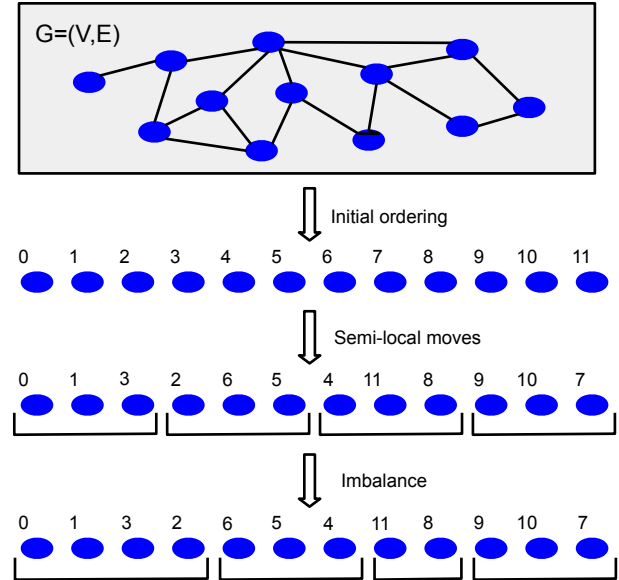
- Balanced Partitioning:
  - Given graph  $G(V, E)$  with edge weights
  - Find  $k$  clusters of approximately the same size
  - Minimize Cut, i.e., #intercluster edges
- NP-hard even to approximate.
- Goal: Solve at Scale



# Outline of Algorithm

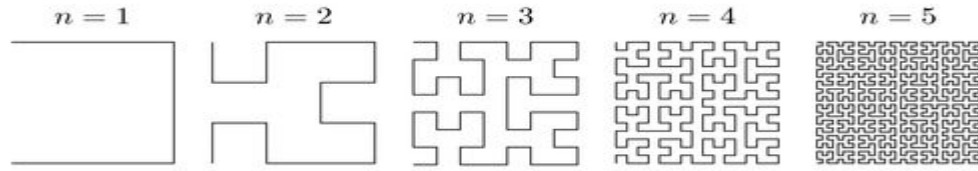
## Three-stage Algorithm:

1. Initial Ordering: One-dimensional embedding
  - a. Space-filling curves
  - b. Hierarchical clustering
2. Semi-local moves
  - a. Min linear arrangement
  - b. Optimize by random swaps
3. Introduce imbalance
  - a. Dynamic programming
  - b. Linear boundary adjustment
  - c. Min-cut boundary optimization

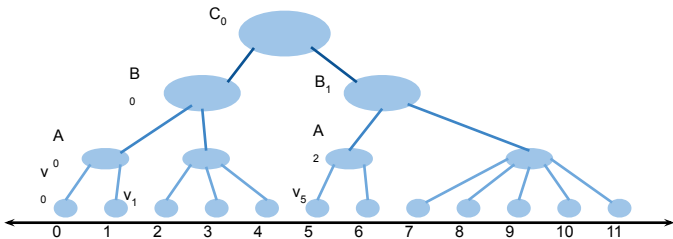


# Step 1 - Initial Embedding

- Space-filling curves (Geo Graphs)



- Affinity Hierarchical clustering (General Graphs)



# Datasets

- Social graphs
  - Twitter: 41M nodes, 1.2B edges
  - LiveJournal: 4.8M nodes, 42.9M edges
  - Friendster: 65.6M nodes, 1.8B edges
- Geo graphs
  - World graph > 1B edges
  - Country graphs (filtered)

# Comparison to Previous Work (LiveJournal)

- Paper

- Best balance and cut size

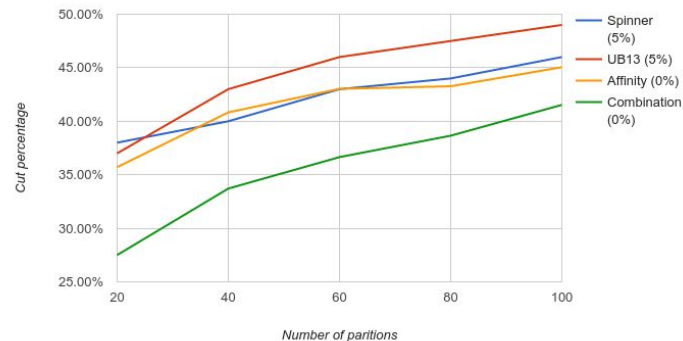
- Dataset

- LiveJournal: 4.8M nodes, 42.9M edges

- Related work

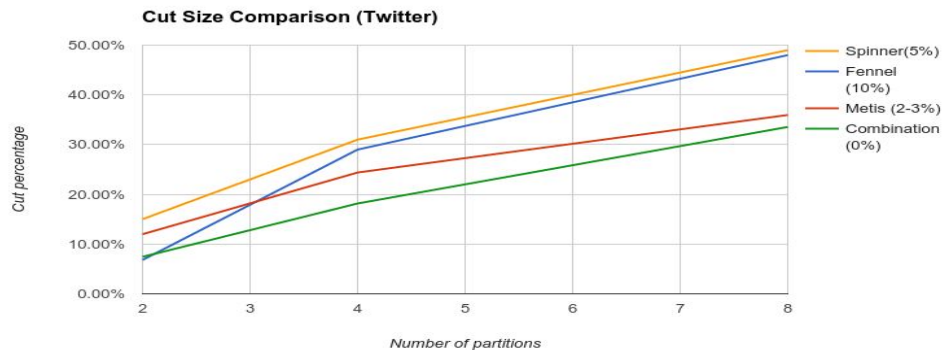
- Spinner (recent) arXiv, [Martella et al.]
- UB13, WSDM'13 [Ugander & Backstrom]
  - Developed at Facebook
  - Balanced label propagation

Cut Size Comparison (LiveJournal)



k	Spinner (5%)	UB13 (5%)	Affinity (0%)	Balanced Partition (0%)
20	38%	37%	35.71%	<b>27.5%</b>
40	40%	43%	40.83%	<b>33.71%</b>
60	43%	46%	43.03%	<b>36.65%</b>
80	44%	47.5%	43.27%	<b>38.65%</b>
100	46%	49%	45.05%	<b>41.53%</b>

# Comparison to Previous Work (Twitter Graph)



<i>k</i>	<i>Spinner</i> <i>(5%)</i>	<i>Fennel</i> <i>(10%)</i>	<i>Metis</i> <i>(2-3%)</i>	<i>BalancedPart.</i> <i>(0%)</i>
2	15%	<b>6.8%</b>	11.98%	7.43%
4	31%	29%	24.39%	<b>18.16%</b>
8	49%	48%	35.96%	<b>33.55%</b>

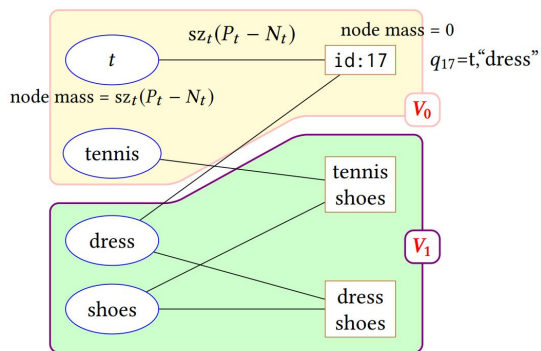
- Twitter: 41M nodes, 1.2B edges

# Examples of Applications of Balanced Partitioning at Google

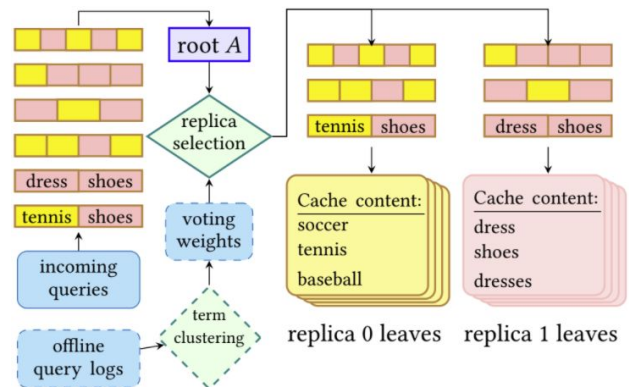
- Serving in Google Maps Directions
  - Serve the graph out of N serves.
  - Minimize the # of multi-server source-destination pairs.
- Balanced partitioning for backend of Google Search (VLDB'19)
  - Better caching properties result in -32% flash consumption
  - Affinity-aware caching via balanced partitioning
- Facilitate A/B experiments under network interference (NeurIPS'19, KDD'18)
  - covered in the first section

# Application: Cache-aware load balancing

- TARS = term-affinitized replica selection
- "[Cache-aware load balancing of data center applications](#)," by Archer, Aydin, Bateni, Mirrokni, Schild, Yang, Zhuang (VLDB'19)



Balanced Graph Partition  $\rightarrow$  TARS  
voting table



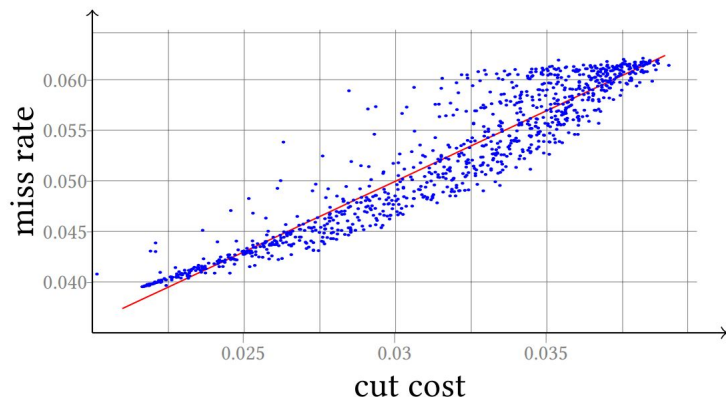
cache-aware replica selection



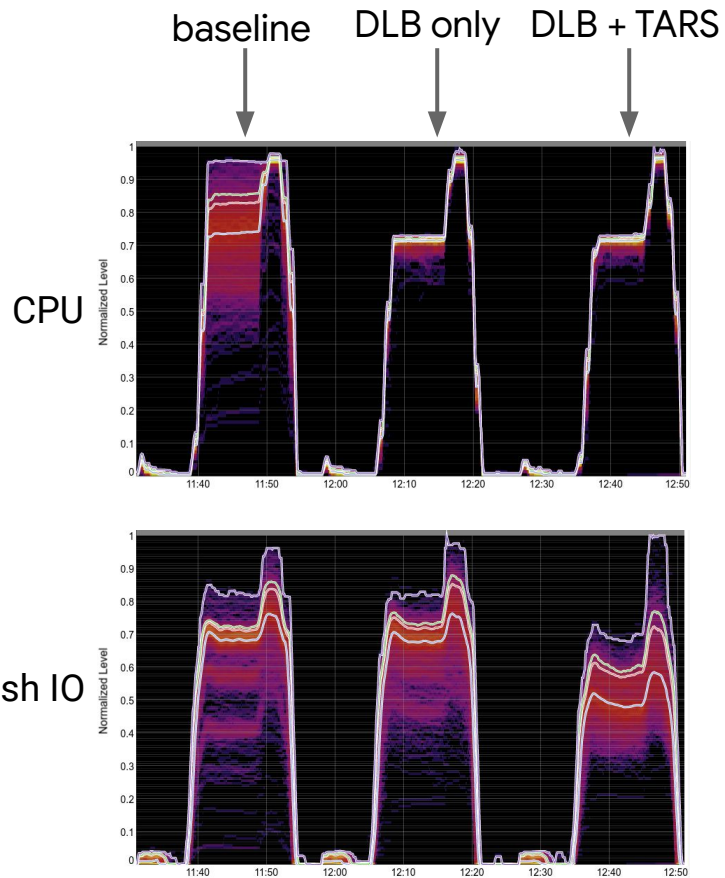
# Cache-aware load balancing via balanced partitioning

Impact on Search Backend:

- -32.5% flash IO, -1.5% CPU cost

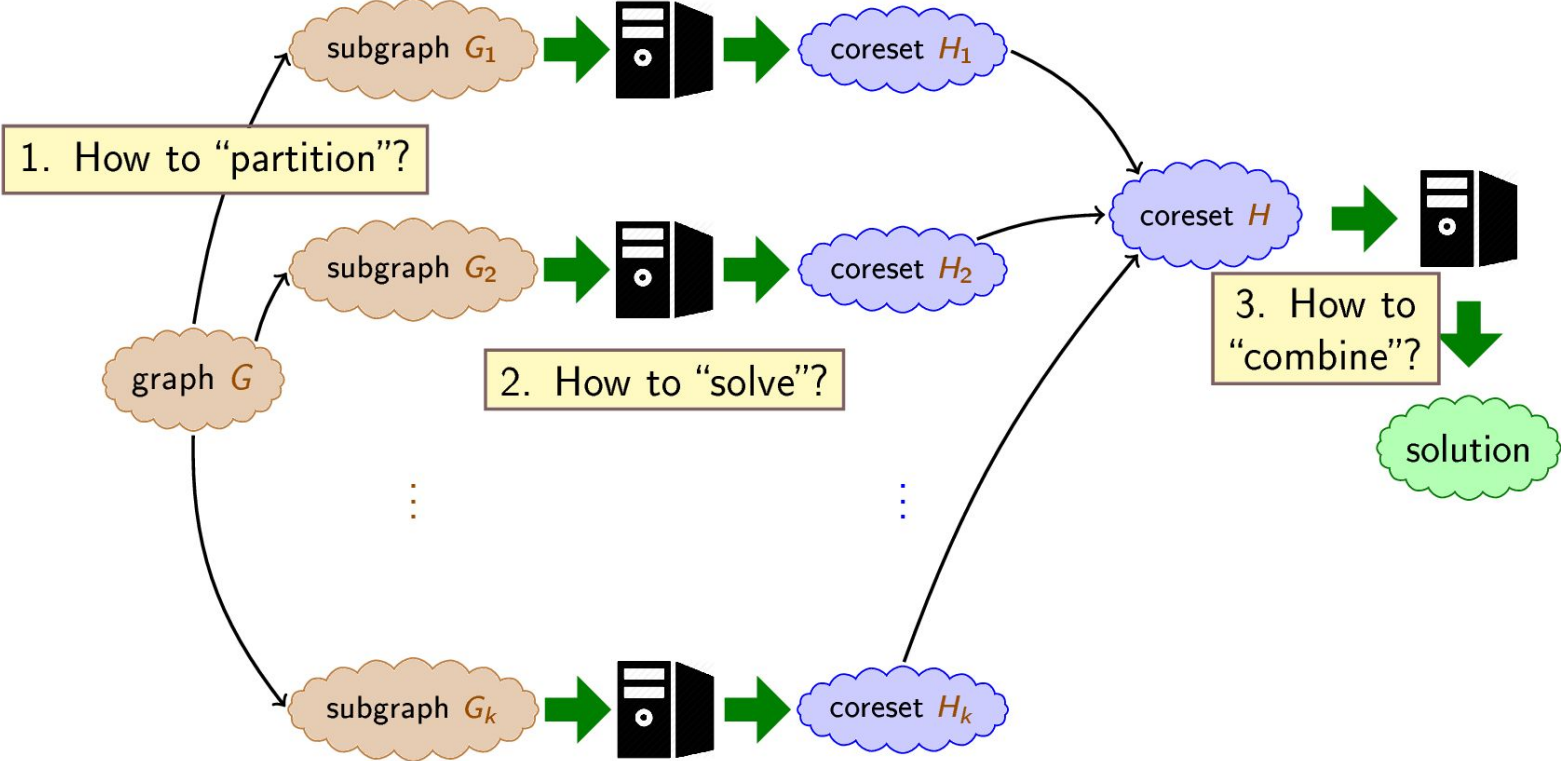


Graph cut cost predicts FBM cache miss rate



# Randomized Composable Core-sets

Send each edge to 1 machine or constant #machines at random.



# Composable Core-sets for Distributed Algorithms

**Composable core-sets:** Defined on a metric space.

1. Diversity Maximization,
  - PODS'14 by Indyk, Mahdian, Mahabadi, Mirrokni
  - for Feature Selection in AAAI'17 by Abbasi, Ghadiri, Mirrokni, Zadimoghaddam
2. Capacitated  $\ell_p$  Clustering, NeuIPS'14 by BateniBhaskaraLattanziM. ← This Talk

**Randomized composable core-sets:** Beyond Metric Spaces.

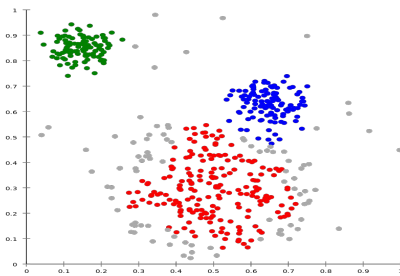
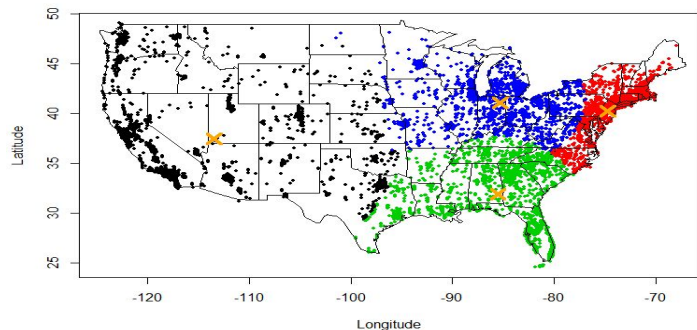
3. Submodular Maximization, STOC'15 by M. Zadimoghaddam
4. Feature Selection (Column Subset Selection), ICML'16 by Alschulter et al.
5. Bipartite Matching, SODA'19 by Assadi et al.  
Weighted Matching, by AssadiBateniMirrokni, ICML'2019

**LSH-based composable core-sets:**

6. Coverage Problems: by Bateni, Esfandiari, M., SPAA'17 + KDD'18
7. Extreme k-center via LSH-based partitioning, by Bateni, Esfandiari, Fischer, M.,

# Distributed Metric Clustering

**Clustering:** Divide data into groups containing “nearby” points



**Minimize:**

**$k$ -center :**  $\max_i \max_{u \in S_i} d(u, c_i)$

**$k$ -means :**  $\sum_i \sum_{u \in S_i} d(u, c_i)^2$

**$k$ -median :**  $\sum_i \sum_{u \in S_i} d(u, c_i)$

Metric space  $(d, X)$

$\alpha$ -approximation  
algorithm: cost less than  
 $\alpha^* \text{OPT}$

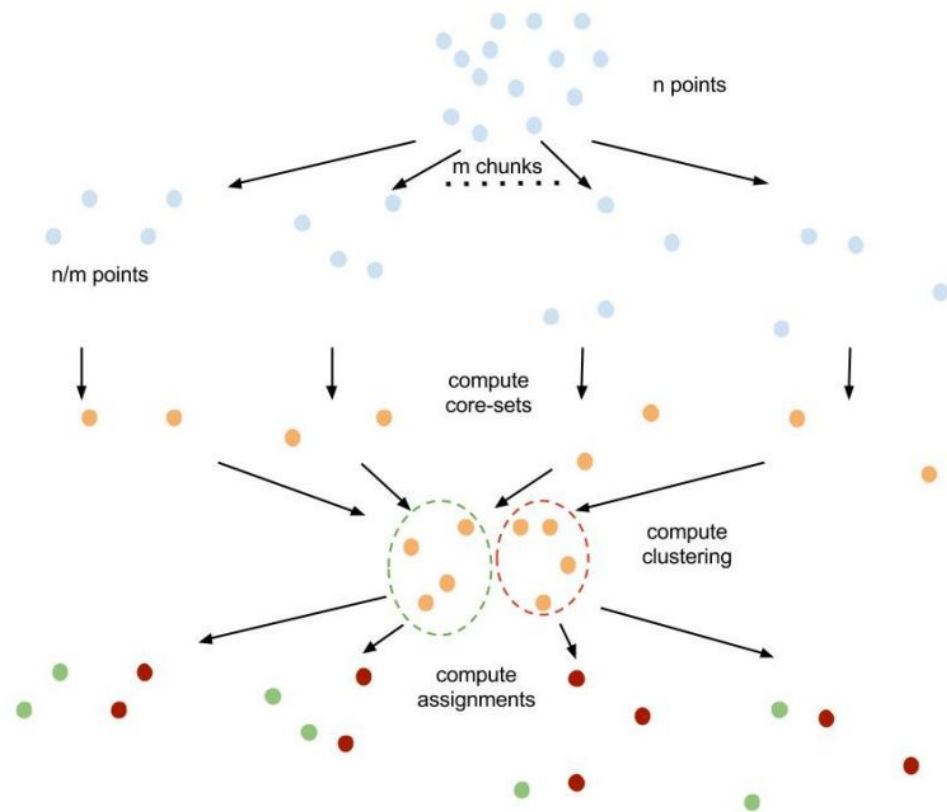
# Distributed Metric Clustering

## Core-set Framework:

- Divide into chunks  $V_1, V_2, \dots, V_m$ 
  - Random or using LSH
- Come up with “representatives”  $S_i$  on machine  $i$  with size  $\ll |V_i|$ .
- Solve on union of  $S_i$ , others by closest rep.

Balanced Clustering via Mapping Core-sets (Bateni, Bhaskara, Lattanzi, Mirrokni, NeurIPS'14)

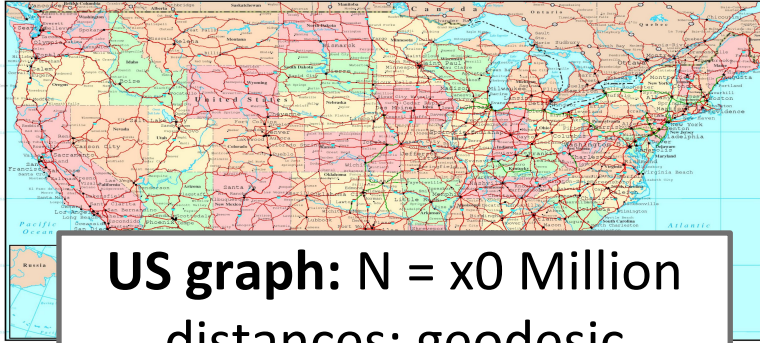
- Theoretical guarantee: 3 rounds, constant approximation.
- Empirical study → next slide



# Empirical Study

**Aim:** Test algorithm in terms of (a) scalability, and (b) quality of solution obtained

**Setup:** Two “base” instances and subsamples (used  $k=1000$ , #machines = 200)



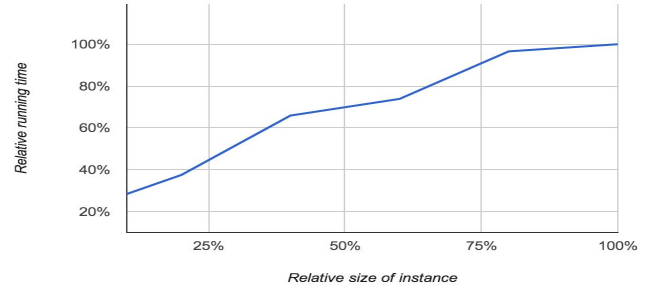
**US graph:**  $N = x0$  Million  
distances: geodesic



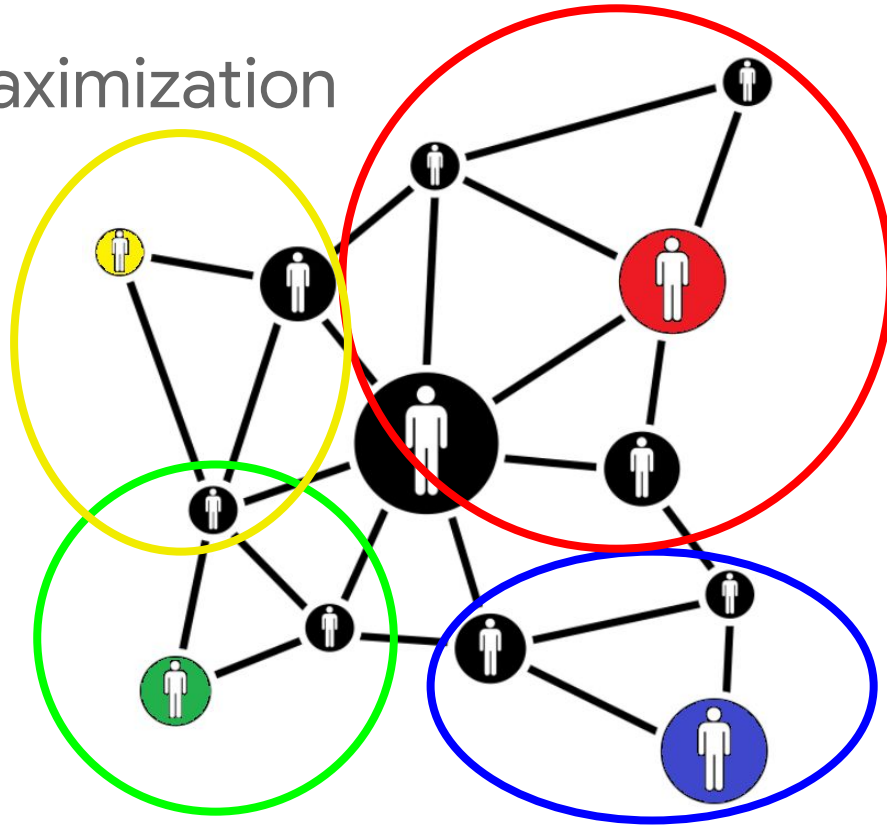
**World graph:**  $N = x00$  Million  
distances: geodesic

size of seq. inst.      increase in OPT

US	1/300	<b>1.52</b>
World	1/1000	<b>1.58</b>

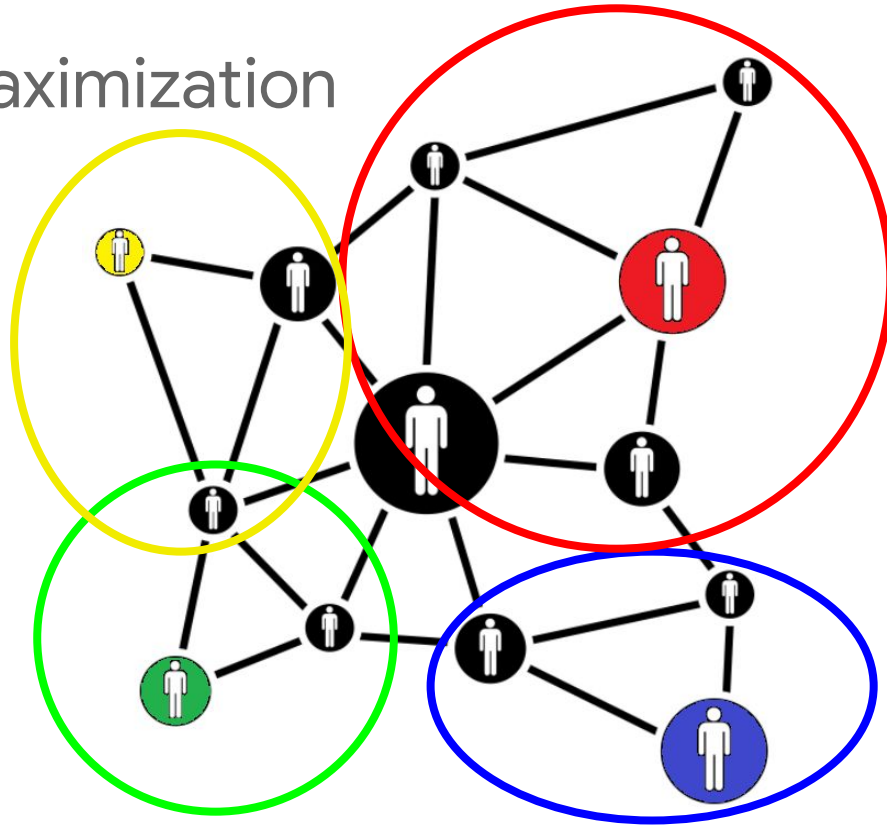


# Coverage maximization



Select  $k$  nodes to *cover* the maximum number of neighbors

# Coverage maximization



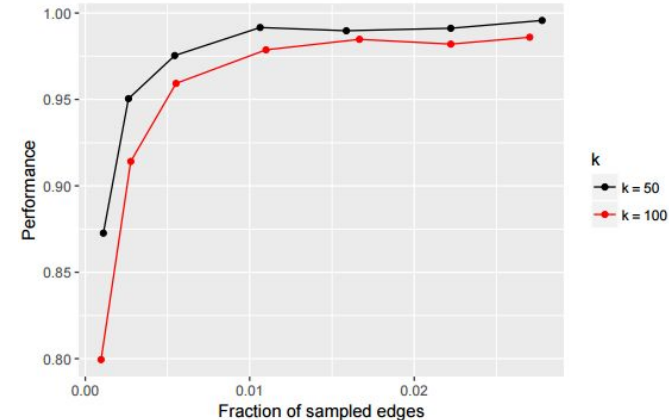
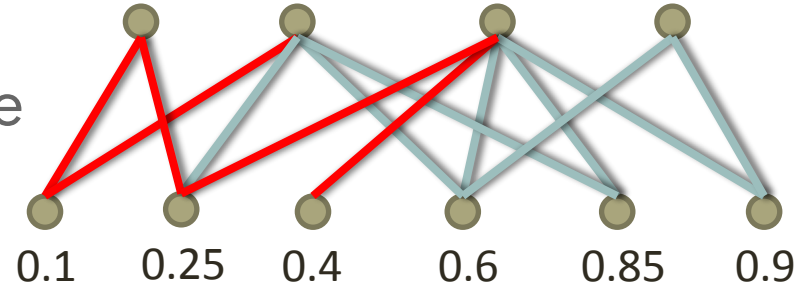
This extends k-center clustering for which we have an edge between every two point with optimal distance

Select  $k$  nodes to cover the maximum number of neighbors



# Coverage Maximization

- Given: A family of subsets  $S_1 \dots S_m$
- Goal: choose  $k$  subsets  $S'_1 \dots S'_k$  with the maximum union cardinality.
- Technique: Sketching + CoreSets
- Generate random numbers for items.
- Keep  $O(n)$  edges with minimum hash value but no more than  $O(n/k)$  per item.
  - Almost optimal approximation guarantees
  - Small sketches (0.01–3% of data) provide good approx (96%).
  - Bateni, Esfandiari, M., SPAA'17 and KDD'18.



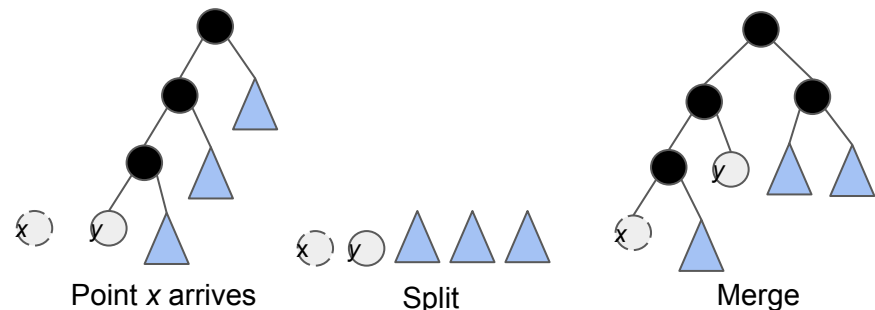
# Dynamic Distributed Clustering

We focused on batch/offline algorithms in this talk and this workshop, but we should highlight the need for dynamic/online distributed clustering:

## Online Hierarchical Agglomerative Clustering (OHAC)

- **Goal:** Maintain a hierarchy over a stream of points.
- **Algorithm:** When a new point arrives, run a split-merge procedure on the existing hierarchy.
  - **Split:** break the hierarchy into a forest.
  - **Merge:** run HAC on the forest and the new point.

Menon, Rajagopalan, Sumengen, Citovsky, Cao, Kumar: **Online Hierarchical Clustering Approximations**. Arxiv abs/1909.09667 (2019)



**Next, we need to combine distributed, consistent, & dynamic algorithms?** Work in progress, e.g.,

Italiano, Lattanzi, Mirrokni, Parotsidis, **Dynamic Algorithms for the Massively Parallel Computation Model**. SPAA'19

Fichtenberger, Lattanzi, Norouzi-Fard, Svensson: **Consistent k-Clustering for General Metrics**, SODA'21.



# Conclusions: Summary of Algorithmic Techniques

- Clustering is one of the most popular tools in the library.
- Challenge: Scalability
- Techniques: Message Passing, Random Walks, Sketching and Locally Sensitive Hashing(LSH), Composable Core-sets, ...
  
- Many variants, e.g., we didn't cover overlapping clustering here. Related to ego-Nets and also label propagation discussed later ...
- Many objectives: Next talk covers this in the context of community detection.

# Further Reading

We focused on batch/offline clustering algorithms. Examples of recent papers not covered:

Bressan, Cesa-Bianchi, Lattanzi, Paudice, **Exact Recovery of Mangled Clusters with Same-Cluster Queries**, NeurIPS'20.

Heinrich Jiang, Jennifer Jang, Kuba Lacki, **Faster DBSCAN via subsampled similarity queries**, NeurIPS'20.

Menon, Rajagopalan, Sumengen, Citovsky, Cao, Kumar: **Online Hierarchical Clustering Approximations**. Arxiv'19.

Italiano, Lattanzi, Mirrokni, Parotsidis, **Dynamic Algorithms for the Massively Parallel Computation Model**. SPAA 2019

Ghaffari, Lattanzi, Mitrovic: **Improved Parallel Algorithms for Density-Based Network Clustering**. ICML'19.

Lattanzi, Sohler: **A Better k-means++ Algorithm via Local Search**. ICML'19

Monath, Dubey, Guruganesh, Zaheer, Ahmed, McCallum, Mergen, Najork, Terzihan, Tjanaka, Wang, Wu: **Scalable Bottom-Up Hierarchical Clustering**, Arxiv'20.

Fichtenberger, Lattanzi, Norouzi-Fard, Svensson: **Consistent k-Clustering for General Metrics**, SODA'21.





# Community detection

Jakub Łącki

Collaborators: Vahab Mirrokni, Christian Sohler

# Community detection

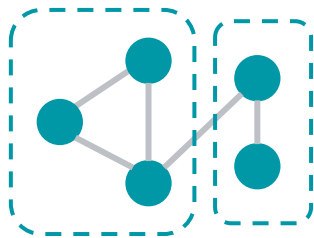
## Problem

**Find communities  
in a social network**

## Method

**Cluster graph  
nodes into densely  
connected subsets**

# Desirable properties



## Output

Clusters  $\leftrightarrow$  ground truth communities

Dense, sparsely connected clusters

Large number of small clusters



## Algorithm

Scalable

Provable running time and quality guarantees

Automatically detected number of clusters

# Quality metrics & algorithms

## Metrics

### Modularity

[NewmanG, PRE'04]

### Conductance

### Normalized cut

[ShiM, PAMI'00]

### Density

### Cut sparsity

## Algorithms

### Spectral

[ShiM, PAMI'00]

### MCL

[EnrightDO, NAR'02]

### Infomap

[RosvallB, PNAS'08]

### Louvain

[BlondelGLL, JSTAT'08]

### Leiden

[TraagVE, Nature'19]

### Motif-based

[BensonGL, Science'16]  
[TsourakakisPM, WWW'17]



# Modularity

*Finding and evaluating community structure in networks.* Newman, Girvan, PRE 2004

- $\text{deg}(x)$  = degree of node  $x$
- $C(x)$  = cluster of node  $x$
- $m$  = number of edges
  
- **Modularity** of a clustering:

$$\frac{1}{2m} \sum_{\substack{xy \in E \\ C(x) = C(y)}} 1 - \frac{\text{deg}(x)\text{deg}(y)}{2m}$$

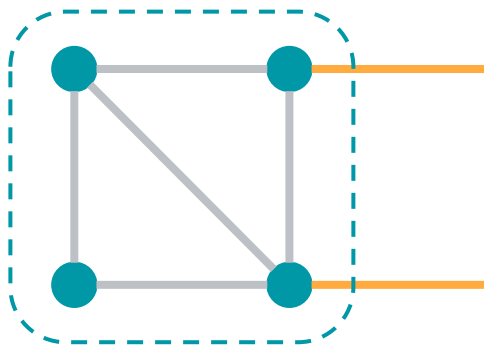
Algorithm:

- Louvain method - greedy approach
- Very effective in practice, but little theoretical guarantees

# Conductance and normalized cut

**Conductance\*** of a cluster  $C$ :

$$\phi(C) = \frac{\text{number of edges leaving } C}{\text{total degree of nodes in } C}$$



Total degree inside cluster =  $3 + 2 + 3 + 2 = 10$   
 Conductance =  $2 / 10 = 0.2$

**Normalized cut** of a clustering: sum of cluster conductances

# Coconductance - definition

Ł., Mirrokni, Sohler, ongoing work

- $p > 0$  - parameter (canonical setting is  $p = 1$ )
- Coconductance of a clustering:

$$\sum_{\text{cluster } C} (1 - \varphi(C))^p$$



## Coconductance clustering

- **Maximize** total coconductance
- For  $p = 1$ , closely related to normalized cut

# Coconductance - algorithms

**Practical algorithm**

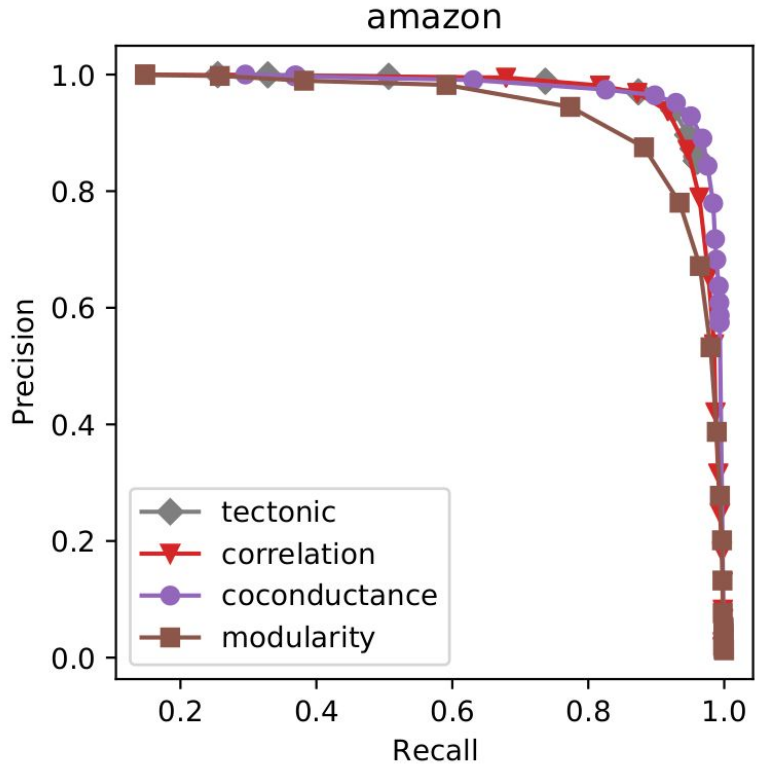
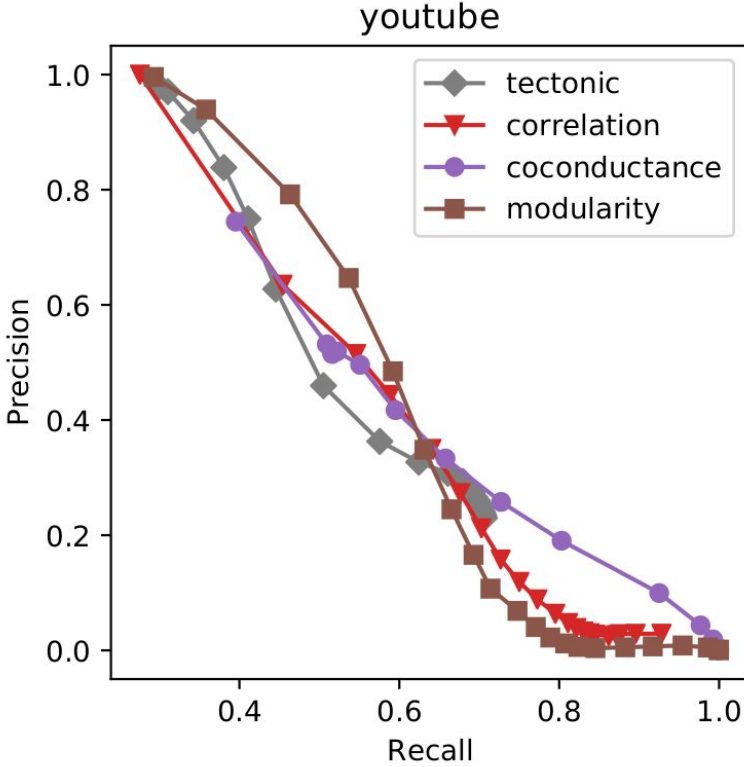
**Adaptation of the  
Louvain method**

**Good empirical  
quality**

**Theoretical algorithm ( $p=1$ )  
Constant approximation of the  
optimal solution**

**Linear time**

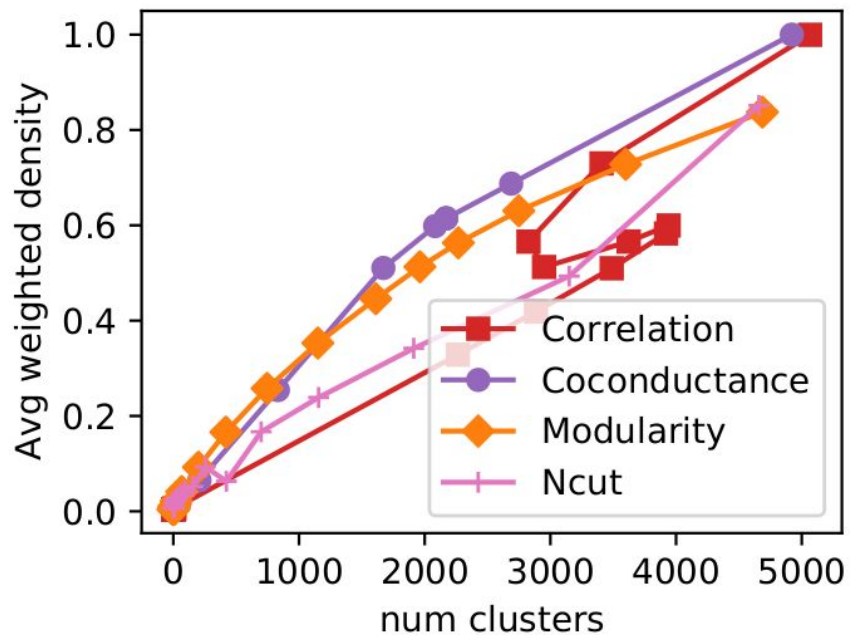
# Coconductance - empirical results



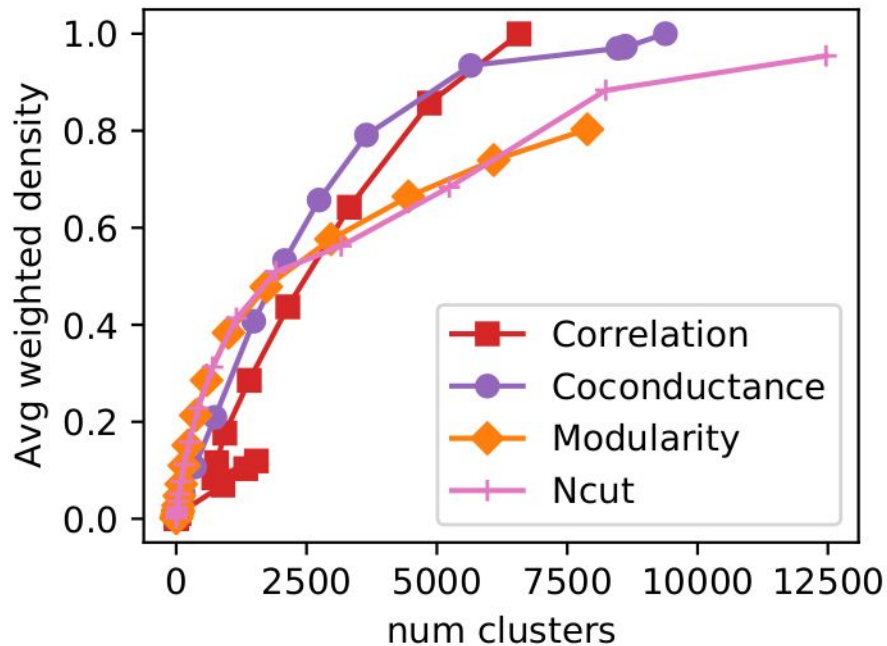
Datasets from SNAP, methodology from [TsourakisPM, WWW'17]

# Coconductance - empirical results

wiki-Vote

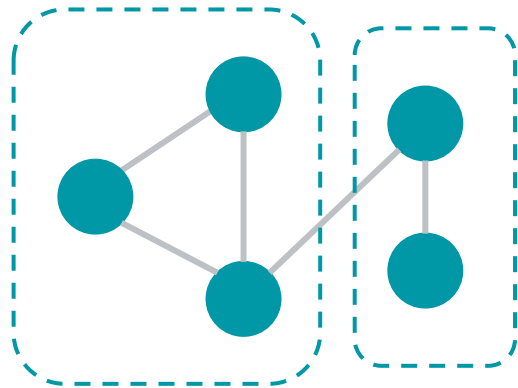


ca-AstroPh



# Summary

- Active area of research
- Many existing algorithms / metrics
  - No “one-size-fits-all” solution
- New algorithm / metric: co-conductance





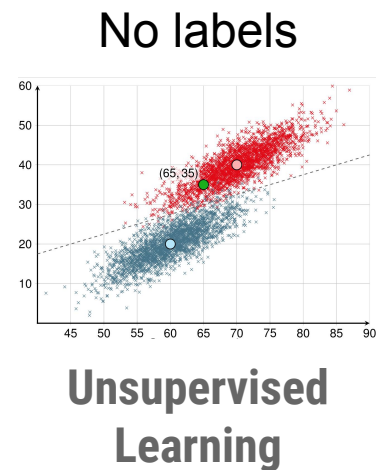
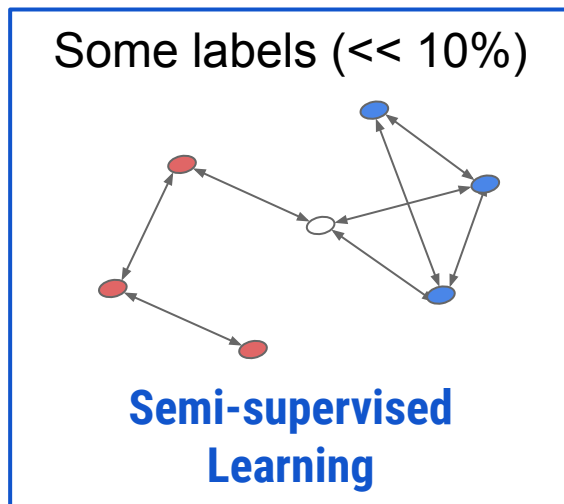
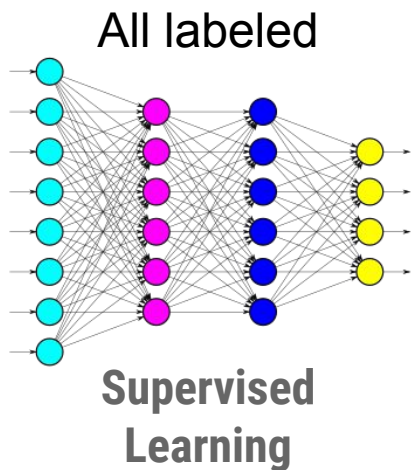
# Label Propagation

Allan Heydon



# Semi-Supervised Learning (SSL)

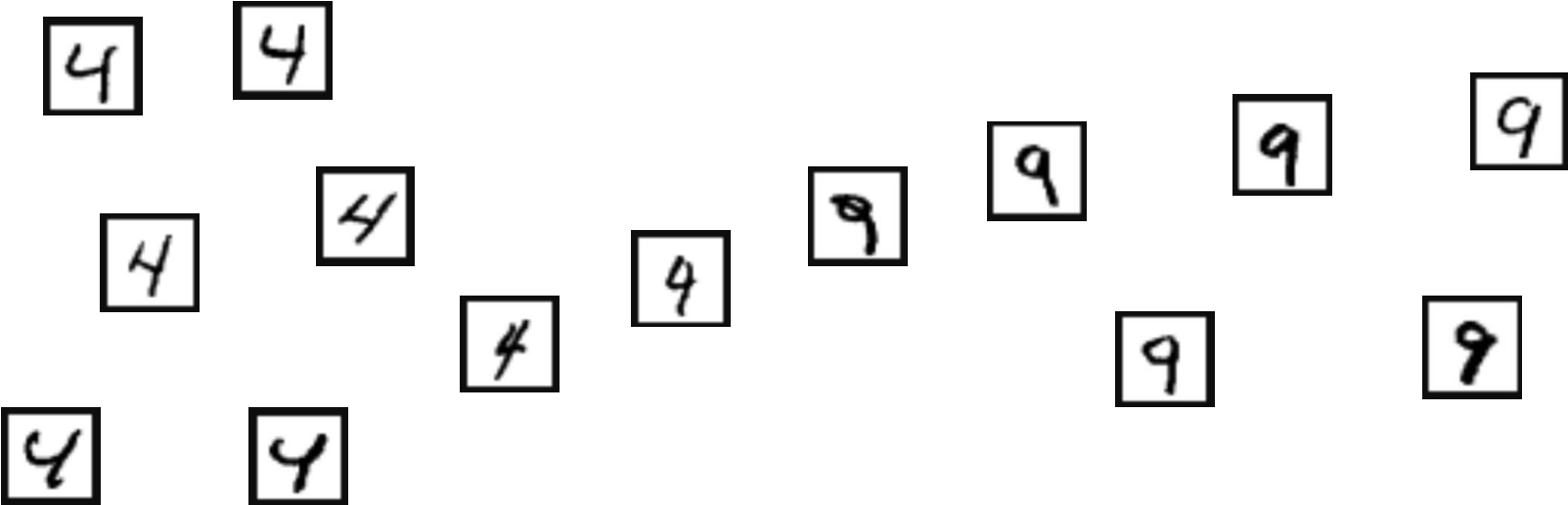
Different approaches based on the amount of labeled data:



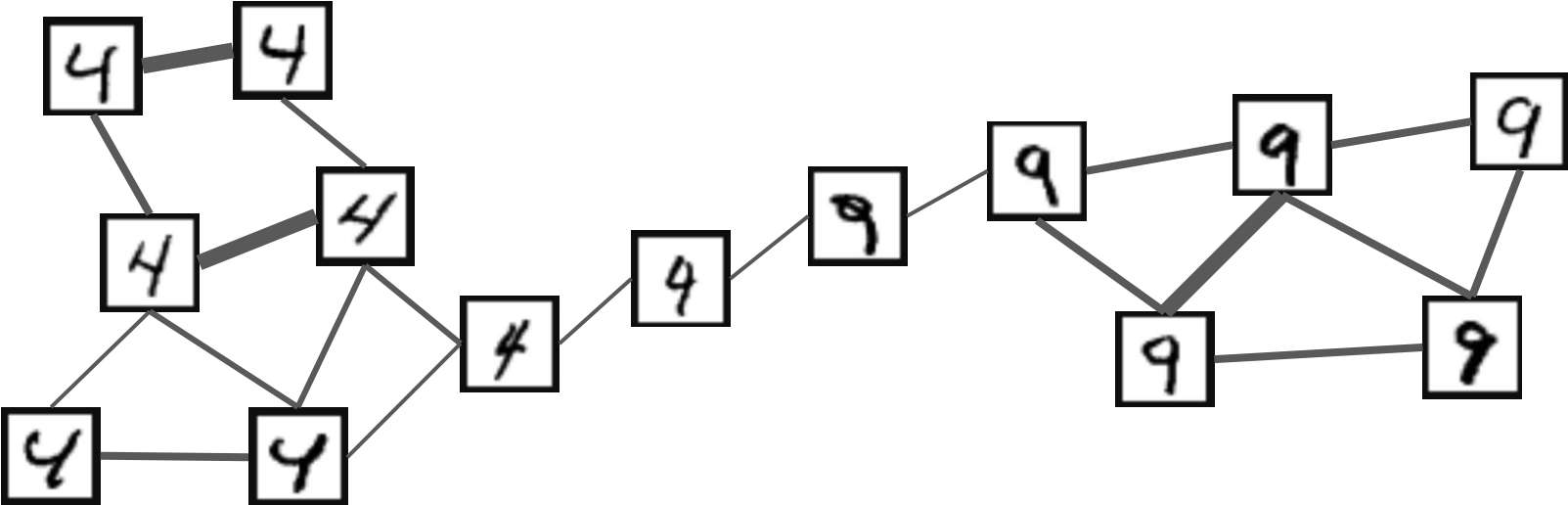
# Similarity Graphs

- Goal: Learn labels for unlabeled instances using *context*.
- Leverage **similarity relationships** between instances!
  - “Similar instances should have similar learned labels.”
  - Graph can be based on *natural* relationships or *computed* from node features.
- Landmark paper:
  - [\*Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions\*](#), X. Zhu, Z. Ghahramani, J. Lafferty, Proc of ICML-2003, Aug, 2003.
  - Solved using matrix operations, which don't scale well.
- Idea: Iteratively propagate labels along graph edges.

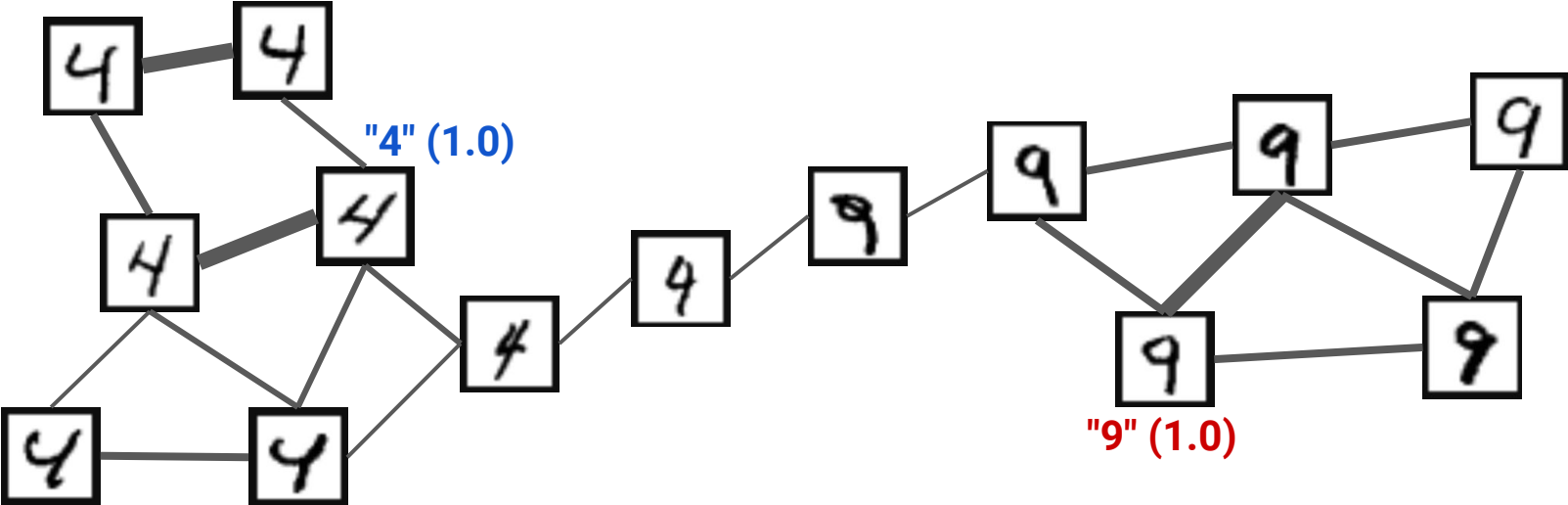
# SSL Example - Data Instances



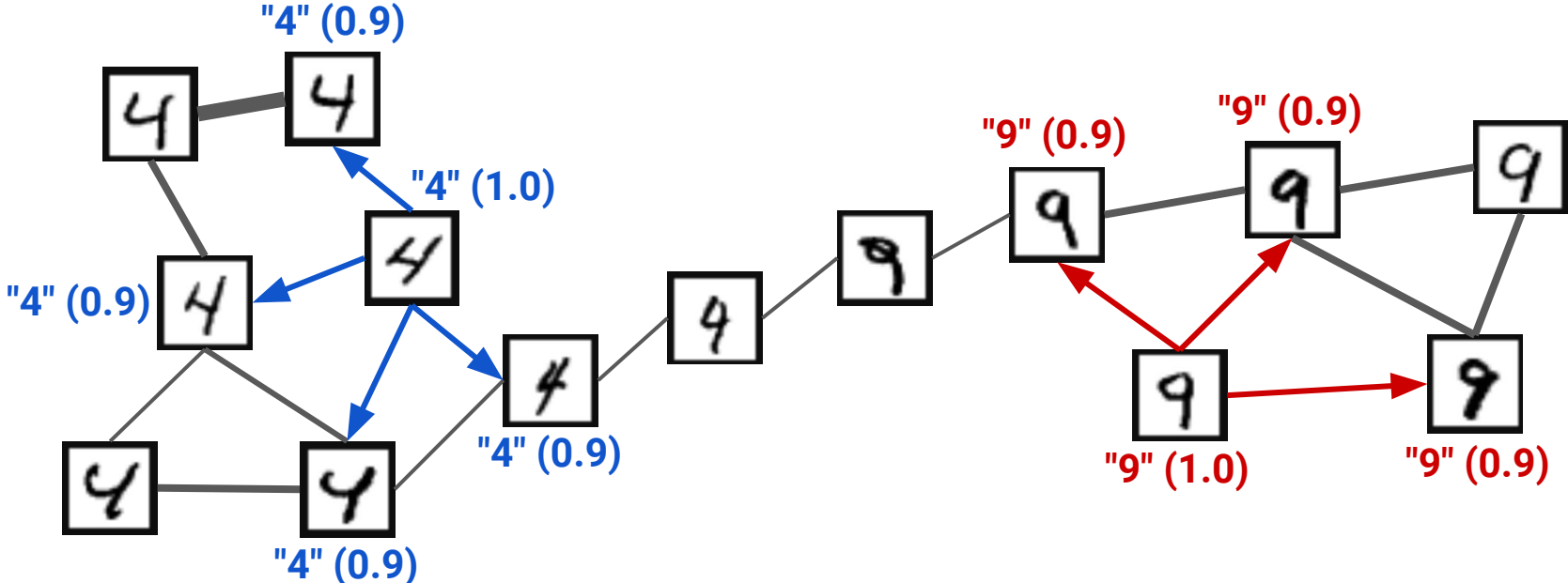
# SSL Example - Add Similarity Graph Edges



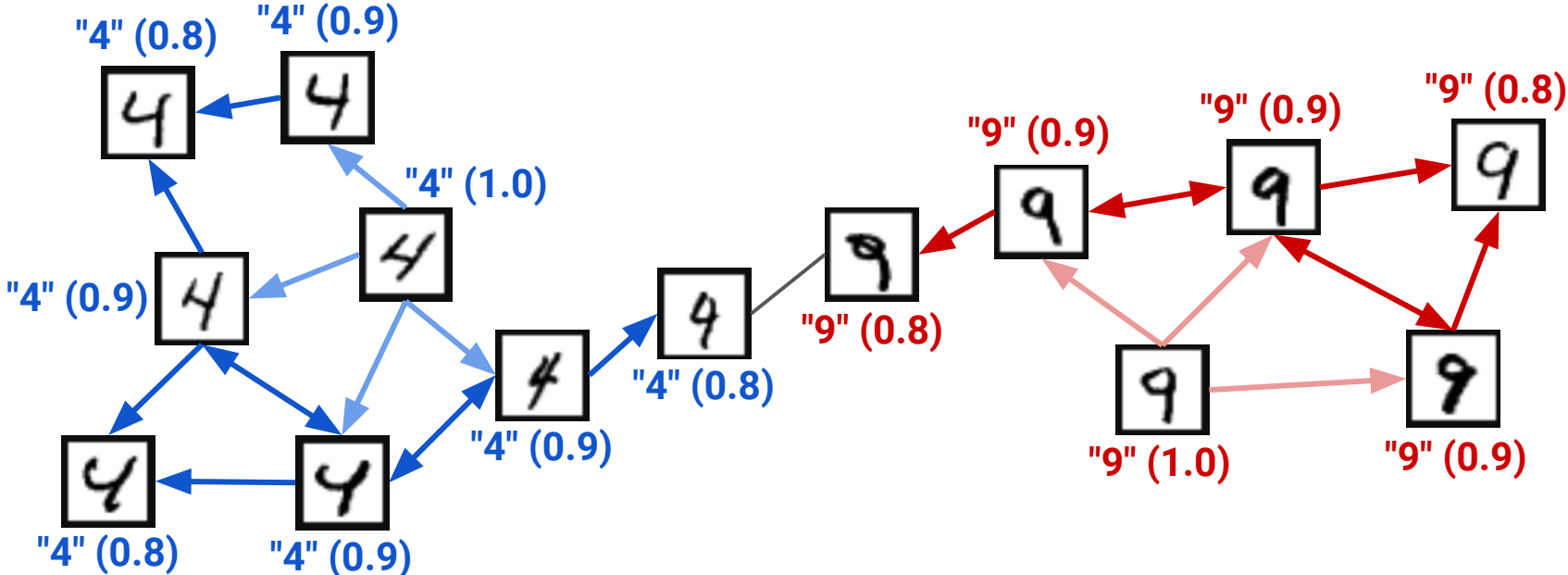
# SSL Example - Add "Seed" Labels



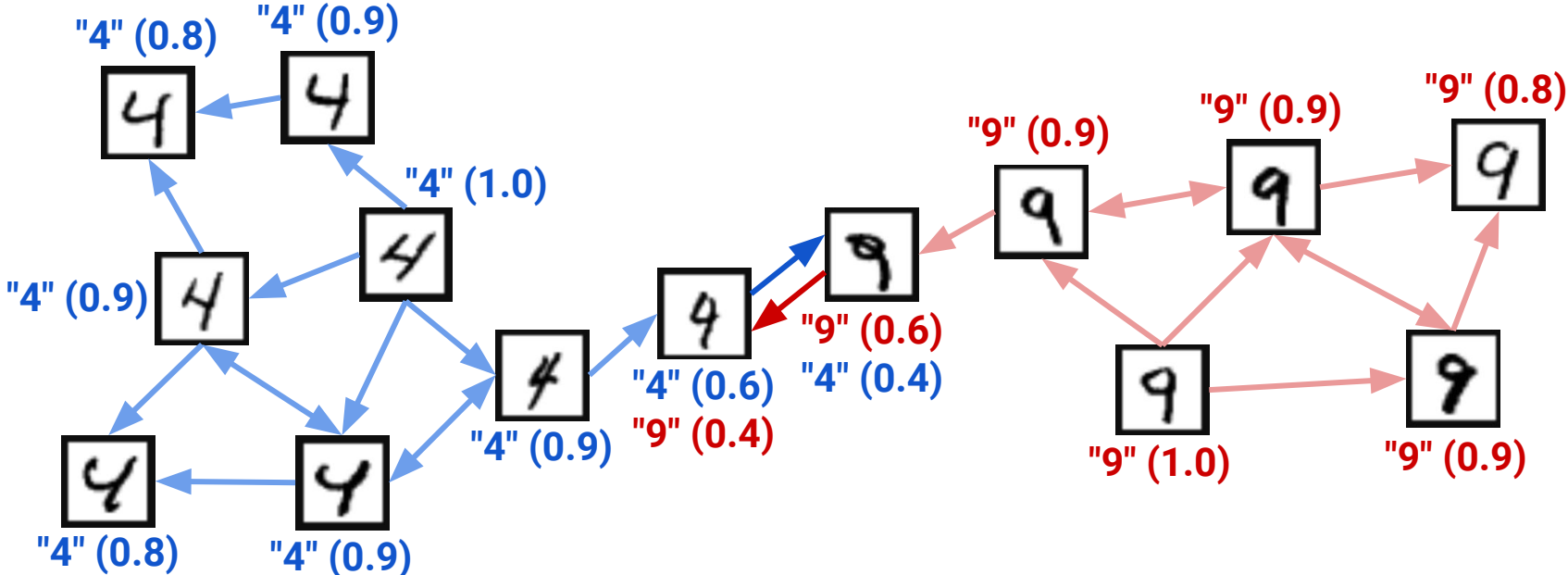
# SSL Example - Iteration 1



# SSL Example - Iteration 2



# SSL Example - Iteration 3





# Label Propagation API



- **Input:**
    - Similarity signal (weighted edges)
    - Training labels/weights ("seed" vertices)
    - Test labels/weights ("validation" vertices)
- } Labeled Vertices
- **Output:**
    - Learned labels for most/all vertices.
    - Thresholding is typically applied to select "strong" learned labels.

# SSL Applications

Generality of the framework permits a variety of applications:

- Spam and abuse detection (typically binary classification).
- Multi-class text and video classification.
- Identification of incorrect noisy labels to enable *label cleaning*.
- Natural language processing, e.g., sentiment and emotion detection, improving recall by identifying synonymous phrases.
- Augmentation of label data for downstream model training.

# System Properties

- **General**
  - Nodes can be of the same (homogeneous) type or of different (heterogeneous) types.
  - Graph edges can represent arbitrary similarity relationship(s) (with different types).
  - Node IDs and seed/validation labels are arbitrary strings.
- **Flexible**
  - Handles *binary*, *multi-class*, and *multi-label* problems.
  - Supports both positive and negative input label weights.
  - Pluggable vertex propagation algorithm/class (defaults to weighted average).
  - Can be used as part of a larger machine-learning pipeline.
- **Scalable**
  - Scales to XT edges, XXXB nodes, XXXM distinct labels.
  - Implemented as a massively parallel computation involving XK machines.
  - Optimization: keep the top K labels per node on each iteration.

# Label Update Function

- Learn by propagating labels over the graph.
- Iterative algorithm attempts to minimize the following objective function [1]:

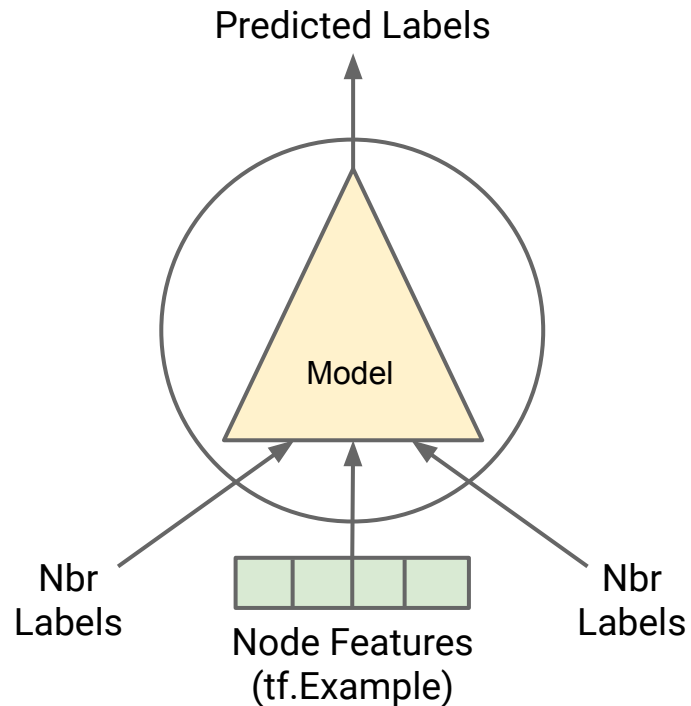
$$\begin{aligned}
 & \text{Labeled loss} && \text{Neighbor loss} && \text{Prior loss} \\
 & \left( s_v \sum_{l \in L} \|\widehat{Y}_{v,l} - Y_{v,l}\|^2 \right) + \left( \mu_{np} \sum_{u \in N(v)} w_{u,v} \sum_{l \in L} \|\widehat{Y}_{v,l} - \widehat{Y}_{u,l}\|^2 \right) + \left( \mu_{pp} \sum_{l \in L} \|\widehat{Y}_{v,l} - U_l\|^2 \right) \\
 & \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \\
 & \text{Seed penalty} \quad \text{Learned weight} \quad \text{Seed weight} \quad \text{Neighbor penalty} \quad \text{Edge weight} \quad \text{Prior penalty} \quad \text{Prior label} \\
 & (1.0 \text{ for seeds})
 \end{aligned}$$

[1] "Large Scale Distributed Semi-Supervised Learning Using Streaming Approximation", S. Ravi and Q. Diao, Proceedings of AISTATS, May, 2016.

## Learned Label Update Function [2]

Idea: Train a model to exploit node **features**.

- Label update function inputs:
  - Neighbor labels.
  - Node features (tensorflow.Example)
- Leverages the power of non-linear models: tree-based models, DNNs.
- Model training:
  - Run LP to generate training data.
  - Train model on labeled (seed) nodes.



[2] "Collective Classification in Network Data", P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, *AI Magazine*, 29(3), Sept, 2008, pages 93-106, <https://doi.org/10.1609%2Faimag.v29i3.2157>.

# Conclusions

## Label propagation:

- Is a *semi-supervised* learning technique requiring  $\ll$  10% of nodes to be labeled
- Leverages a *similarity graph* to propagate labels between neighbors
- *Scales* to very large graphs and large label spaces and
- Can be applied to a wide variety of problem types.
- Is available publicly as a Google Cloud [AI Workshop experiment](#).

Google AI Blog post: [Graph-powered Machine Learning at Google](#)

# Graph Neural Networks

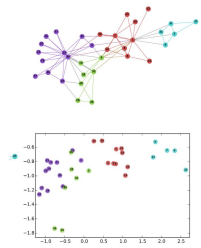
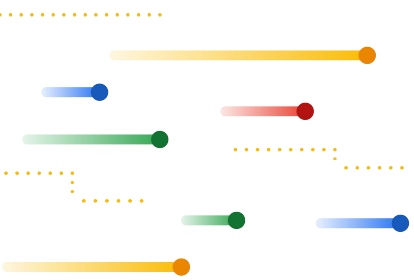
Bryan Perozzi, Amol Kapoor,  
John Palowitch, Alessandro  
Epasto

Graph Neural Networks and  
Graph Embeddings

PPRGo: GNNs at Scale

Debiasing GNNs

Learning Multiple Embeddings



# Graph Embeddings and Graph Neural Networks

Bryan Perozzi



# Section Overview

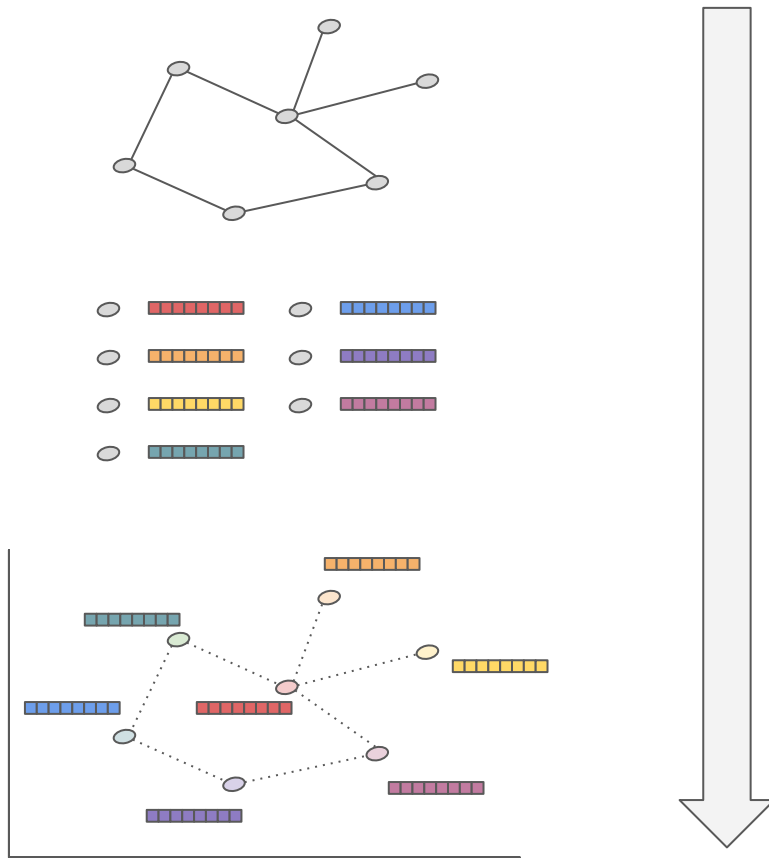
1. Graph Embeddings
2. Graph Convolutions
3. Challenges of GNNs

# Graph Embedding

An embedding is a high dimensional float-vector representation of information, often generated by the inner layer of a deep neural network.

Embedding intuition: the *information* is the same, the *representation* is different, e.g. rgb vs cmyk, or encryption.

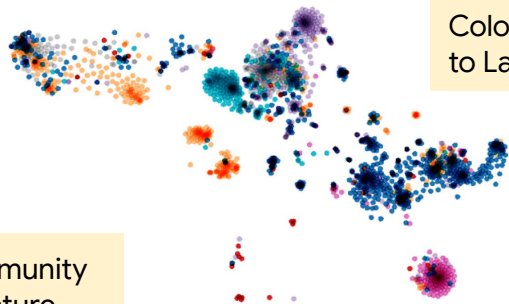
A graph embedding is simply a representation of graph data. The high dimensional graph information (structure/features) are mapped to a lower dimensional space.



# Graph Embedding

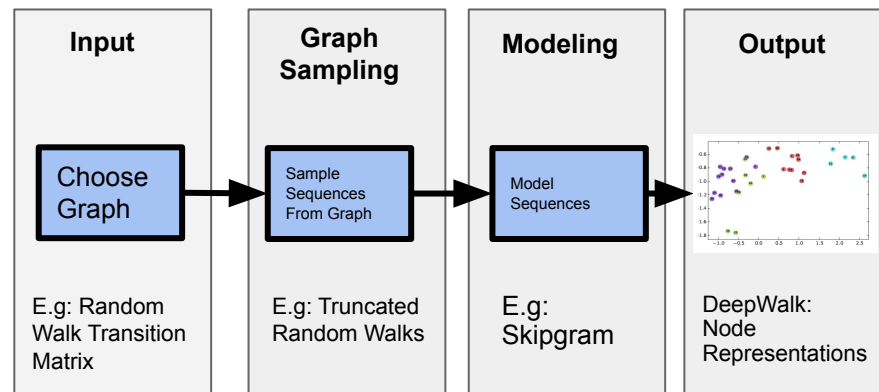
Initial work focused on using random walk reconstruction for unsupervised representation learning.

Emphasis on faithful encoding of source graph & communities.



Colors correspond to Labels

Community Structure Preserved



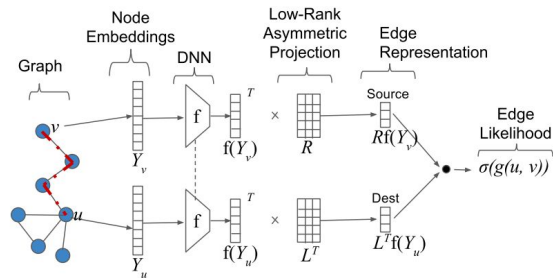
DeepWalk Paradigm for Graph Representation Learning

# Many extensions since

## 1. Directed Graphs

Learning Edge Representations via Low-Rank Asymmetric Projections

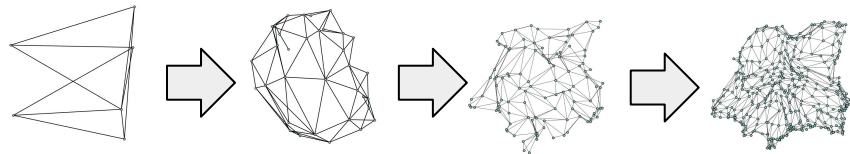
*S Abu-El-Haija, B Perozzi, R Al-Rfou (CIKM'17)*



## 2. Hierarchical Structure

HARP: Hierarchical Representation Learning for Networks

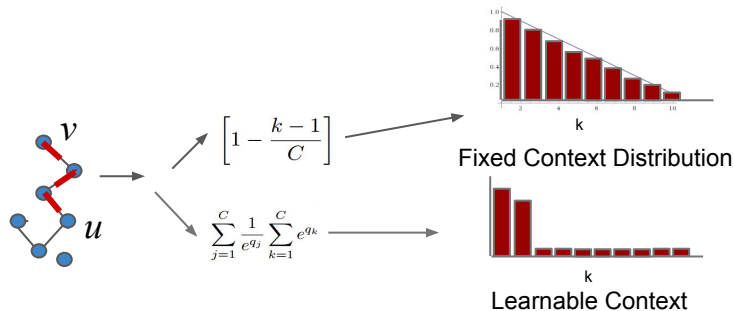
*H Chen, B Perozzi, Y Hu, S Skiena (AAAI'18)*



## 3. Graph Attention Models

Watch Your Step: Learning Node Embeddings via Graph Attention

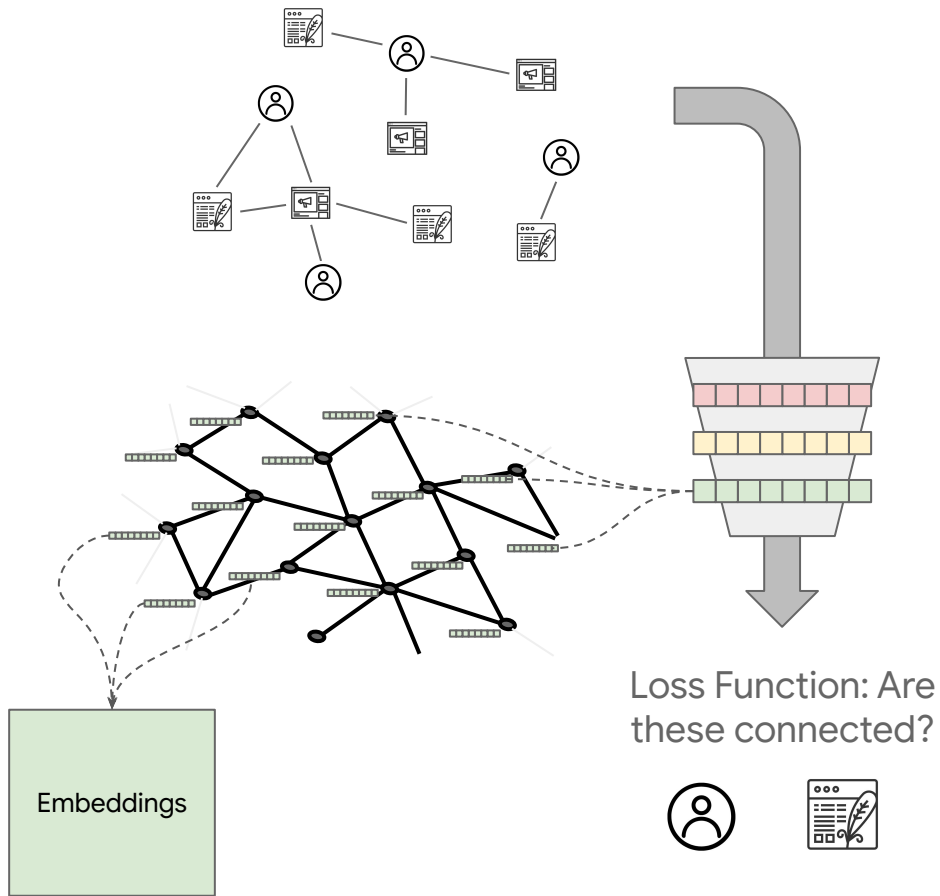
*S Abu-El-Haija, B Perozzi, R Al-Rfou, AA Alemi (NeurIPS'18)*



# Graph as a Modality

Graph embeddings are the foundation of using graphs as a data modality (like images), because they allow us to store, compare, and reason about information coming from many domains.

We can even do this for graphs which have complex, heterogeneous, structure.

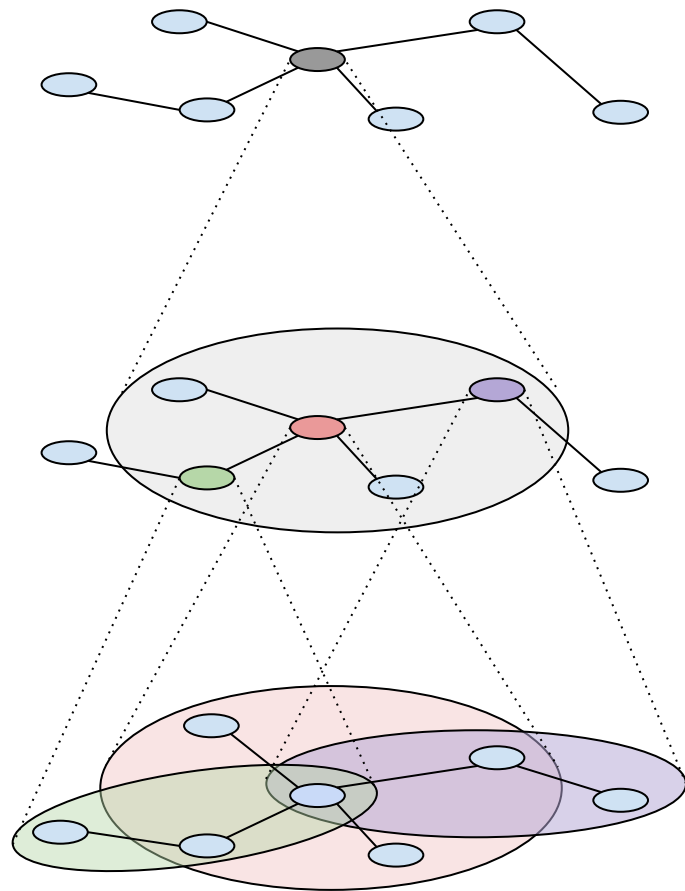


# GCNs

Graph Convolutional Networks are a way to apply deep learning to local networks within arbitrary graph structures.

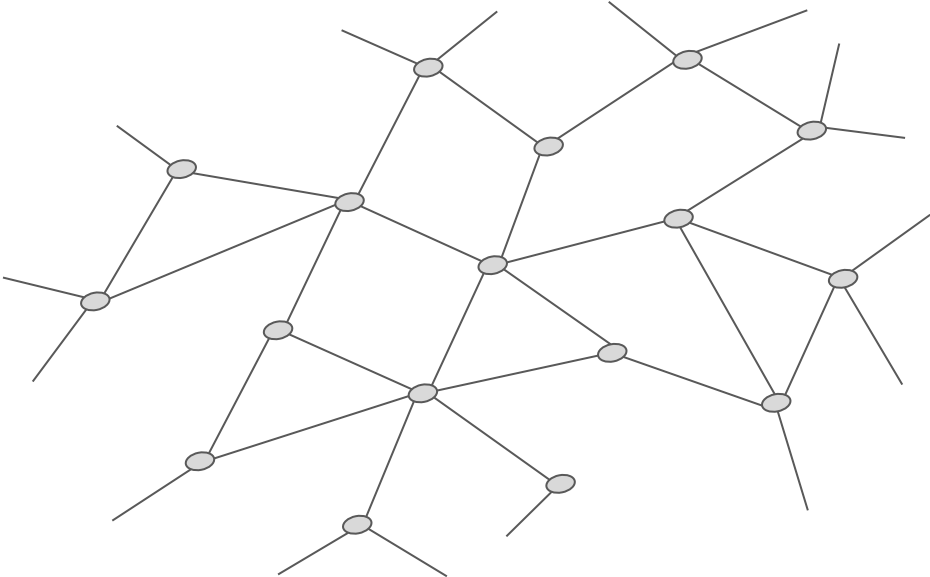
Inspired by convnets: we want to incorporate context!

But non-trivial to do scalably...Tensorflow does not like dynamically shaped inputs :(





# Graph (a patch of a graph)

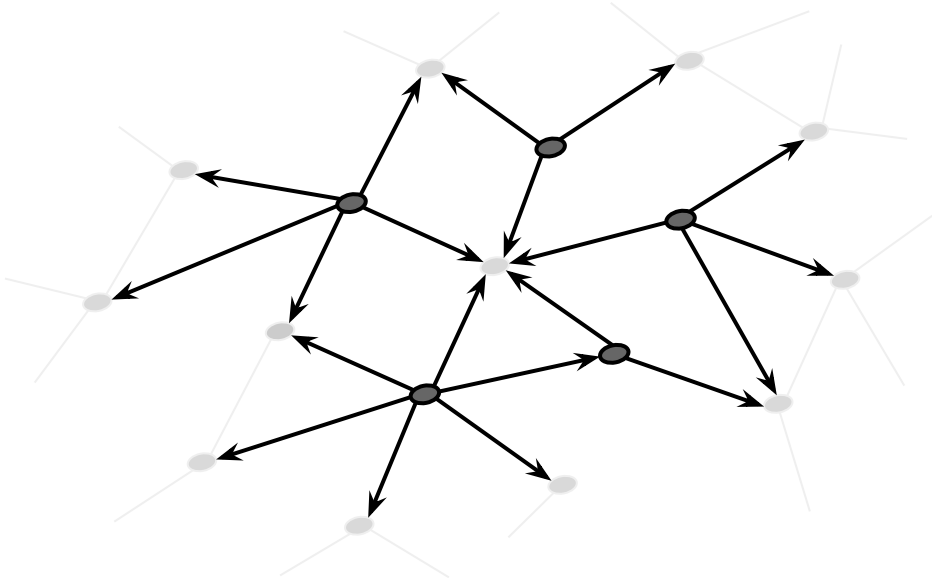




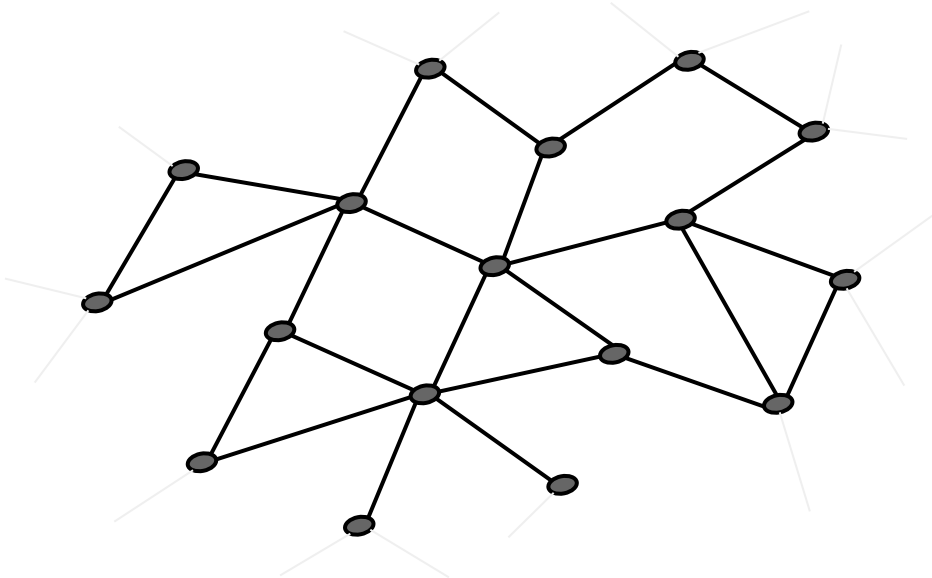
Seed node



# 1-hop Neighborhood

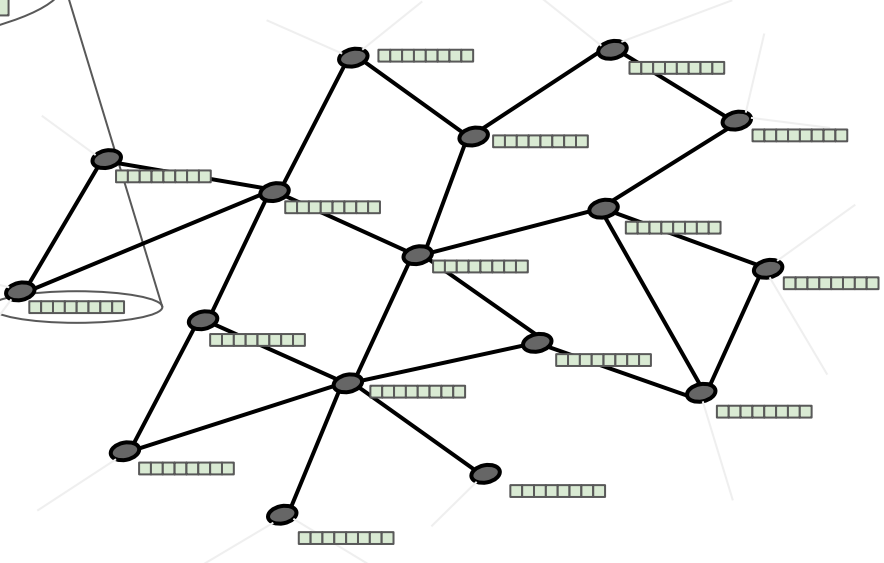


# 2-hop Neighborhood

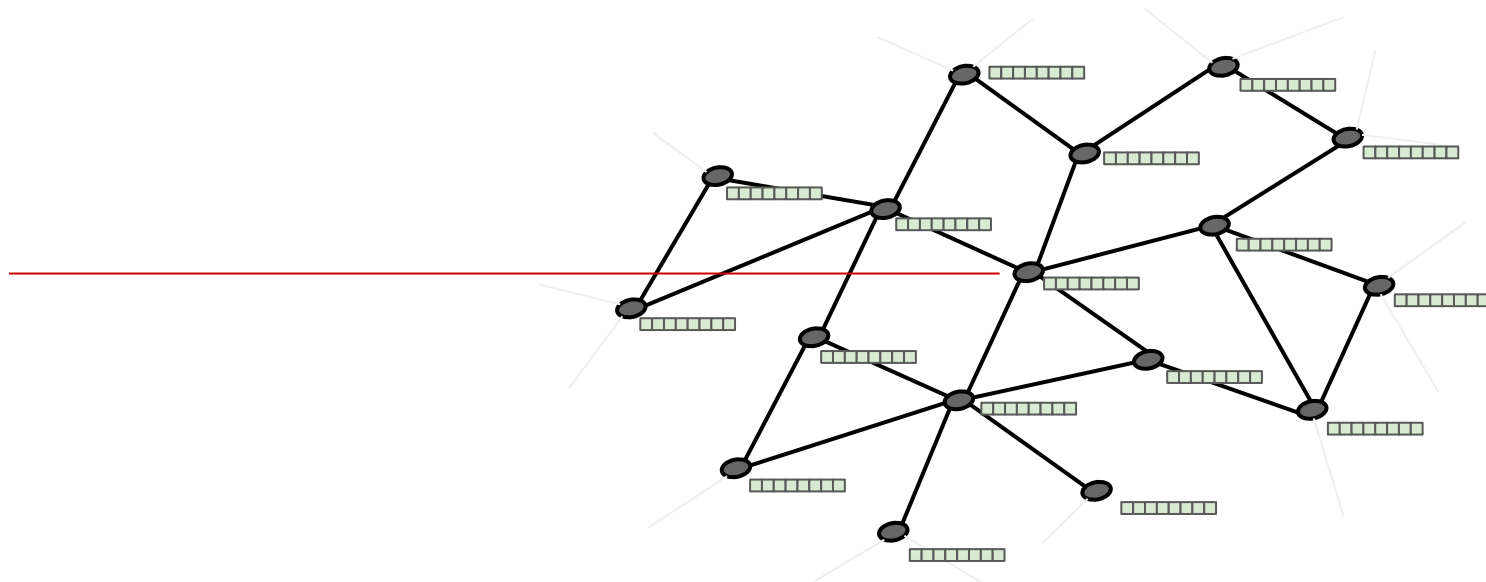


# Features

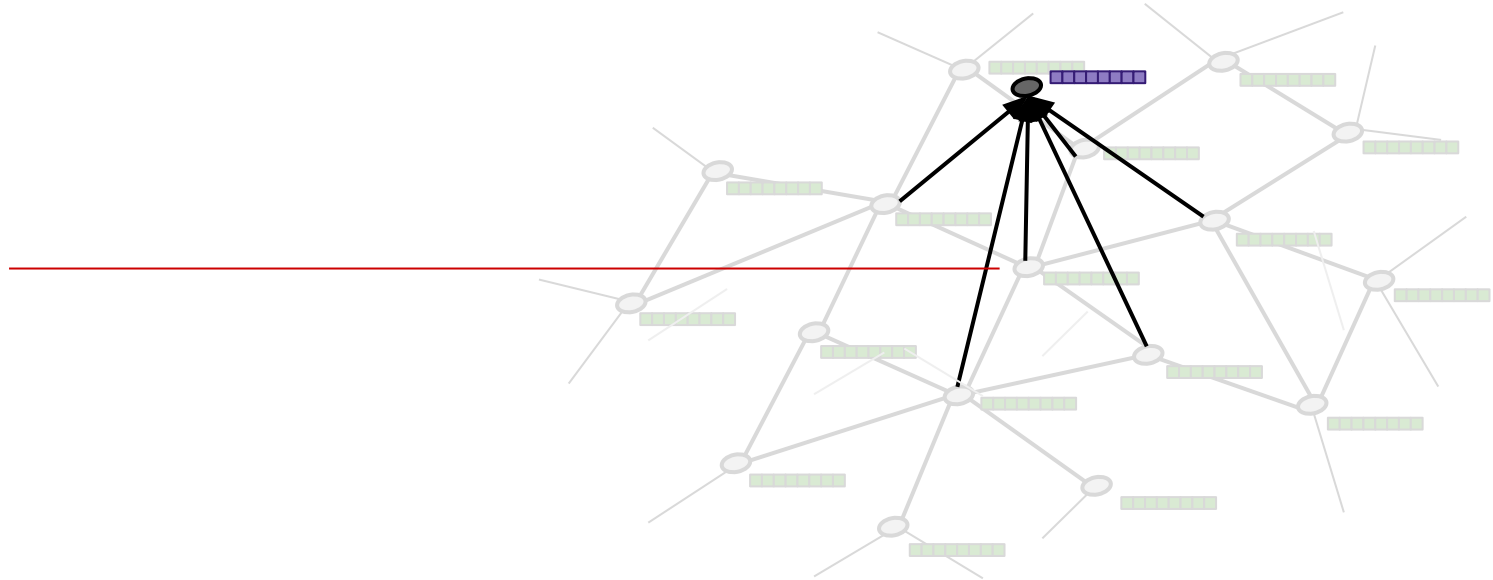
1.2	0.4	4.6	2.1	0.0	0.1	9.4	5.3
-----	-----	-----	-----	-----	-----	-----	-----



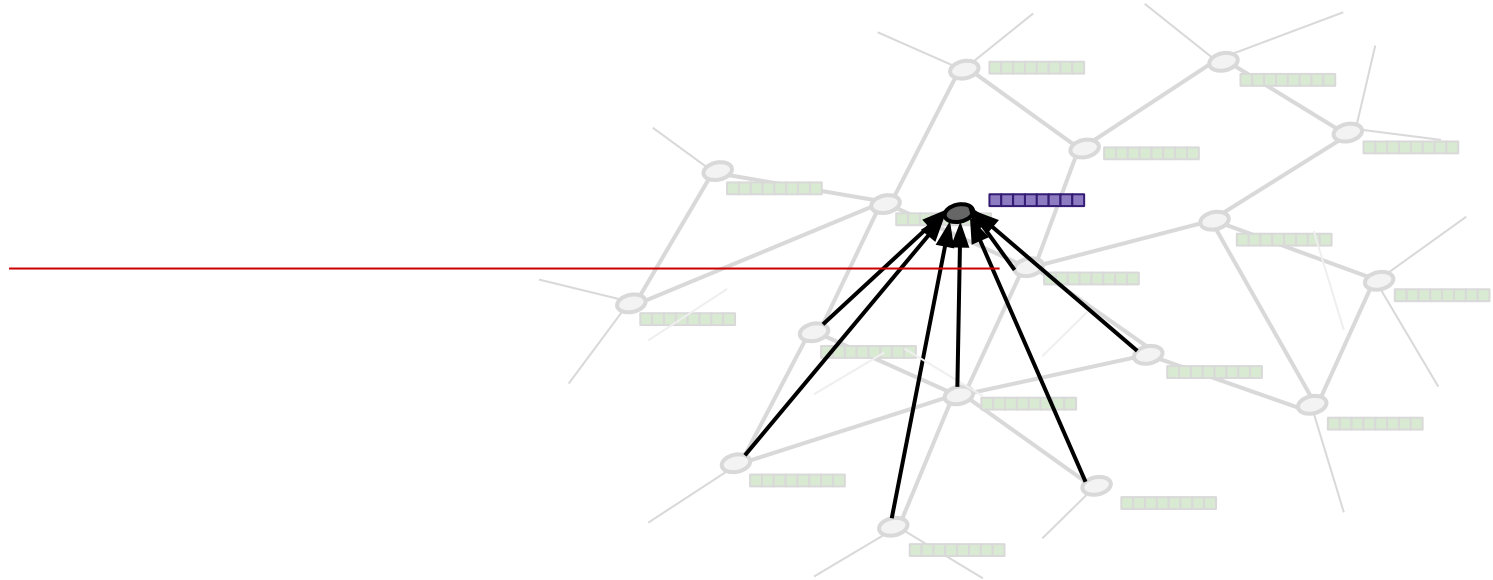
# A Graph Convolution



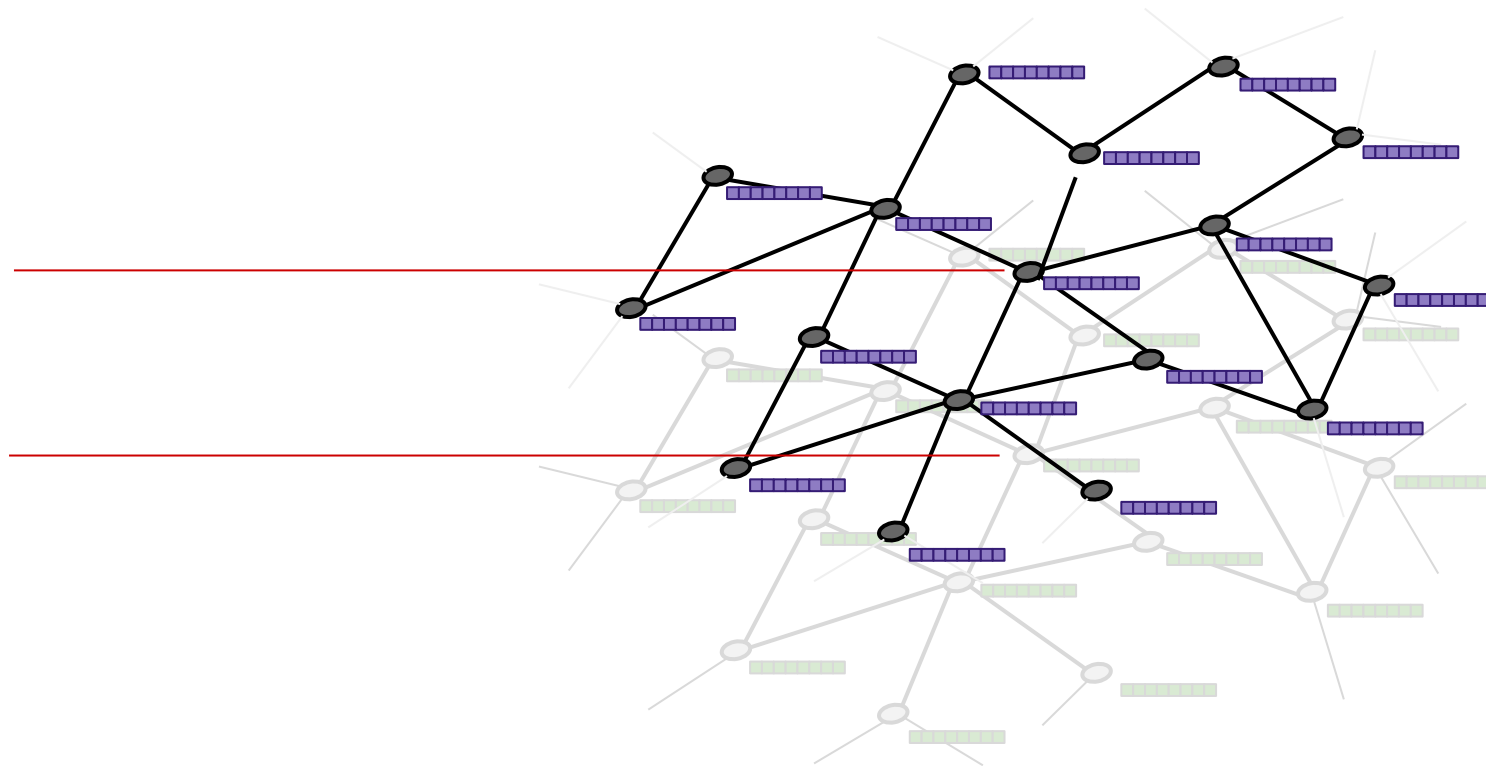
# A Graph Convolution



# A Graph Convolution

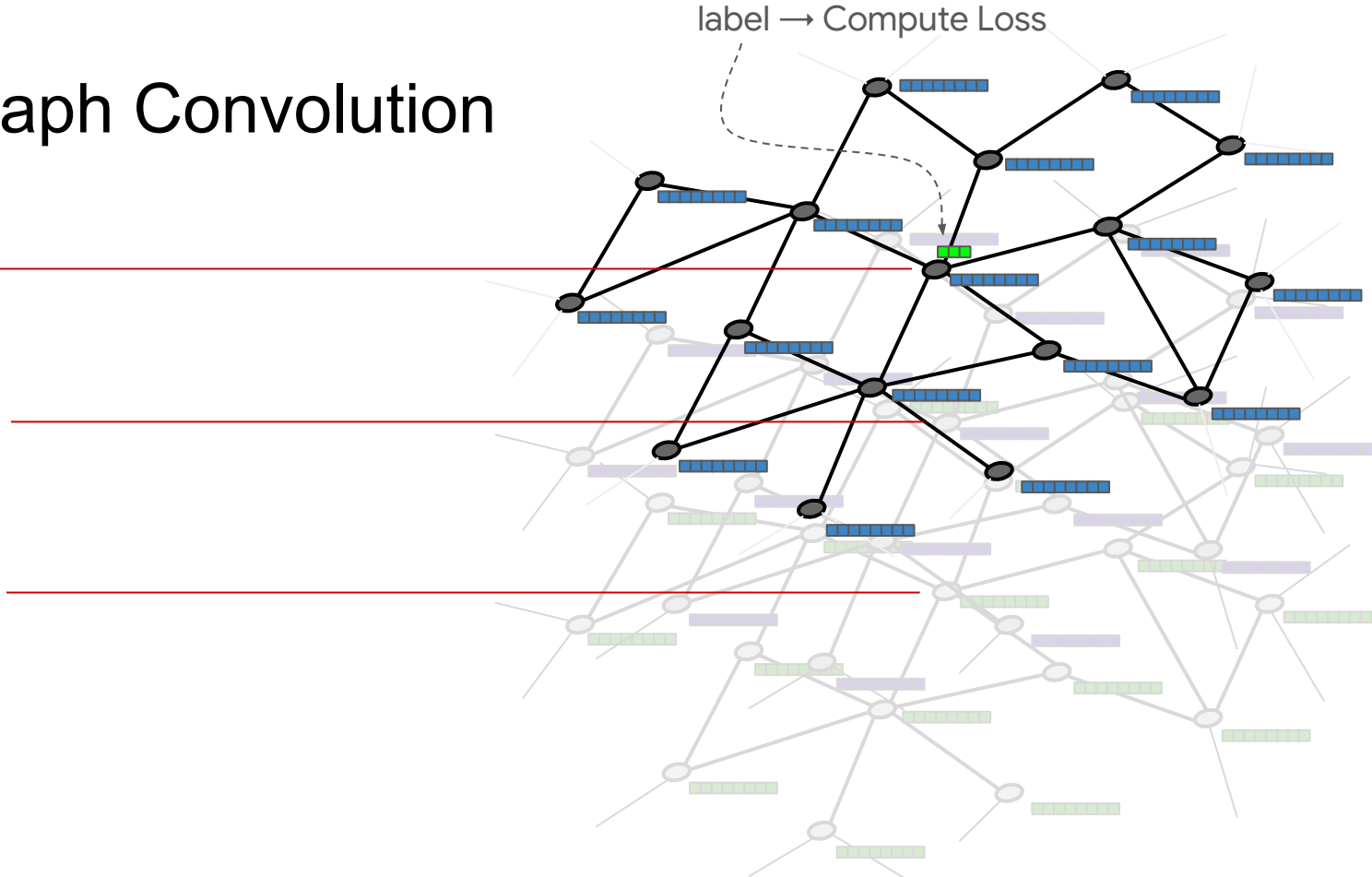


# A Graph Convolution





# A Graph Convolution

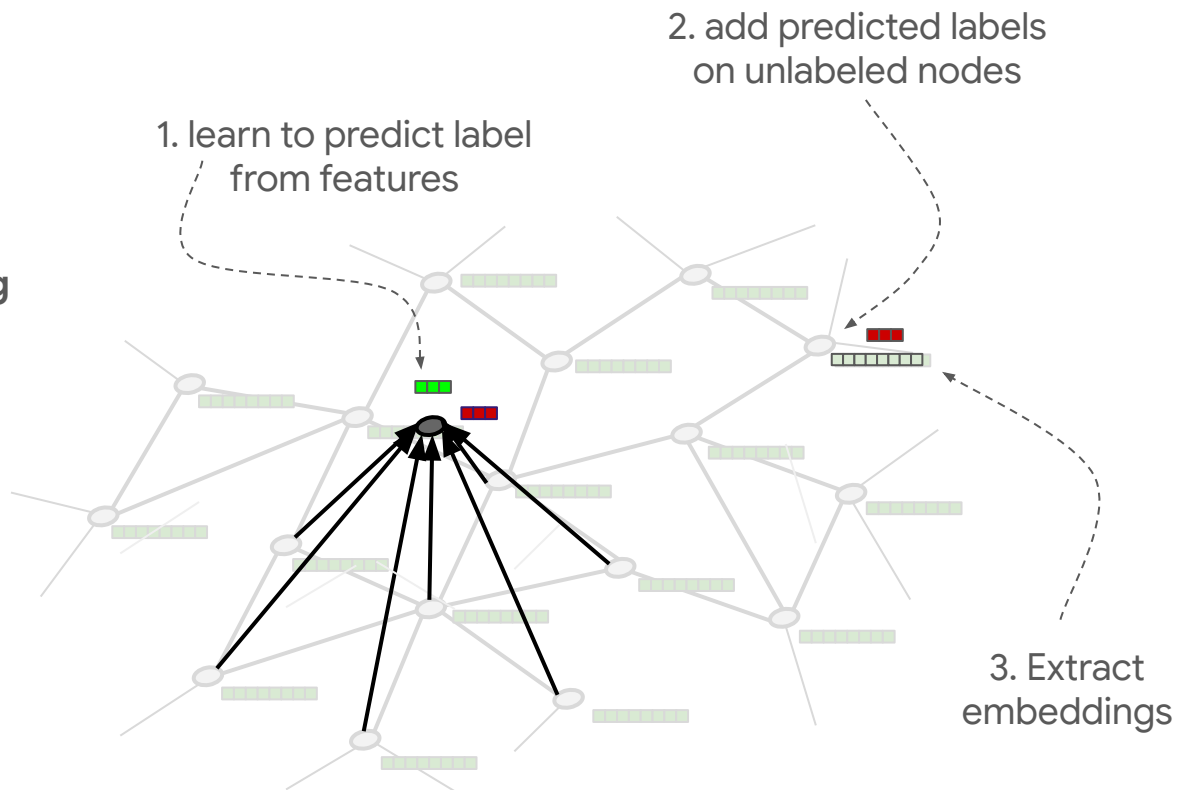


# A Graph Convolution

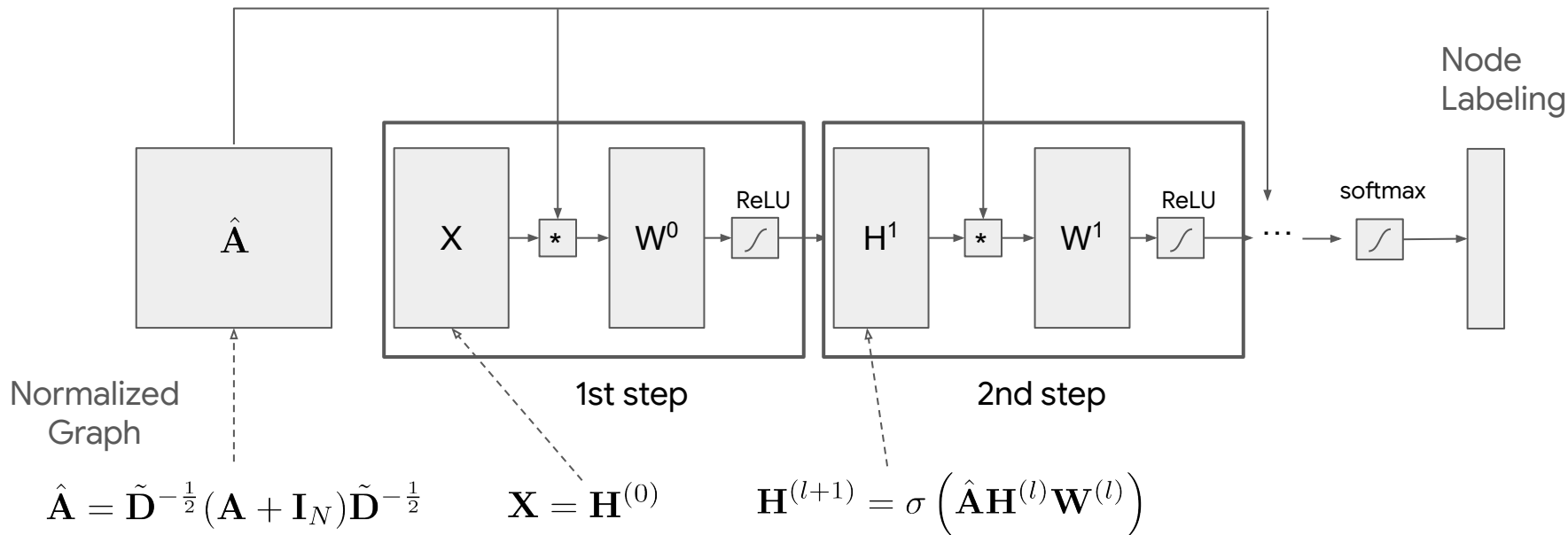
Features  $\rightarrow$  Prediction

You can use this to **label the existing nodes** of a graph.

Learn embeddings and classification in one shot.



# The Graph Convolutional Network (GCN) Model



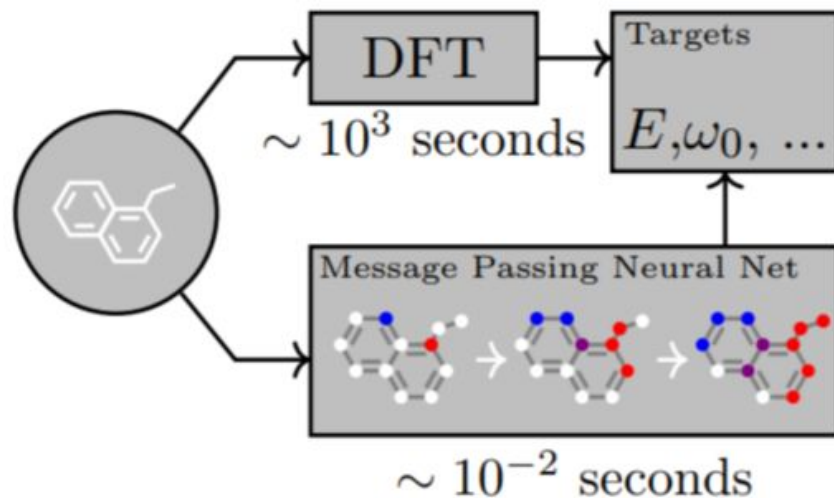
Semi-Supervised Classification with Graph Convolutional Networks

Thomas N. Kipf, Max Welling (ICLR'17)

# Generalizations of GCNs

## Message Passing Neural Networks:

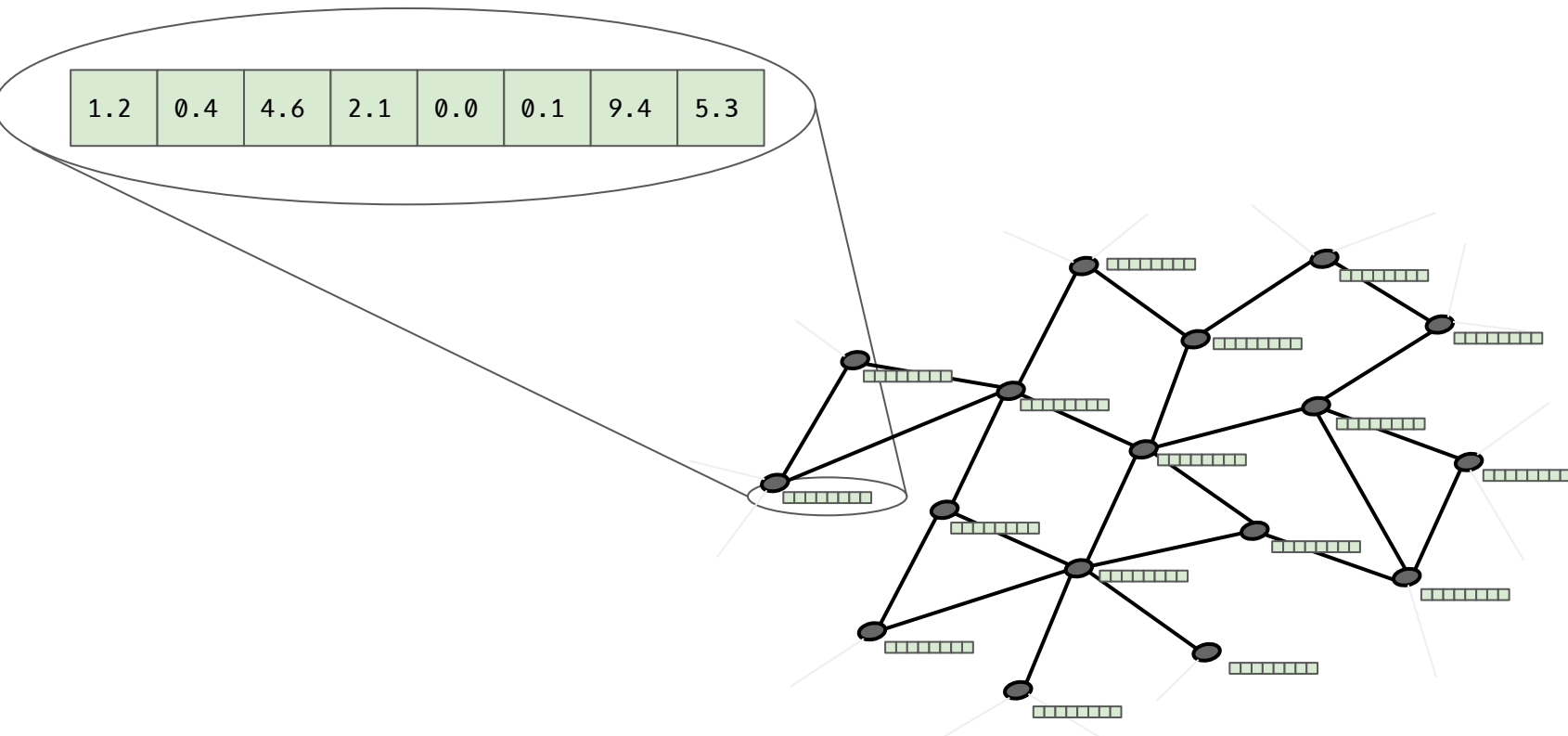
- MPNNs can naturally incorporate heterogeneous vertices and edges.
- They provide arbitrary control of when/how messages are passed.



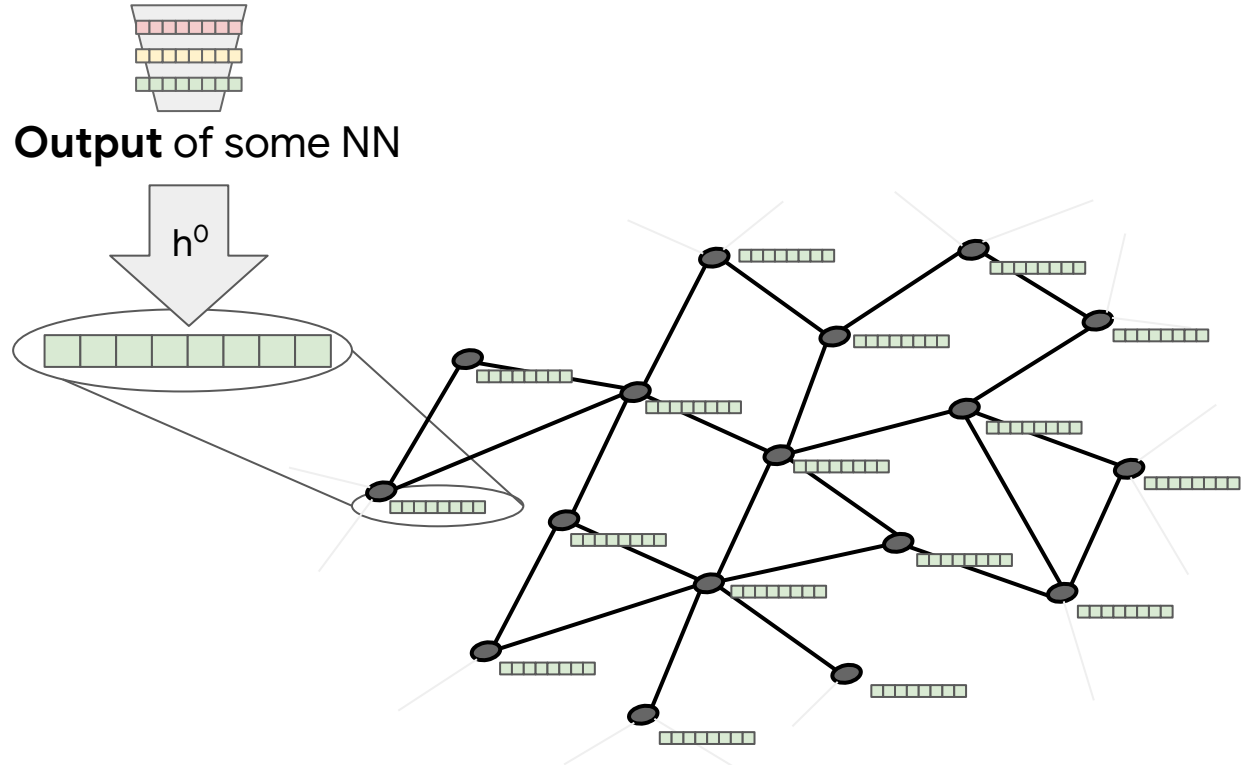
Neural Message Passing for Quantum Chemistry

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, George E. Dahl (ICML'17)

# MPNN: Each node has a state (embedding)



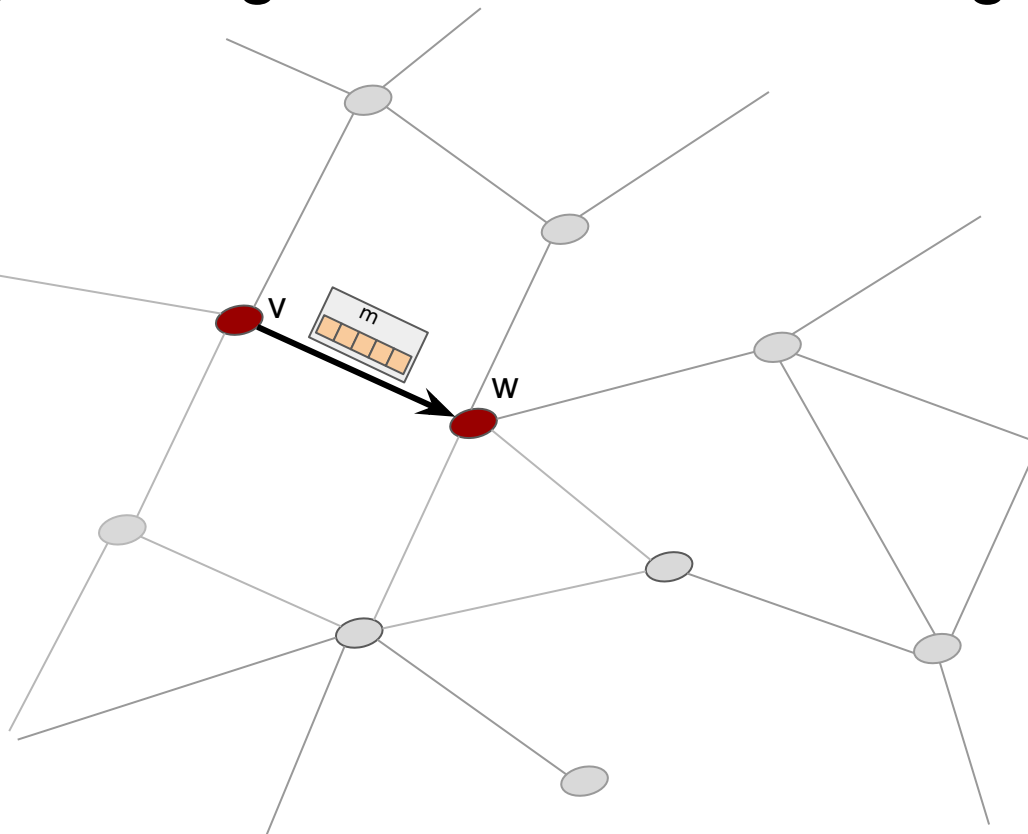
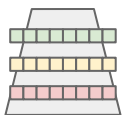
# MPNN: Initial states are fed into GCN



# MPNN: Messages are generated for each edge

$$\mathbf{m}_{v,w}^0 = \mathbf{M}(\mathbf{h}_v, \mathbf{h}_w)$$

Output of some NN

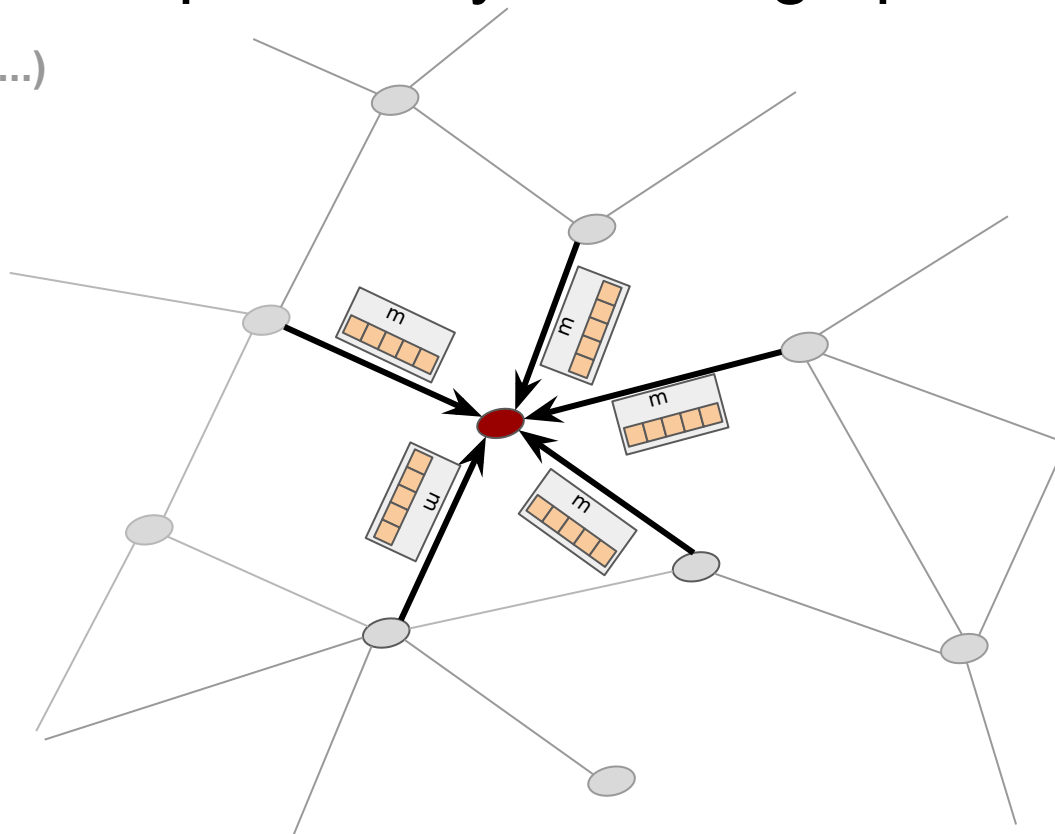
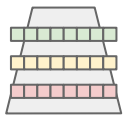


# MPNN: States are updated by “message passing”

(Max, Sum, Attention,...)

$$\mathbf{h}^1 = \mathbf{H}(\mathbf{h}^0, \sum \mathbf{m}^0)$$

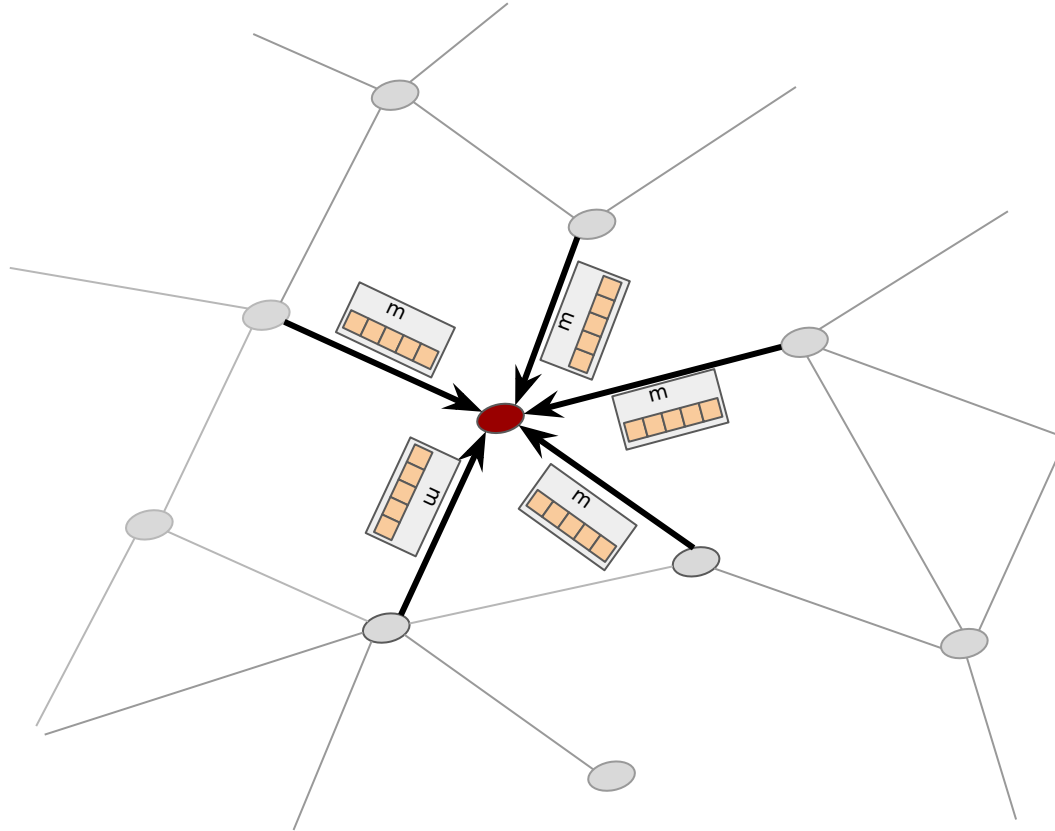
Output of some NN





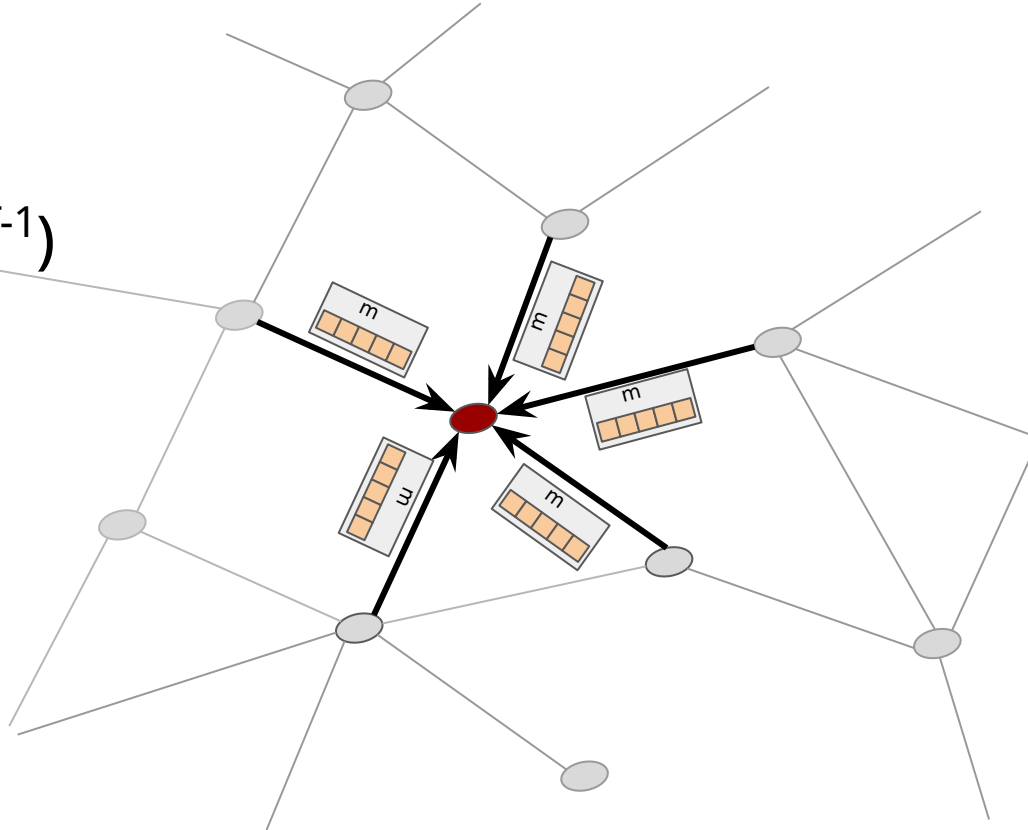
# MPNN: ... and updated ...

$$\mathbf{h}^{t+1} = \mathbf{H}(\mathbf{h}^t, \sum \mathbf{m}^t)$$

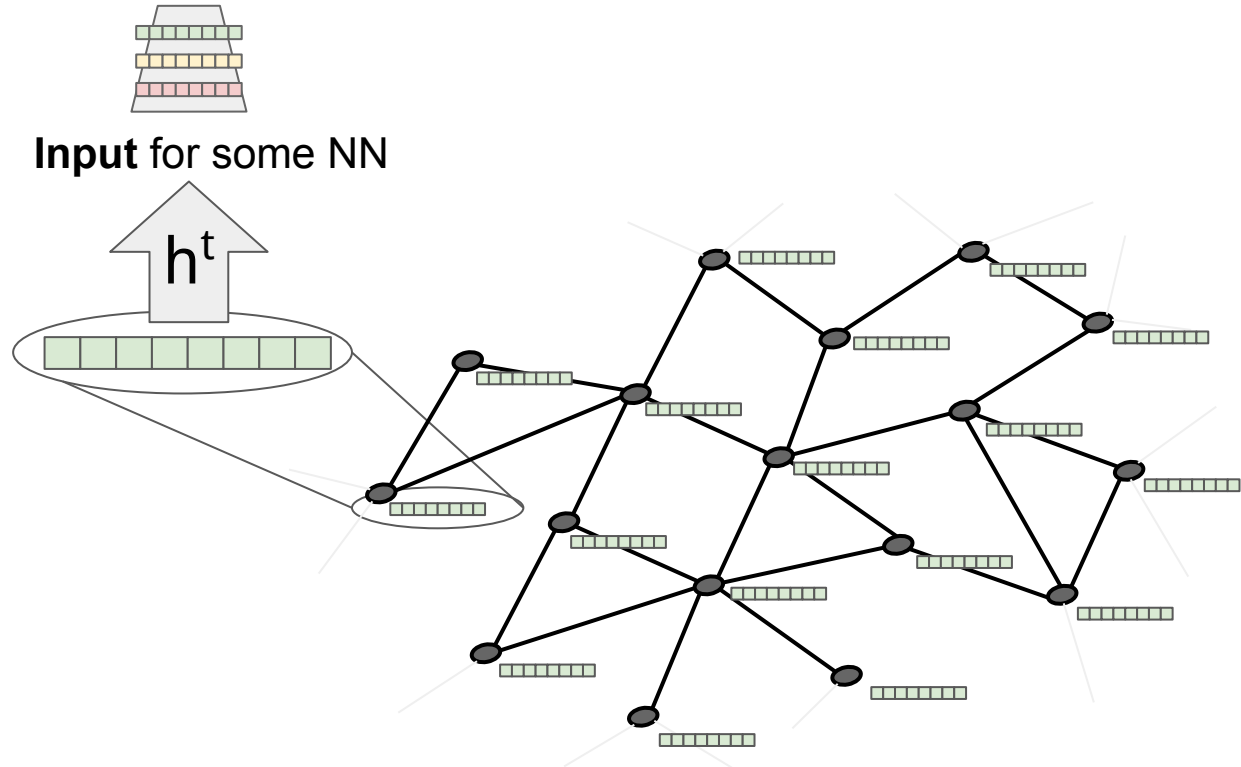


MPNN: ... and updated.

$$\mathbf{h}^T = \mathbf{H}(\mathbf{h}^{T-1}, \sum \mathbf{m}^{T-1})$$



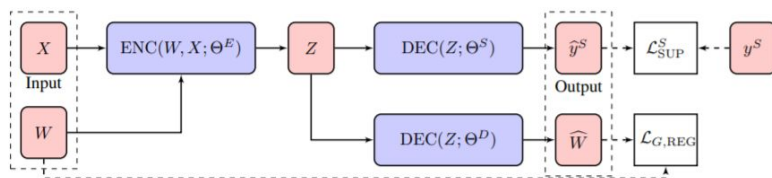
# MPNN: Final states are “read-out”



# There's a lot more!

Increased interest in the area has led to an explosion of models for all kinds of graph data.

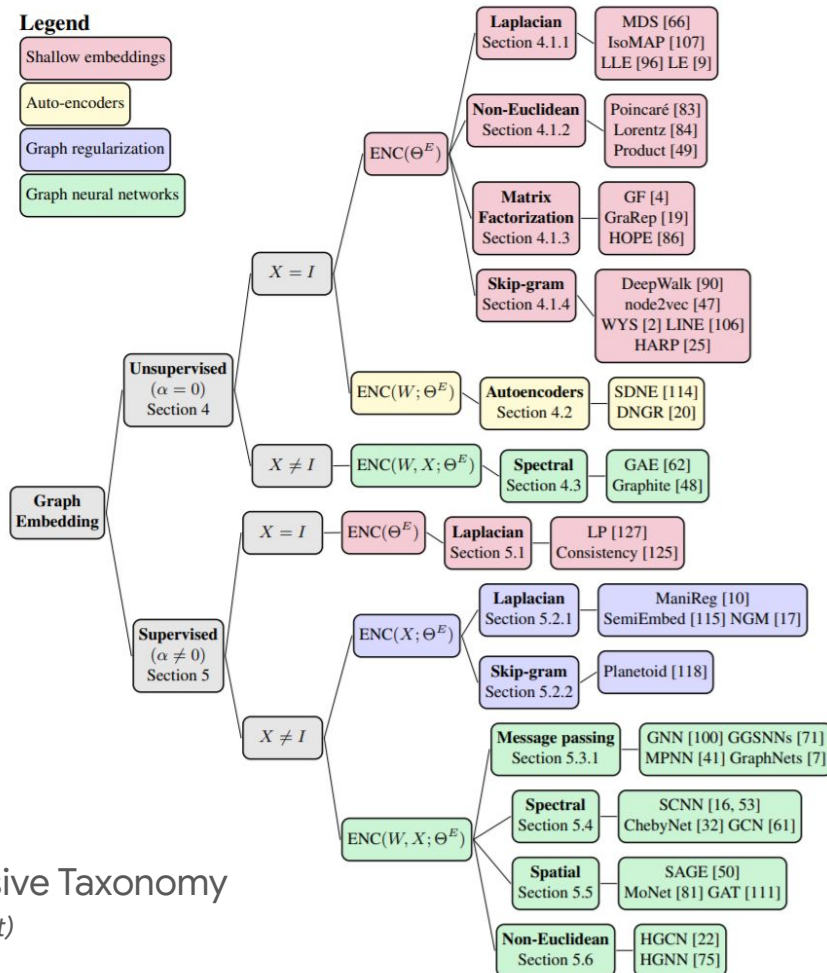
Thankfully many models share common elements, such as an encoder/decoder paradigm:



The **GraphEDM** model.

## Machine Learning on Graphs: A Model and Comprehensive Taxonomy

I Chami, S Abu-El-Hajja, B Perozzi, C Ré, K Murphy (*preprint*)

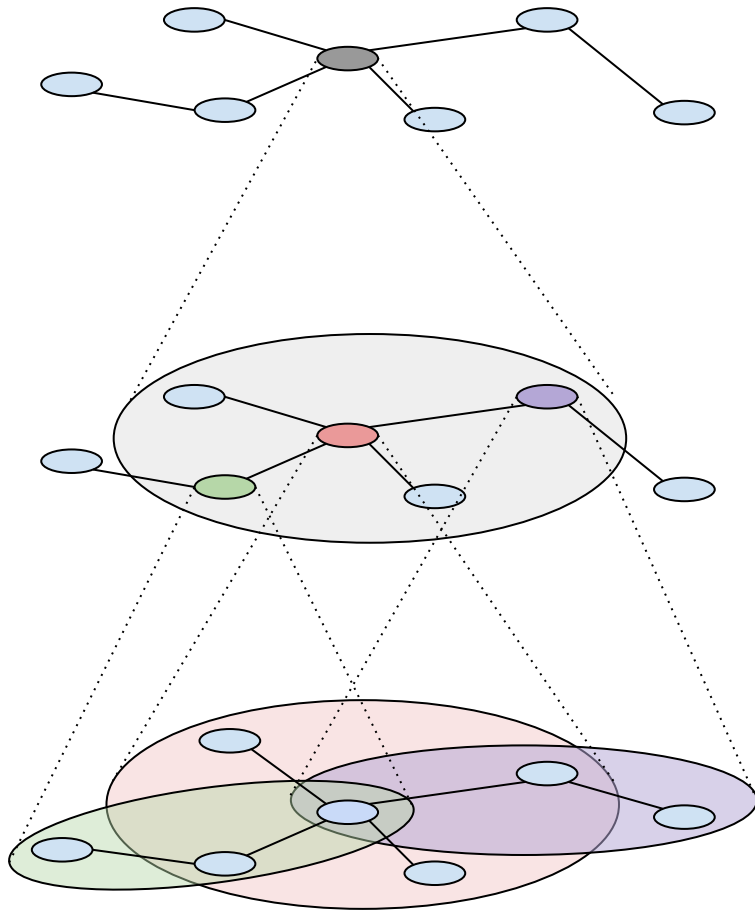


# Challenges of Graph Neural Networks

# Representation Complexity

Unfortunately GCN's aren't perfect. Let's recap how they work real quick:

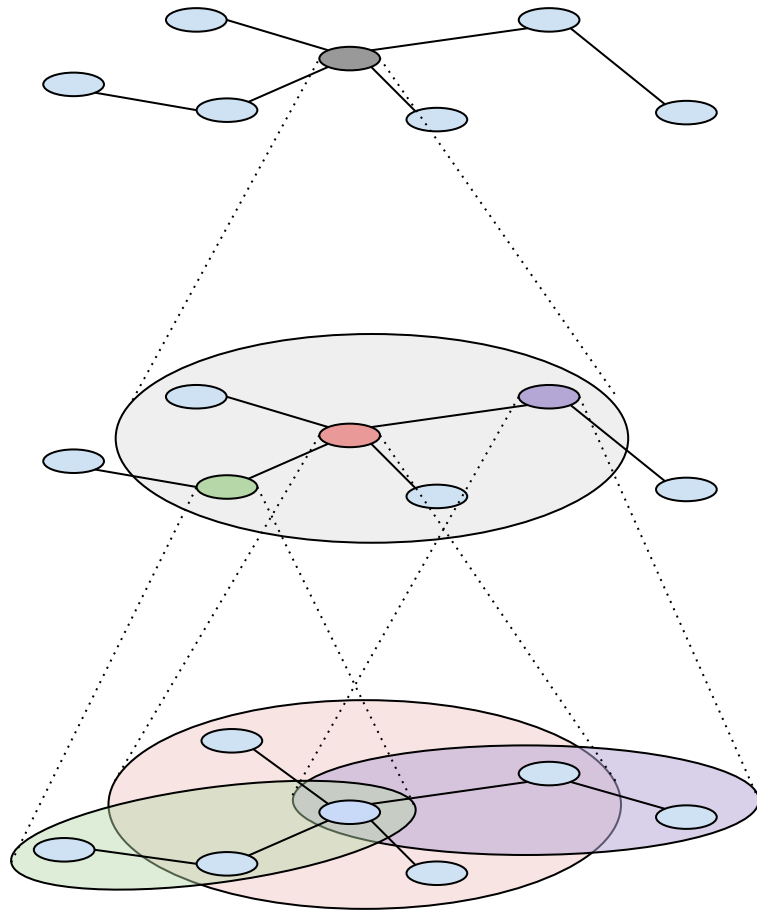
- 1) Start with a graph where each node has some features;
- 2) Aggregate the one-hop neighborhood of each node to create a context-embedding;
- 3) Repeat step 2 until you reach the desired neighborhood size;
- 4) Convert the final embedding into a label.



# Representation Complexity

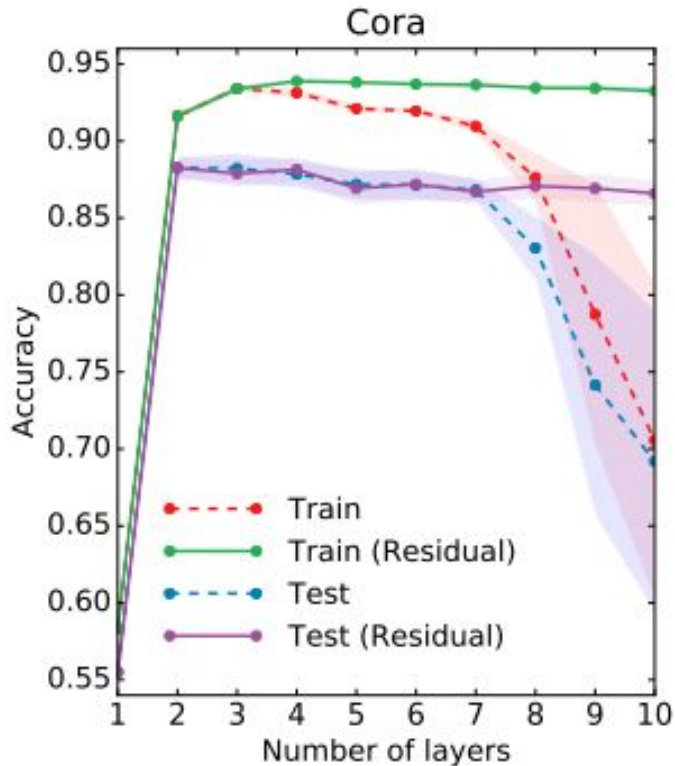
Unfortunately GCN's aren't perfect. Let's recap how they work real quick:

- 1) Start with a graph where each node has some features;
- 2) Aggregate the one-hop neighborhood of each node to create a context-embedding;
- 3) Repeat step 2 until you reach the desired neighborhood size.**
- 4) Convert the final embedding into a label.



# Oversmoothing & GCNs

Kipf and Welling (ICLR'17) demonstrate that adding layers after the 2nd one is a waste of time and compute. In theory, a GCN can learn from an arbitrary deep network. In practice, this will never happen.



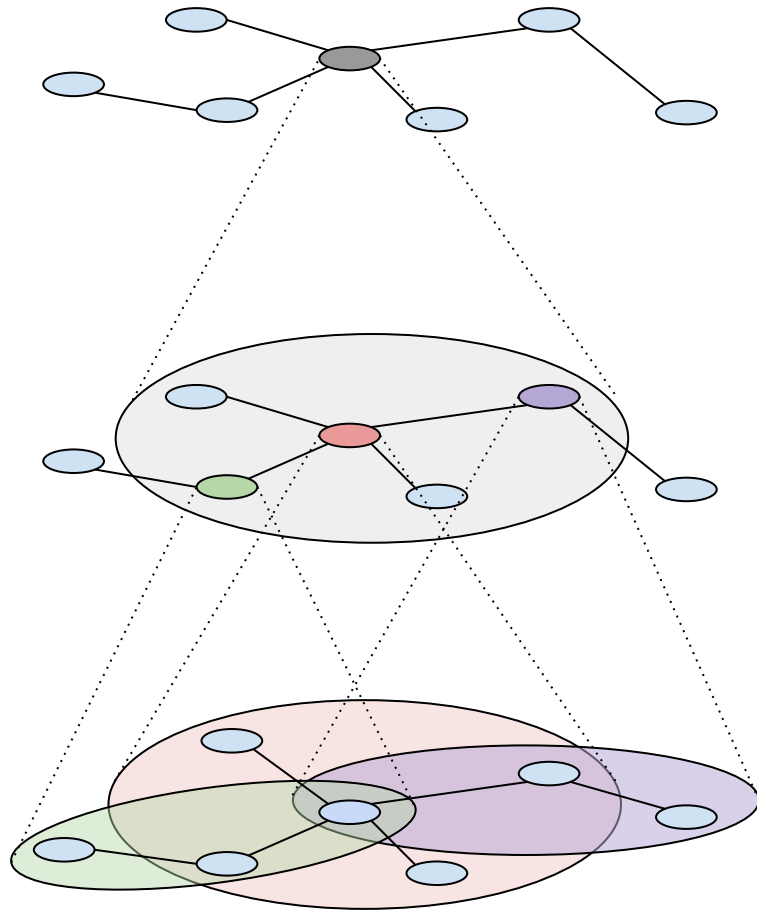


# Oversmoothing & GCNs

Kipf and Welling (ICLR'17) demonstrate that adding layers after the 2nd one is a waste of time and compute. In theory, a GCN can learn from an arbitrary deep network. In practice, this will never happen.

**Why does this happen?** Information in a GCN architecture is aggregated at each layer. This has two impacts:

- 1) Nearby neighbors are used more frequently, resulting in an extremely strong proximity bias.
- 2) The model can *only* learn aggregations between hops. Nothing else.



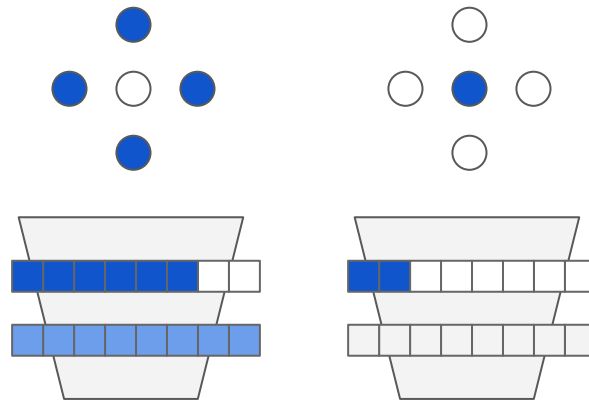
# Where GCNs Fail

We can envision an obvious failure mode of GCNs: when a node has the opposite labels as its neighbors.

On such a prediction task, the GCN -- which is capable of learning only aggregations of nearby neighbors -- would predict the exact opposite of what we want.

A traditional GCN layer can only 'view' one neighborhood hop at a time. Imagine if you could never have a convolution filter larger than 3x3!

What color is the central node?



Blue

White

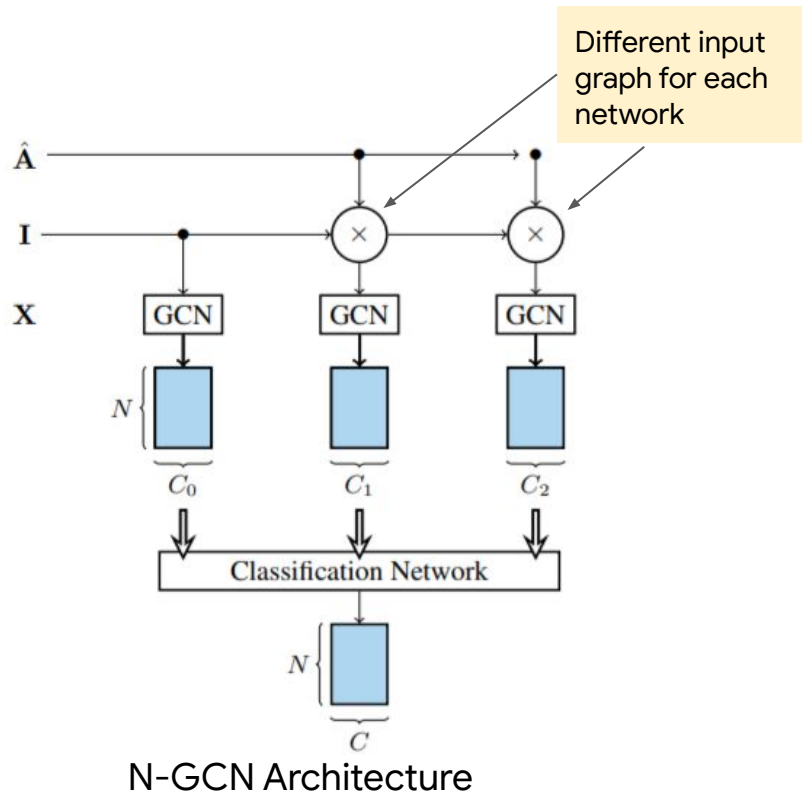


# N-GCN: Mixture of Experts

**First take:** Capture local and global information from a family of GCNs trained on increasingly dense graphs.

Embeddings from the ensemble of networks is combined into a single classification verdict.

Creates rich representations in principle, but models are large (and therefore slow).



N-GCN: Multi-scale graph convolution for semi-supervised node classification

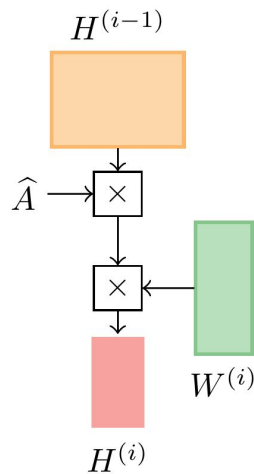
# MixHop: Expanding our Contextual Horizons

**Better answer: expand the filter size!**

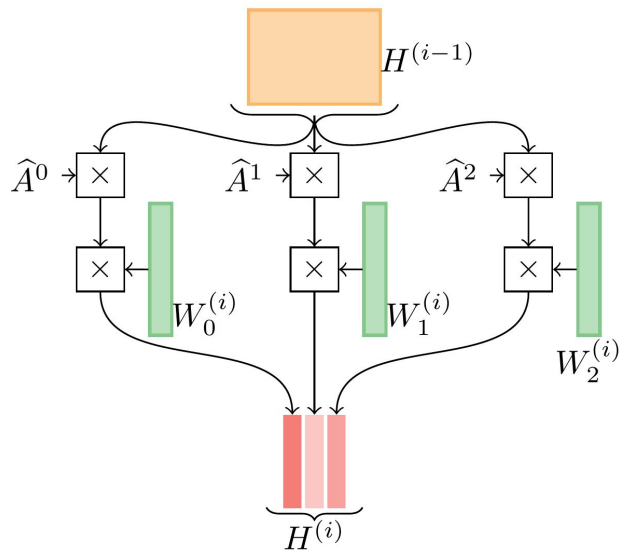
I.e., pushing multi-scale into the filter itself.

For each layer in MixHop, we use a 0, 1, 2, ...N hop neighborhood, and allow the layer to learn to aggregate across all of these hops simultaneously.

$$H^{(i+1)} = \sigma(\hat{A}H^{(i)}W^{(i)})$$



$$H^{(i+1)} = \left\| \sigma \left( \hat{A}^j H^{(i)} W_j^{(i)} \right) \right\|_{j \in P}$$



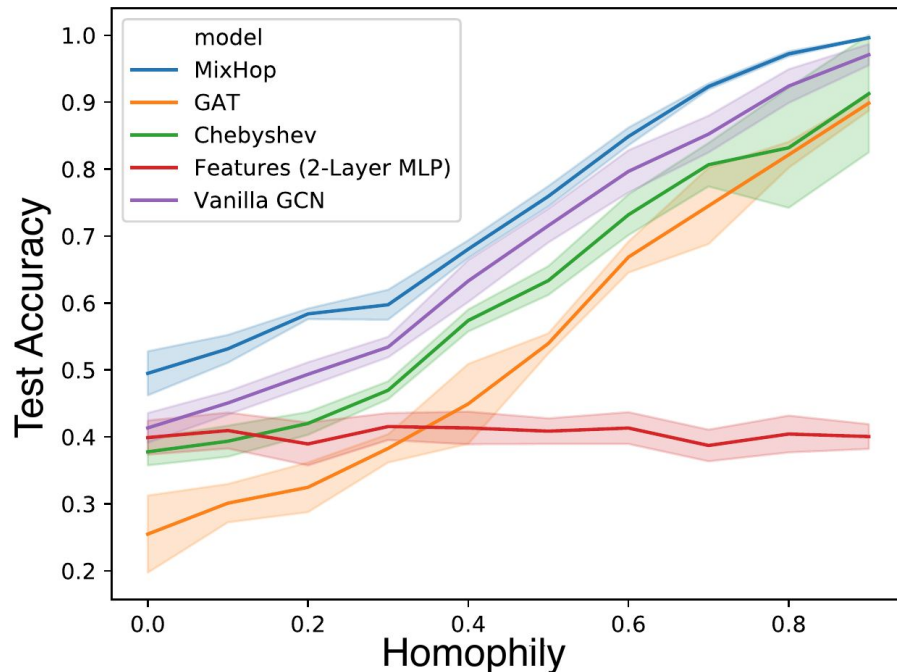
Mixhop: Higher-order Graph Convolution Architectures via Sparsified Neighborhood Mixing

# MixHop: Expanding our Contextual Horizons

## Better answer: expand the filter size!

I.e., pushing multi-scale into the filter itself. For each layer in MixHop, we use a 0, 1, 2, ...N hop neighborhood, and allow the layer to learn to aggregate across all of these hops simultaneously.

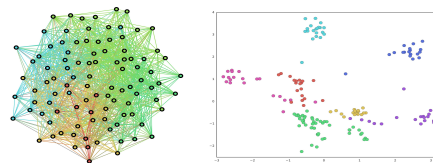
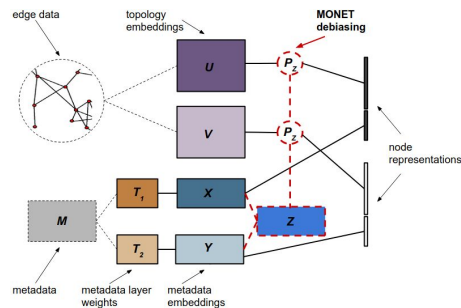
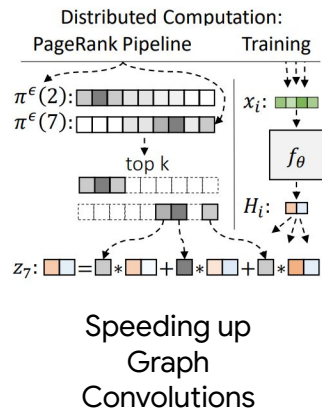
This is a massive qualitative improvement. The MixHop model can learn *difference functions* across layers, which (in image terms) are equivalent to *edge detectors*. When stacked with other MixHop layers, we can build up native hierarchical graph representations.



# More Challenges of Graph Learning

In the following sections, we'll cover additional solutions we've developed to a number of practical challenges of using GNNs:

- How can we make GCNs fast?
- What biases might a GNN contain?
- How can we model complex interactions?



Understanding nodes which have multiple communities

# Citations

## PAPERS:

DeepWalk: Online Learning of Social Representations  
B Perozzi, R Al-Rfou, S Skiena (KDD'14)

Learning Edge Representations via Low-Rank Asymmetric Projections  
S Abu-El-Hajja, B Perozzi, R Al-Rfou (CIKM'17)

HARP: Hierarchical Representation Learning for Networks  
H Chen, B Perozzi, Y Hu, S Skiena (AAAI'18)

Watch Your Step: Learning Node Embeddings via Graph Attention  
S Abu-El-Hajja, B Perozzi, R Al-Rfou, AA Alemi (NeurIPS'18)

Semi-Supervised Classification with Graph Convolutional Networks  
Thomas N. Kipf, Max Welling (ICLR'17)

Neural Message Passing for Quantum Chemistry  
Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, George E. Dahl (ICML'17)

Machine Learning on Graphs: A Model and Comprehensive Taxonomy  
I Chami, S Abu-El-Hajja, B Perozzi, C Ré, K Murphy (preprint)

N-GCN: Multi-scale graph convolution for semi-supervised node classification  
S Abu-El-Hajja, A Kapoor, B Perozzi, J Lee (UAI'19)

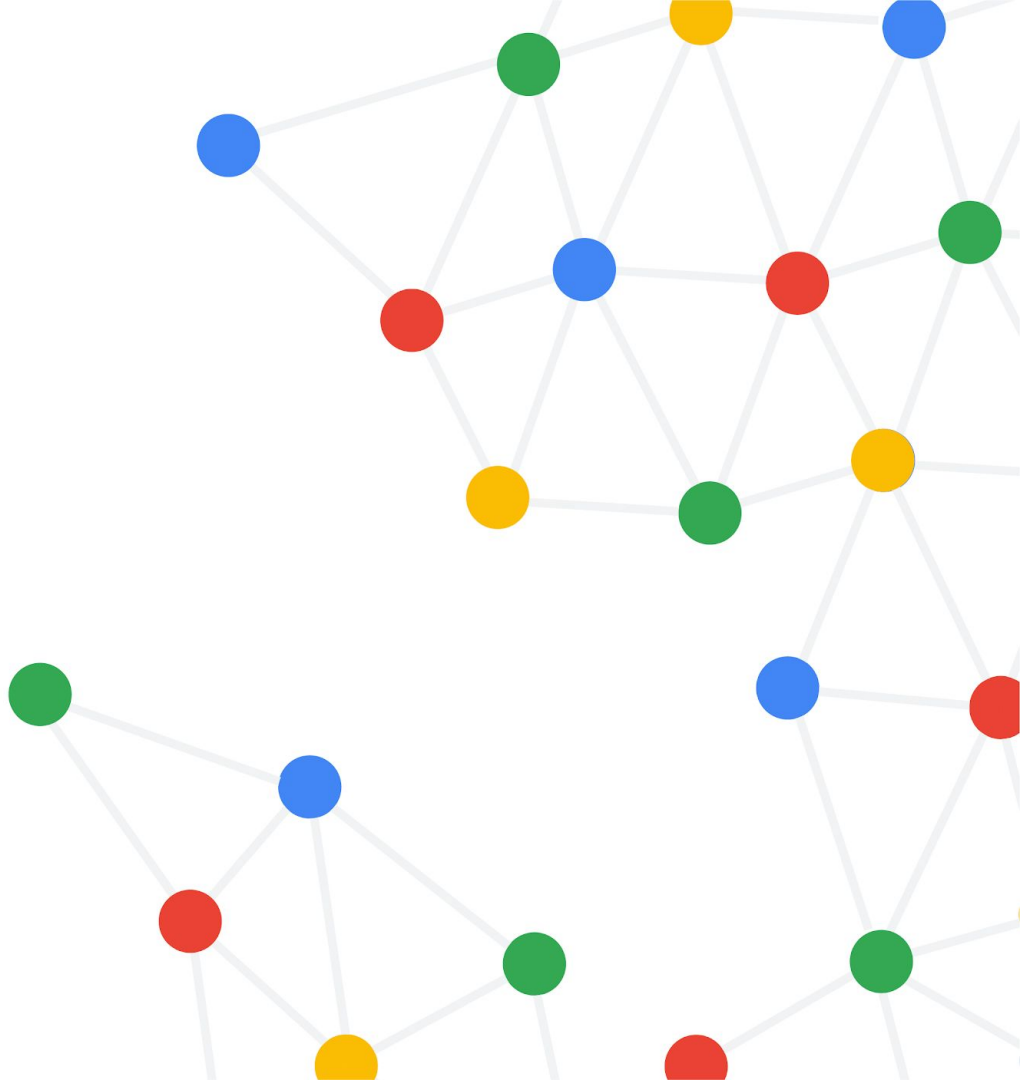
Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing  
S Abu-El-Hajja, B Perozzi, A Kapoor, H Harutyunyan, N Alipourfard, K Lerman, G Ver Steeg, A Galstyan (ICML'19)

## ICONS:

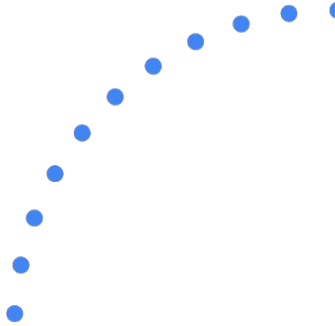
account: <https://thenounproject.com/search/?q=account&i=1931153>

publisher: <https://thenounproject.com/search/?q=publisher&i=3048742>

advertise: <https://thenounproject.com/search/?q=advertiser&i=2374780>



# Graph Neural Networks





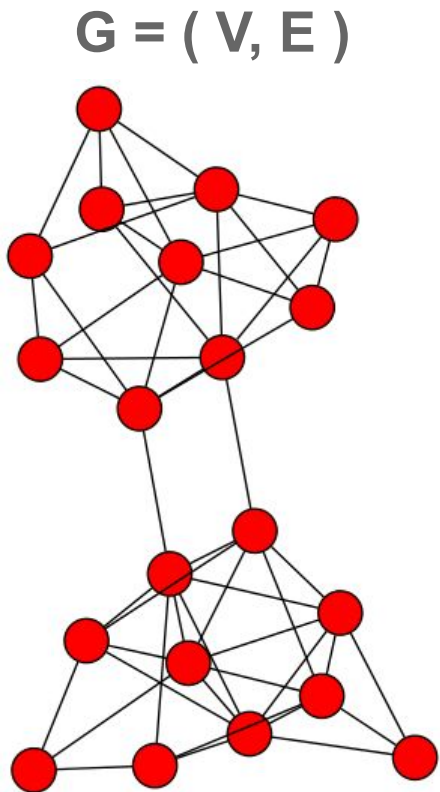
# Debiasing GNNs

John Palowitch, Bryan Perozzi

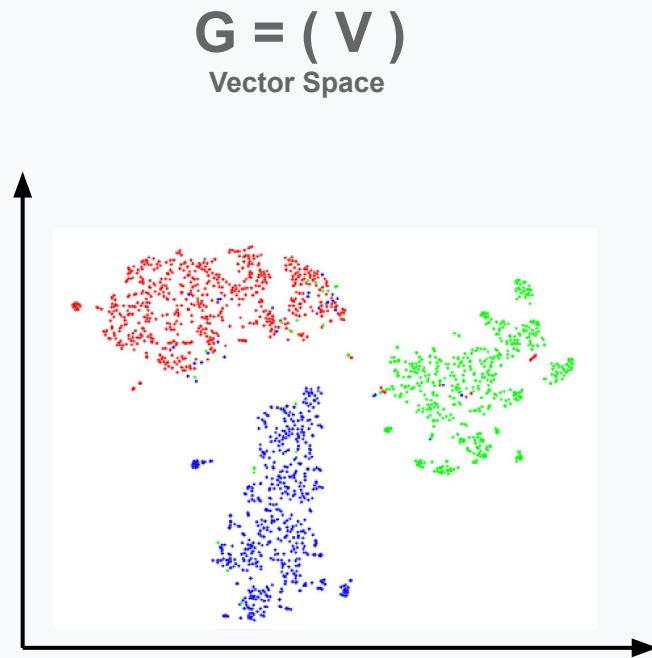
*Google Research*



# Embeddings and embedding layers in GNNs



embed  
→



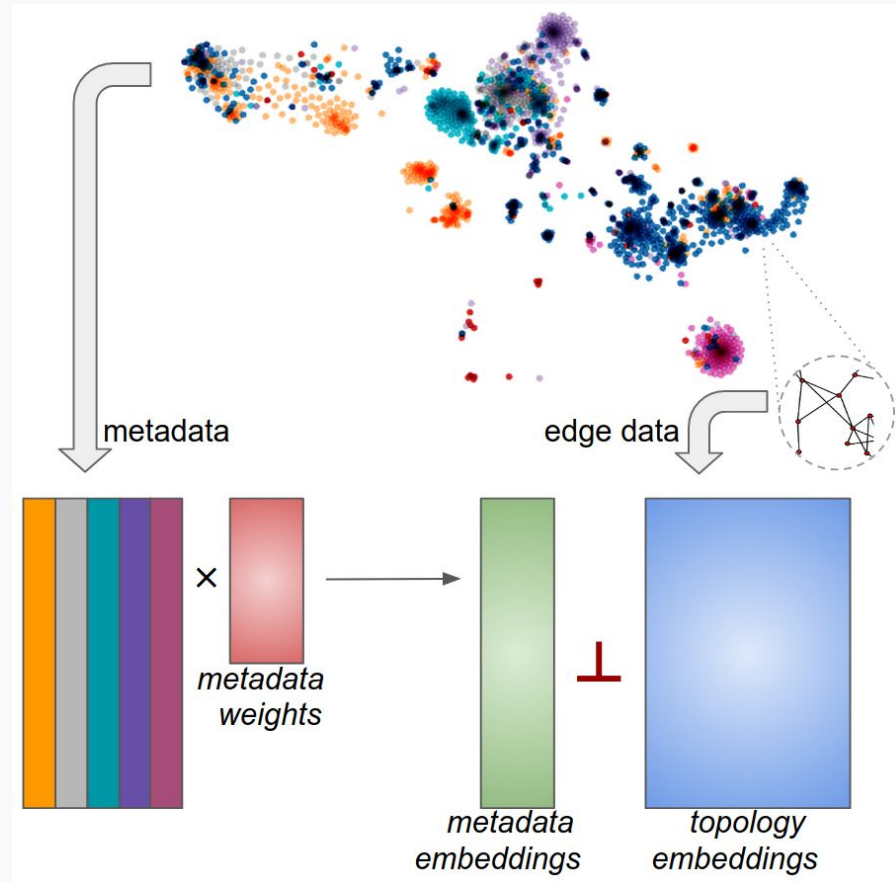
Good for visualization, denoising, and scalable ML

# Metadata and graph embeddings

Many graphs cluster by attributes:

- gender, age, income, etc
- content embedding
- spatial properties

**Problem:** how do we learn embeddings unbiased by *sensitive* metadata?



# Metadata and graph embeddings

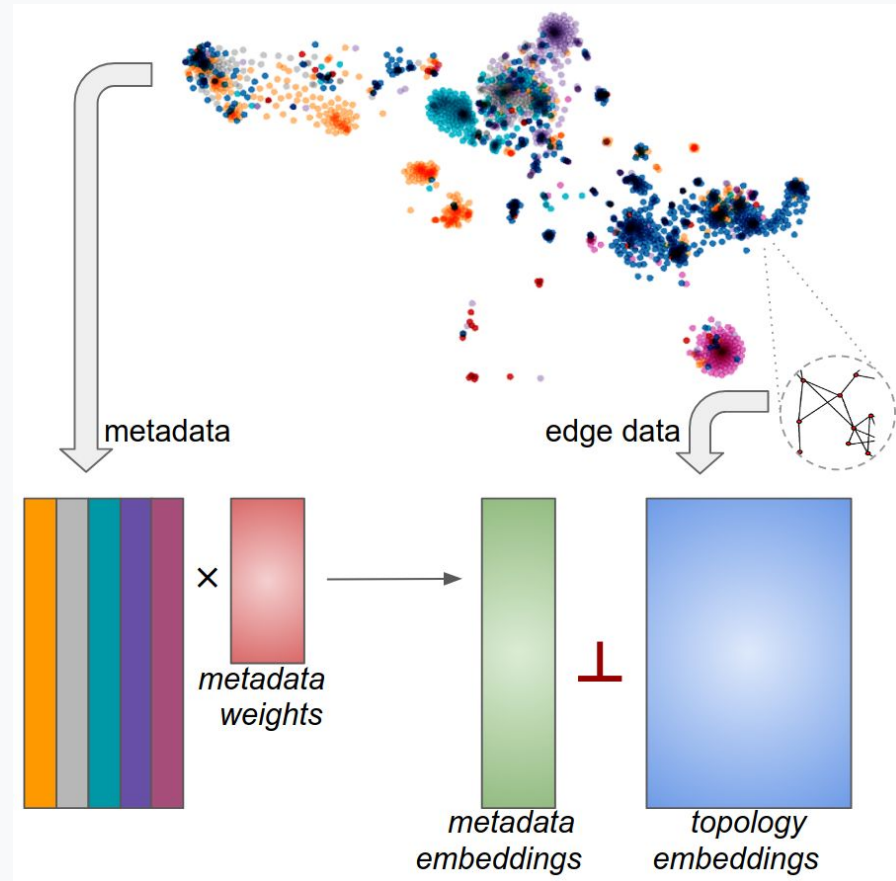
Many graphs cluster by attributes:

- gender, age, income, etc
- content embedding
- spatial properties

**Problem:** how do we learn embeddings unbiased by *sensitive* metadata?

## Related work:

- [Adversarial Debiasing](#) (Bose and Hamilton 2019): train adversary to predict metadata, backpropagate inverse loss.
- [FairWalk](#) (Rahman et al 2019): make random walks conditionally independent of metadata.



# Metadata and graph embeddings

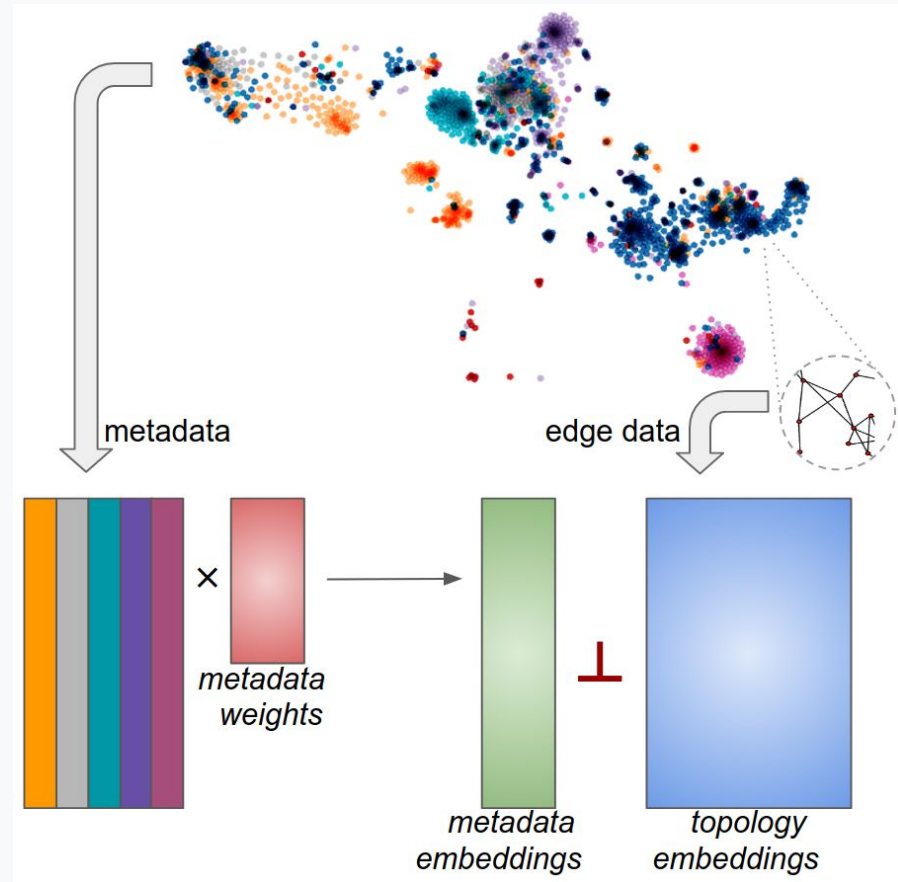
Many graphs cluster by attributes:

- gender, age, income, etc
- content embedding
- spatial properties

**Problem:** how do we learn embeddings unbiased by *sensitive* metadata?

**Our two-part solution:**

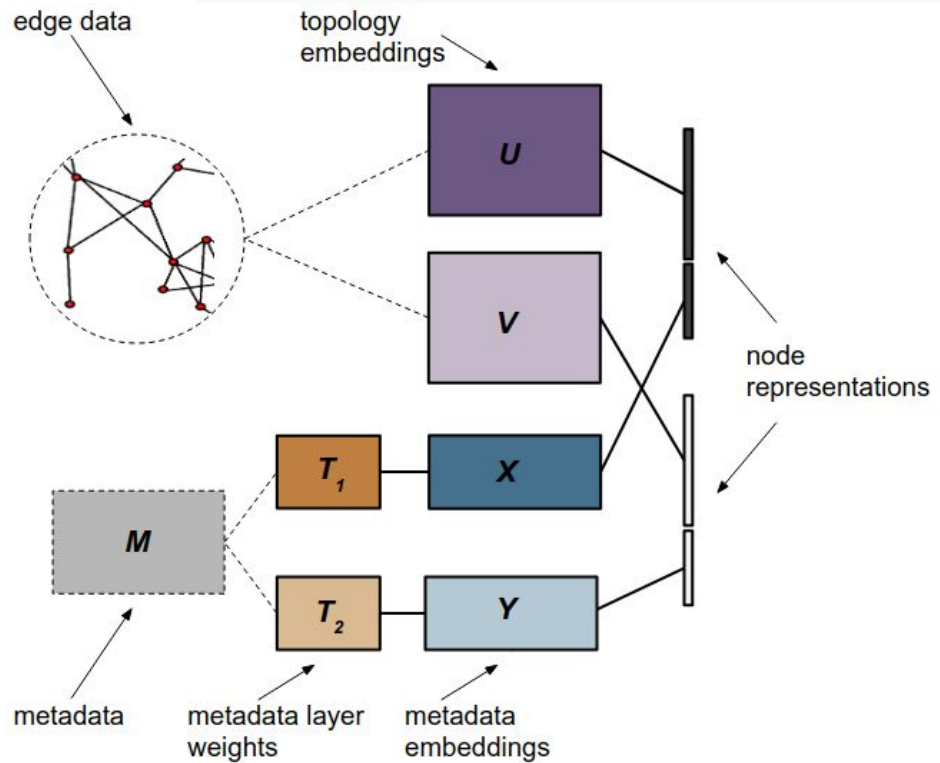
1. Learn metadata embeddings
2. Orthogonalize topology and metadata



# Learn Metadata Embeddings

1. Feed random walks through encoder:
  - **topology** embeddings  $U, V$
2. Feed metadata through NN:
  - **metadata** embeddings  $X, Y$
3. Full graph representations:

$[U, X]$     $[V, Y]$



**Hypothesis:** Metadata embeddings  $X, Y$  encode metadata signal, debiasing  $U, V$

**Result:** some debiasing occurs, but not all.

# Learn Metadata Embeddings

Political Blog (“polblogs”) graph:

- Nodes are political blogs
- Edges are hyperlinks from 2004
- 2 clearly-defined polar clusters

**Metadata Leakage:**

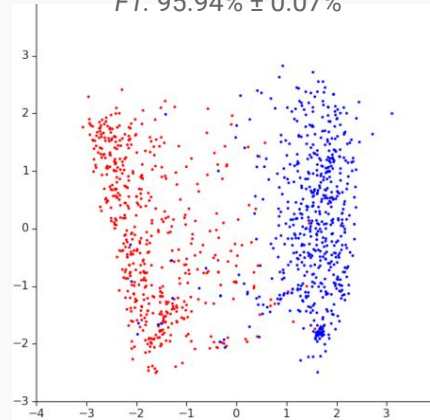
$$\mathcal{ML}(U, X) := \|U^T X\|_F^2$$

**Theorem:** Under a random gradient descent update from the GloVe<sub>meta</sub> model,

$$\mathbb{E}[\mathcal{ML}(U, X)] = \Omega(n) \text{ as } n \rightarrow \infty$$

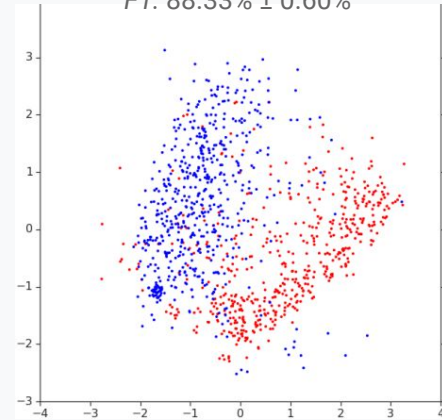
GloVe

*metadata leakage: 6598.0 ± 200.1*  
*F1: 95.94% ± 0.07%*



GloVe<sub>meta</sub>

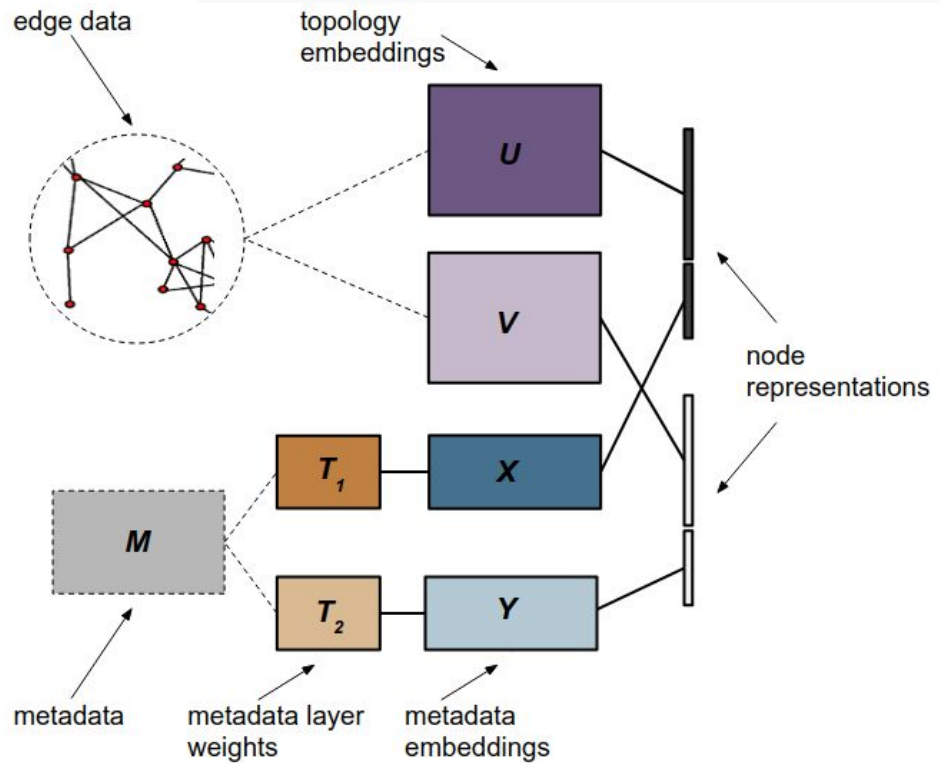
*metadata leakage: 1827.6 ± 289.7*  
*F1: 88.33% ± 0.60%*



*Topology embedding PCA. Color = political affiliation.*

# Learn Metadata Embeddings

1. Feed random walks through encoder:
  - **topology** embeddings  $U, V$
2. Feed metadata through NN:
  - **metadata** embeddings  $X, Y$

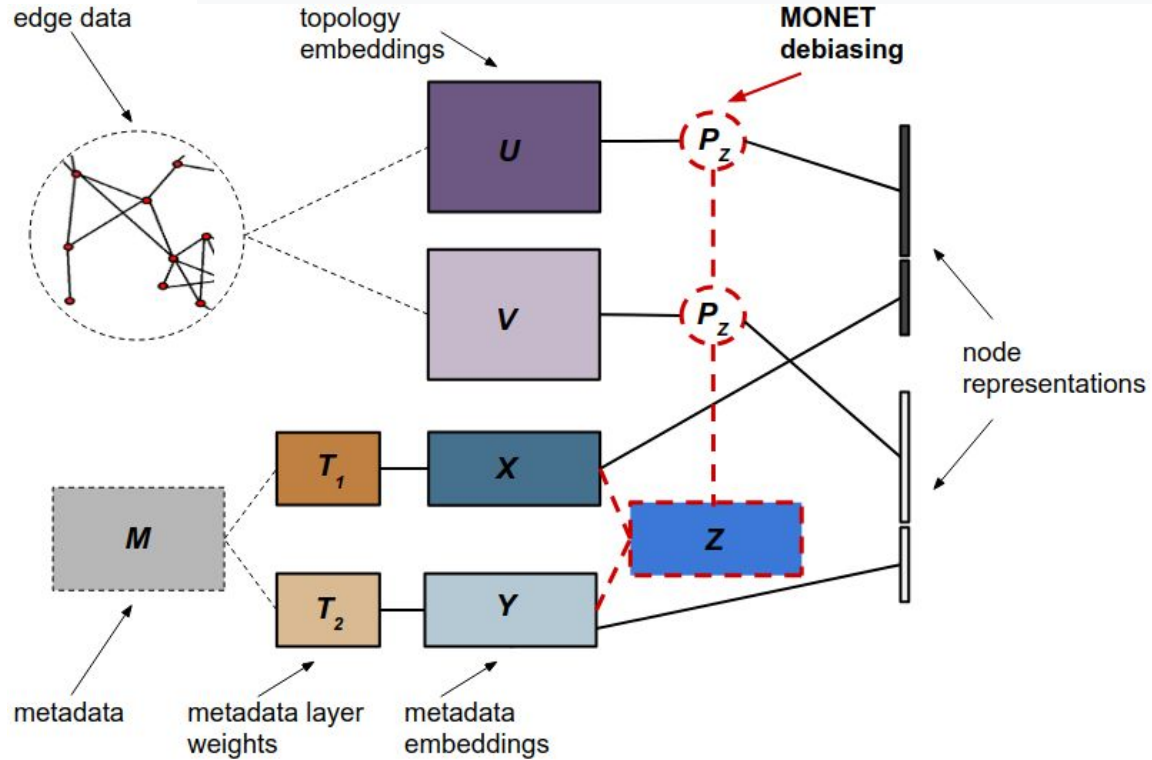




# MONET: Orthogonalize Metadata and Graph Embeddings

1. Feed random walks through encoder:
  - **topology** embeddings  $U, V$
2. Feed metadata through NN:
  - **metadata** embeddings  $X, Y$
3. **MONET: Project  $U, V$  on metadata-orthogonal hyperplane**
4. Full graph representations:

$$[P_Z U, X] \quad [P_Z V, Y]$$



**Result:** exact linear debiasing

# MONET: Orthogonalize Metadata and Graph Embeddings

---

## Algorithm 1: MONET Unit Training Step

---

**Input:** topology embedding  $W$ , metadata embedding  $Z$

**procedure** FORWARD PASS DEBIASING( $W, Z$ )

  Compute  $Z$  left-singular vectors  $Q_Z$  and projection:

$$P_Z \leftarrow I_{n \times n} - Q_Z Q_Z^T$$

  Compute orthogonal topology embedding:

$$W^\perp \leftarrow P_Z W$$

**return** debiased graph representation  $[W^\perp, Z]$

**end procedure**

**procedure** BACKWARD PASS DEBIASING( $\delta_W$ )

  Compute orthogonal topology embedding update:

$$\delta_W^\perp \leftarrow P_Z \delta_W$$

  Apply update:

$$W^\perp \leftarrow W^\perp + \delta_W^\perp$$

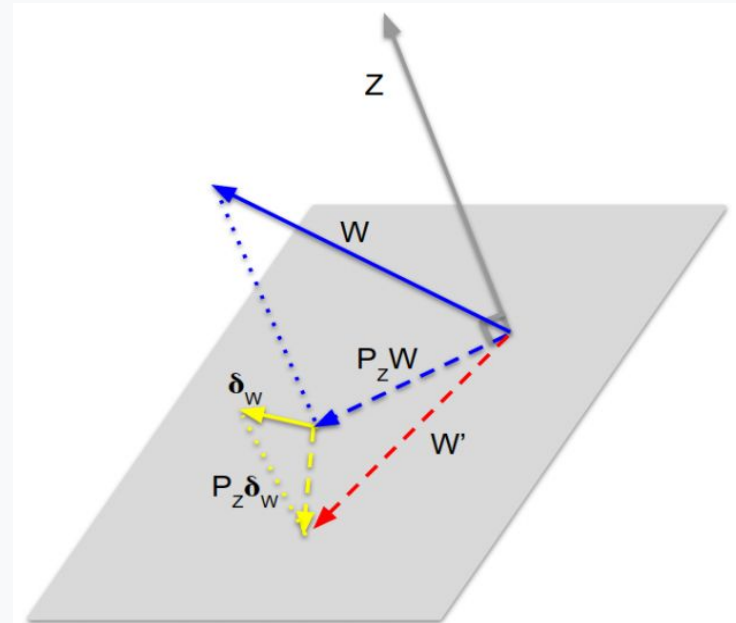
**return** debiased topology embedding  $W^\perp$

**end procedure**

---

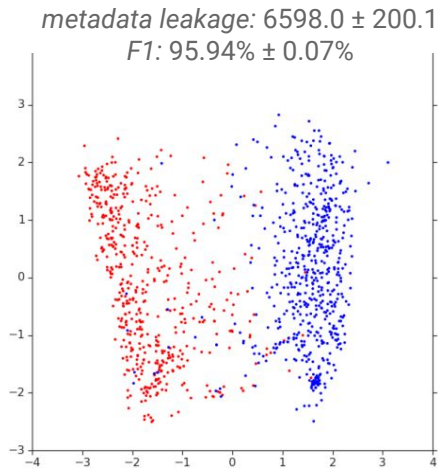
Relaxation:  $P_Z^{(\lambda)} := I_{n \times n} - \lambda Q_Z Q_Z^T$

$$\lambda \in [0, 1]$$

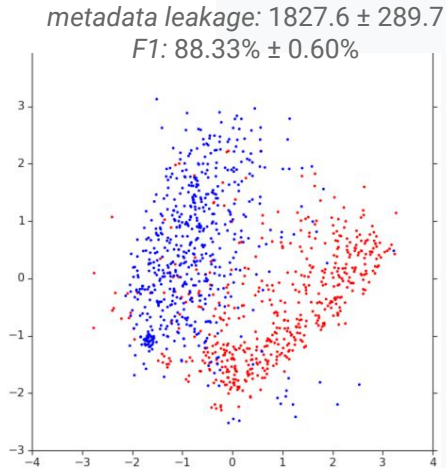


# MONET: Orthogonalize Metadata and Graph Embeddings

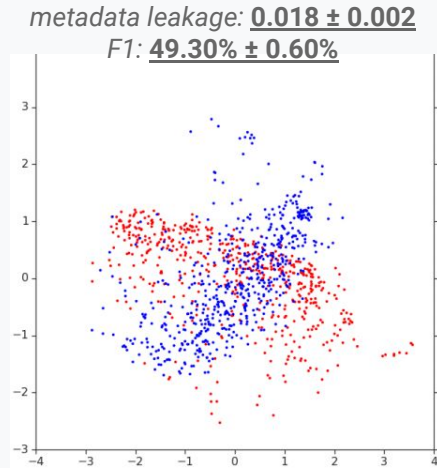
GloVe



GloVe<sub>meta</sub>



MONET<sub>GloVe</sub>



# Experiment 1: MONET debiases blog political affiliation

## Debiasing baselines:

- Adversarial (Bose & Hamilton 2019)
- FairWalk
- GloVe<sub>meta</sub>

## Standard baselines:

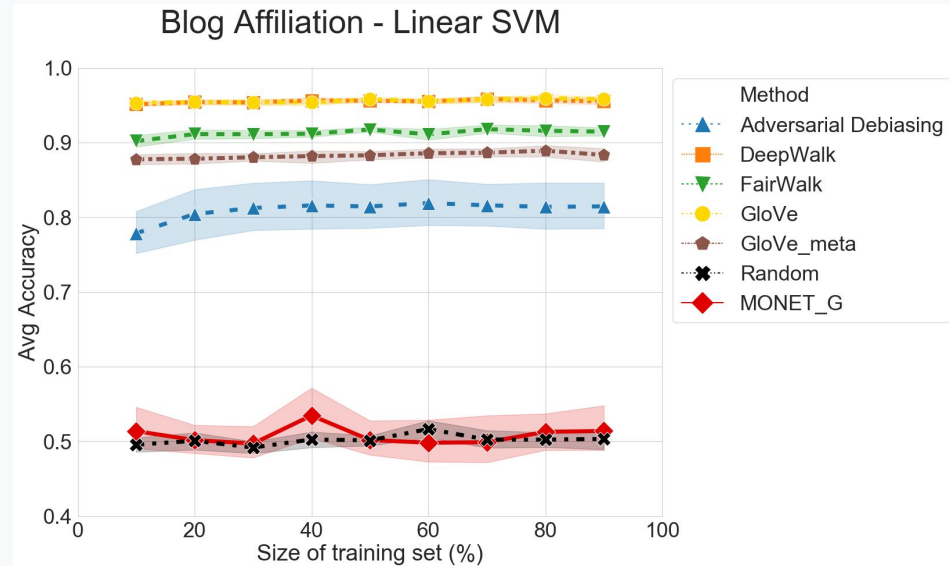
- DeepWalk
- GloVe

## Experiment:

1. Embed the graph
2. Train linear classifier on blog affiliation
3. Compute accuracy (higher = more bias)

**Key Result:** MONET-debiased embeddings consistent with random baseline.

Linear classifier prevented from predicting affiliation.



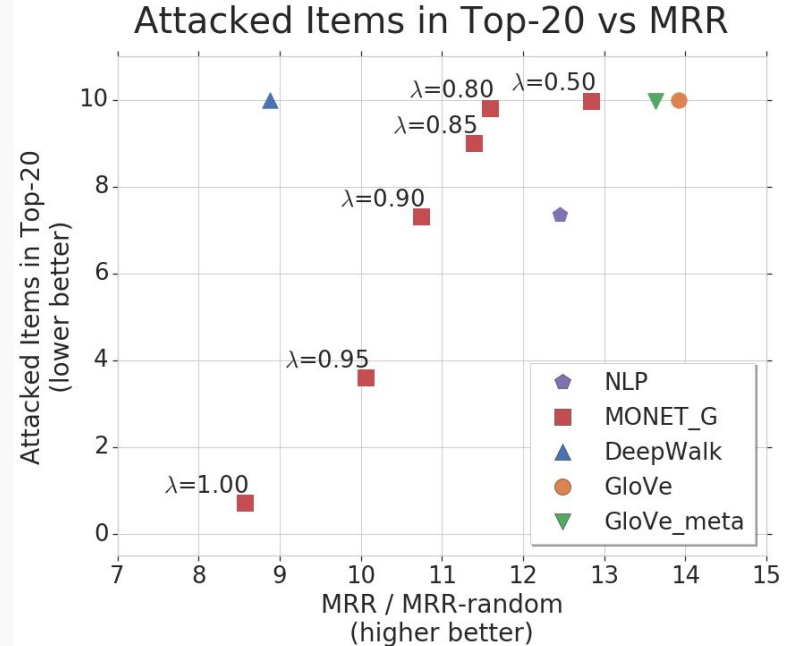
# Experiment 2: MONET debiases shilling attack

## Experiment:

1. Simulate spam attack on MovieLens graph
  - on 10 randomly-chosen videos
  
2. Fold into video-video graph & embed
  - video metadata = # known spam hits
  
3. Metrics:
  - # of attacked videos in 20-nn of other attacked videos  
-----> **Measures bias**
  
  - Embedding Distance MRR to top random walk neighbors  
-----> **Measures signal corruption**

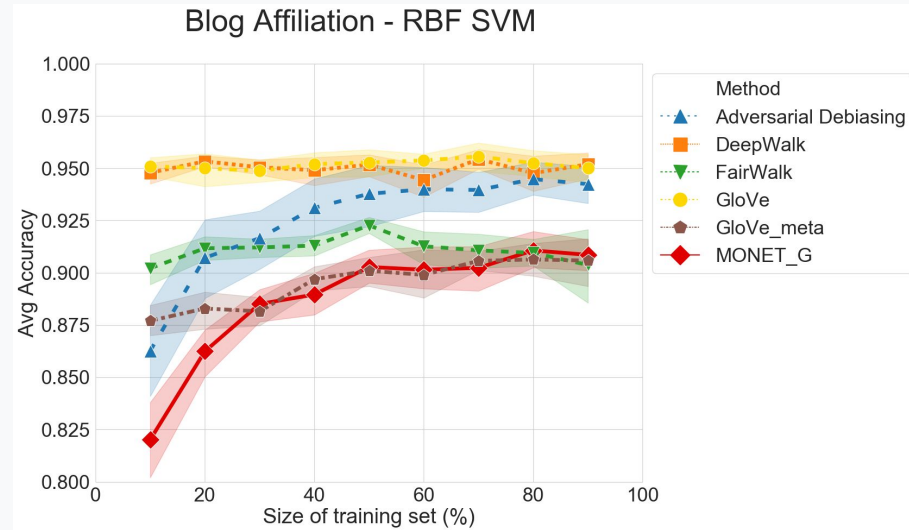
**Key Result:** MONET provides tunable debiasing with bias-accuracy trade-off.

Exact debiasing still results in 8x gain over random.



# Comparison to related work

- MONET only provides linear debiasing. However, unlike other methods, it *guarantees* debiasing with a scalable training-time operation.
- Downside of FairWalk: Nodes with neighbors of only one metadata class will not be debiased.
- Adversarial debiasing can handle non-linear bias in theory, but in practice can fail to do so.



# Thank you!

## Future work:

- Non-linear debiasing
- High-dimensional metadata
- Deep GNNs

Palowitch, John; Perozzi, Bryan; [“MONET: Debiasing Graph Embeddings via the Metadata-Orthogonal Training Unit”](#) to appear at ASONAM 2020 (arxiv:1909.11793)



# PPRGo: GNNs at Scale

Amol Kapoor



# Scaling Graph Neural Networks with Approximate PageRank

*Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, Stephan Günnemann*

*KDD'20.*

<https://arxiv.org/abs/2007.01570>

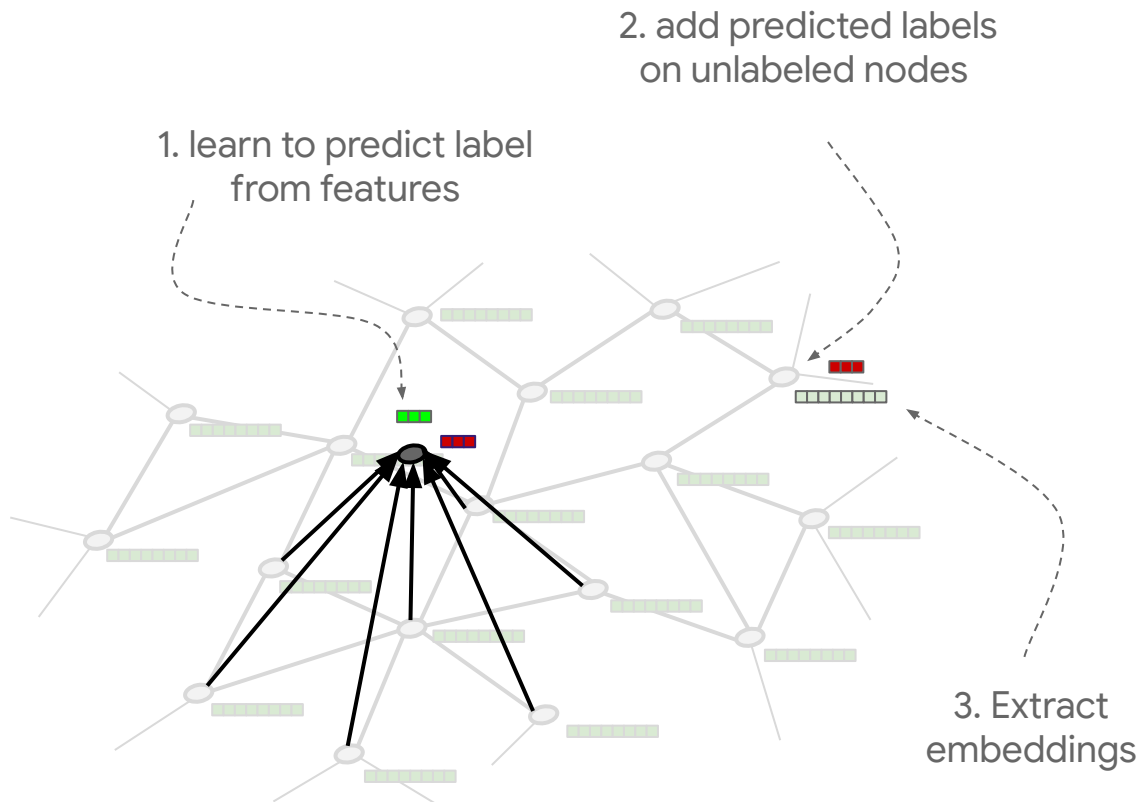


# Background

# GCNs: A Quick Recap

Graph Convolutional Networks are a way of incorporating graph context into the embeddings of a specific node. Using stacked GCN layers, we can build up a hierarchical representation of a graph.

Because each part of the graph convolution is learned, we can utilize node neighborhoods to make smarter decisions in a wide range of tasks. GCNs are therefore an extremely powerful and flexible part of the Graph Mining toolbox.



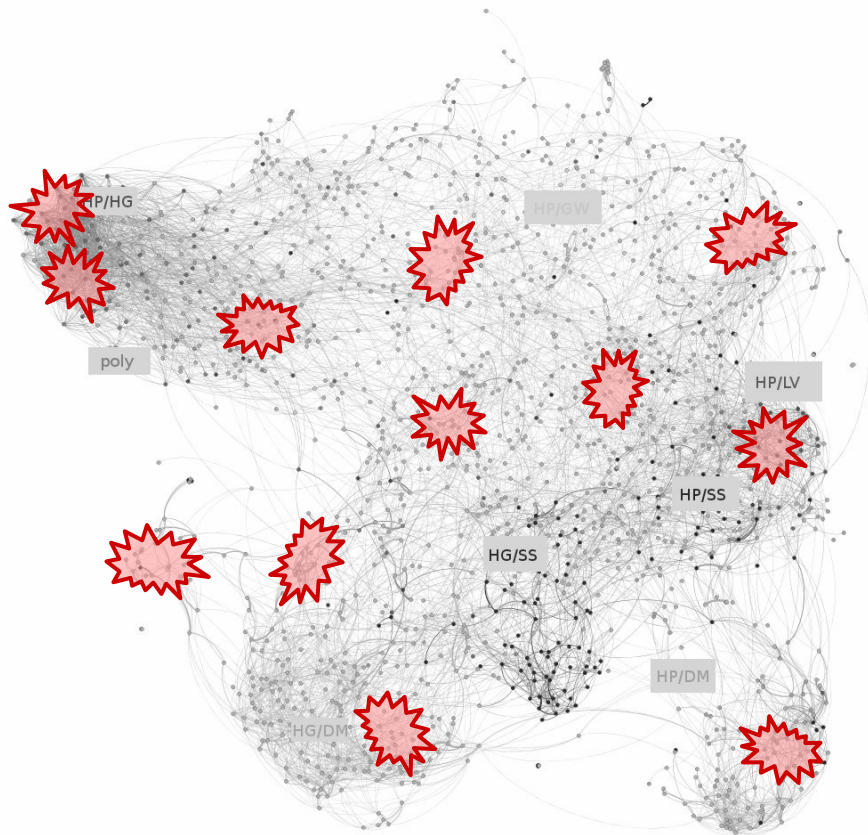
# Scaling GCNs

Traditional GCNs operate by converting the adjacency of a graph into a (sparse) matrix, and using that adjacency matrix as a gather operation to select and average one-hop neighborhoods.

This approach rapidly runs into memory issues as the size of the graph increases.

To scale to million/billion node graphs, we can sample patches of the graph (subgraphs) and train on those.

But this still poses some key challenges.

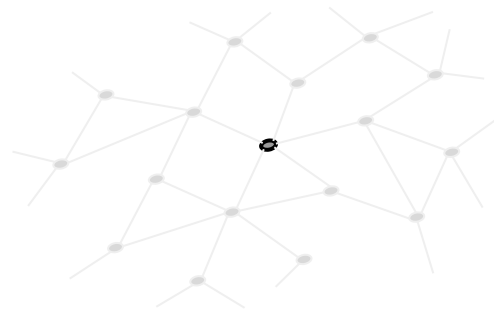


# Where GCNs Fail: Recursion

Traditional GCNs are expensive because they rely on recursive message passing.

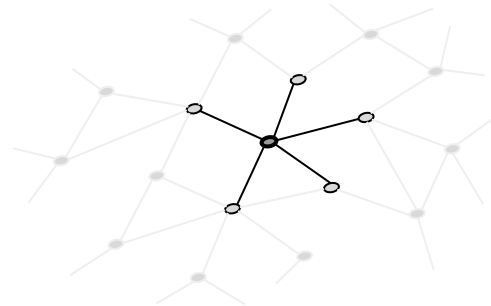
To calculate the embedding of a node, I need to get the embeddings of its neighbors...and its neighbors neighbors...and its neighbors neighbors neighbors...

This is sloooooooooooooow! If a node's first hop has 64 neighbors, and each of THOSE nodes has another 64 neighbors, you're doing 4096 IO lookups to calculate a single node.



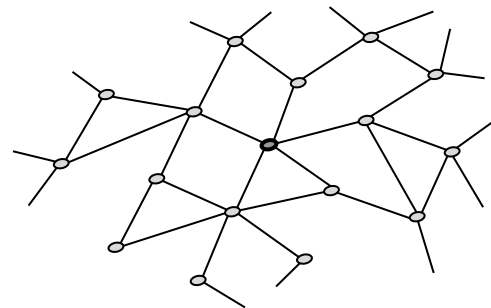
Num Hops: 0

IO Lookups: 1



Num Hops: 1

IO Lookups: 5



Num Hops: 2

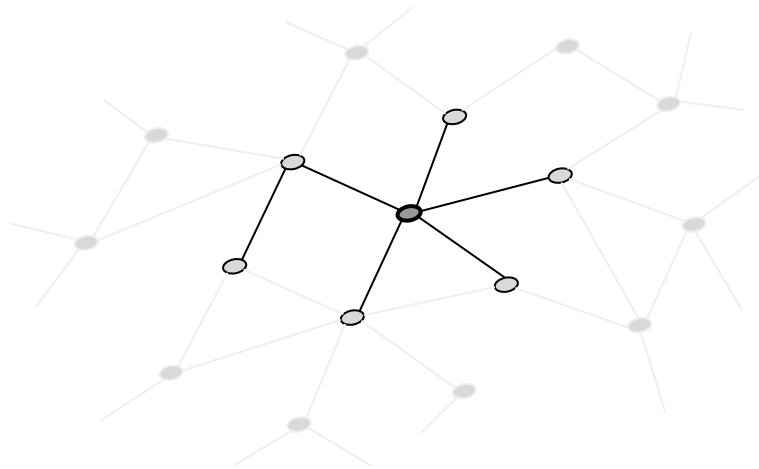
IO Lookups: 17

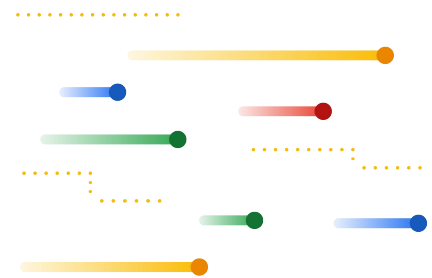
# Where GCNs Fail: Neighborhood Heuristics

Graph Convolutional Networks also bake in the assumption that all neighboring nodes are useful for the final computation.

In practice, this isn't true. Only a few neighboring nodes end up actually being important.

GCNs are effective in part because they are scattershot -- by training over all of the neighboring nodes, the GCN will pick up the important nodes by default. But this isn't scalable, especially in real world graphs where celebrity nodes can have thousands of neighbors.





# Finding PPRGo

# Key Insights

Calculating aggregations at runtime is slow, but there may be mechanisms for separating the aggregation beforehand. If we can pre-calculate aggregations offline, we can save a lot of runtime.

Whatever aggregation mechanism we use should weight nodes by importance. In other words, we don't want to blindly aggregate our node neighborhoods by doing, say, an average.

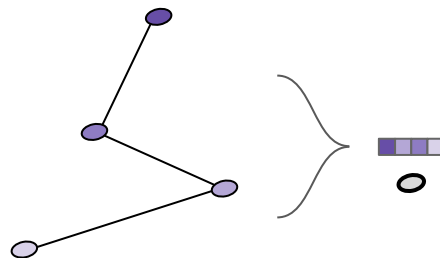
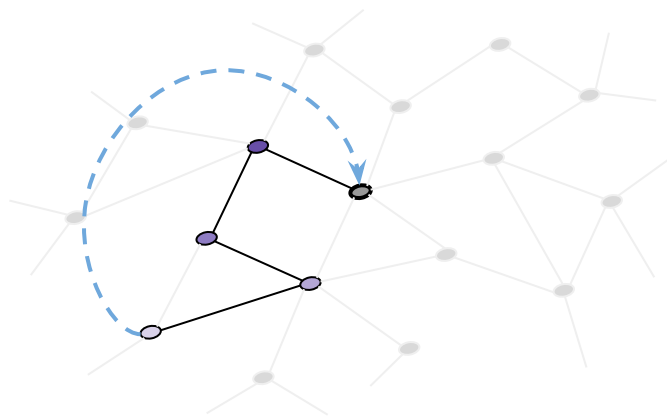


# (Approximate) Personalized Page Rank

A quick review of personalized page rank: for every node, we calculate the stationary distribution of a random walk with some teleport probability.

This gives us a weight vector of the node's neighborhood. Nodes that appear frequently in the random walk are weighted higher than nodes that rarely appear.

Intuition: for a large random walk size, this is akin to an 'infinite hop' attention vector.



# ACL and Power Iteration

We can calculate PPR in a highly scalable, distributed way (using ACL's algorithm). And we can do it offline, separated from actual model training.

During inference, we only need the PPR vector once so it's more efficient to fall back to Power Iteration. Power Iteration approach works well for large graphs with sparse adjacencies. Though expensive to calculate many times, during inference we only need  $N = 1 - 3$ .

$$\text{Computing predictions: } \underbrace{\alpha (\mathbf{I}_n - (1 - \alpha)\mathbf{D}^{-1}\mathbf{A})^{-1}}_{\text{each row is a PPR vector for one node}} \cdot \mathbf{H}$$

$$\text{Power Iteration: } Q^{(0)} = \mathbf{H}, \quad Q^{(p+1)} = \alpha\mathbf{H} + (1 - \alpha)\mathbf{D}^{-1}\mathbf{A}Q^{(p)}$$

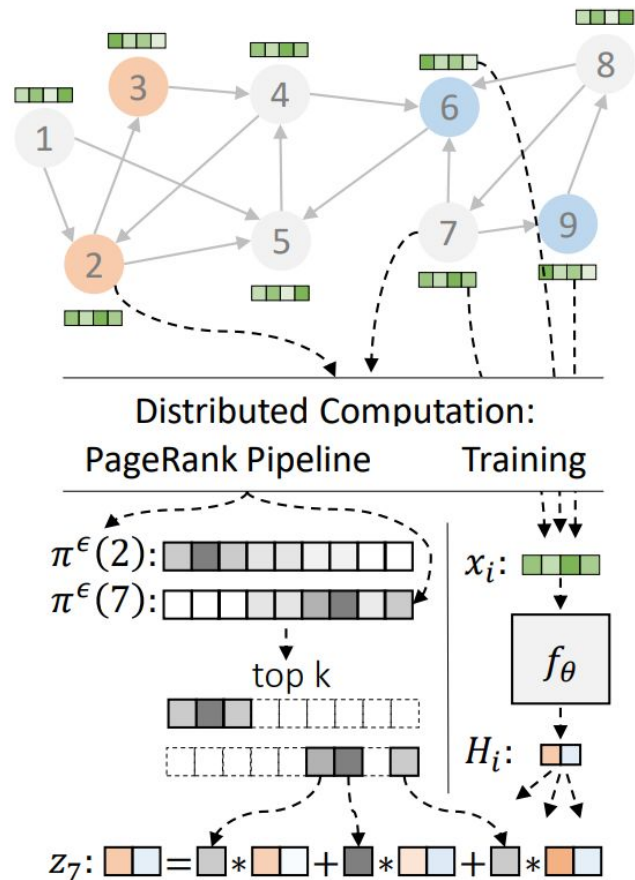
Sparse Inference: forward pass only for a fraction of nodes

$$\mathbf{H} = \begin{bmatrix} - & h_1 & - \\ - & h_2 & - \\ & \vdots & \\ - & h_n & - \end{bmatrix} \approx \begin{bmatrix} - & 0 & - \\ - & h_j & - \\ & \vdots & \\ - & 0 & - \end{bmatrix}$$

# PPRGo: Gotta Go Fast

We have the pieces necessary to create PPRGo.

- First, we calculate PPR vectors offline at scale;
- Then, we train a simple MLP model that ingests the node features and outputs logits;
- We aggregate those logits using the ‘attention’ weights of the top K nodes in the PPR vector, and use the aggregation to calculate a loss;
- At inference we use Power Iteration with ~2-3 iterations to calculate an approximated PPR vector;
- The approximated PPR is fed into the model with node features to produce a final prediction.



PPRGo aggregates the most important nodes in the N hop neighborhood using only 1 hop of computation.



# Results

# Experimental Setup

We train PPRGo and our baselines on several sparsely labeled semi-supervised node classification tasks.

For each dataset, we measure runtime as a sum of preprocessing, training, and inference. We also measure memory usage and, of course, accuracy.

Want to answer two questions:

- What is the tradeoff between accuracy and scalability?
- What is the resource consumption of PPRGo compared to other methods?

Name	CORA	PubMed	Reddit	MAG
Num Nodes	18.7K	19.7k	233K	12.4M
Num Edges	62.4K	44.3k	11.6M	173M
Num Features	8.7K	0.6k	602	2.8M

# PPRGo: Academic Dataset Analysis

**Table 2: Single machine runtime (s), memory (GB), and accuracy (%) for different models and datasets using  $20 \cdot \text{\#classes}$  training nodes. PPRGo shows comparable accuracy and scales much better to large datasets than its competitors.**

	Cora-Full			PubMed			Reddit		
	Time	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.
Cluster-GCN	84(4)	2.435(18)	58.0(7)	54.3(27)	1.90(3)	74.7(30)	2310(50)	21.04(15)	17.1(8)
SGC	92(3)	3.95(3)	58.0(8)	5.3(3)	2.172(4)	75.7(23)	7780(140)	10.15(3)	12.1(1)
APPNP	10.7(5)	2.150(19)	62.8(11)	6.5(4)	1.977(4)	76.9(26)	-	OOM	-
PPRGo ( $\epsilon = 10^{-4}, k = 32$ )	25(3)	1.73(3)	61.0(7)	3.8(9)	1.626(25)	75.2(33)	16.8(17)	5.49(18)	26.6(18)
PPRGo ( $\epsilon = 10^{-2}, k = 32$ )	6.6(5)	1.644(13)	58.1(6)	2.9(5)	1.623(17)	73.7(39)	16.3(17)	5.61(6)	26.2(18)

# PPRGo: In Depth Runtime Analysis on Reddit

**Table 1: Breakdown of the runtime, memory, and predictive performance on a single machine for different models on the Reddit dataset. We use 820 ( $20 \cdot \text{\#classes}$ ) nodes for training. We see that PPRGo has a total runtime of less than 20 s and is two orders of magnitude faster than SGC and Cluster-GCN. PPRGo also requires less memory overall.**

	Runtime (s)						Memory (GB)		Accuracy (%)	
	Preprocessing	Training		Inference		Total	RAM	GPU		
		Per Epoch	Overall	Forward	Propagation	Overall				
Cluster-GCN	1175(25)	4.77(12)	953(24)	-	-	186(21)	2310(40)	20.97(15)	0.071(6)	17.1(8)
SGC	313(9)	0.0026(2)	0.53(3)	-	-	7470(150)	7780(150)	10.12(3)	0.027	12.1(1)
PPRGo (1 PI step)	2.26(4)	0.0233(5)	4.67(10)	0.341(9)	5.85(3)	6.19(4)	13.10(7)	5.560(19)	0.073	26.5(19)
PPRGo (2 PI steps)	2.22(12)	0.021(3)	4.1(7)	0.43(8)	10.1(14)	10.5(15)	16.8(17)	5.42(18)	0.073	26.6(18)



# PPRGo: MAG

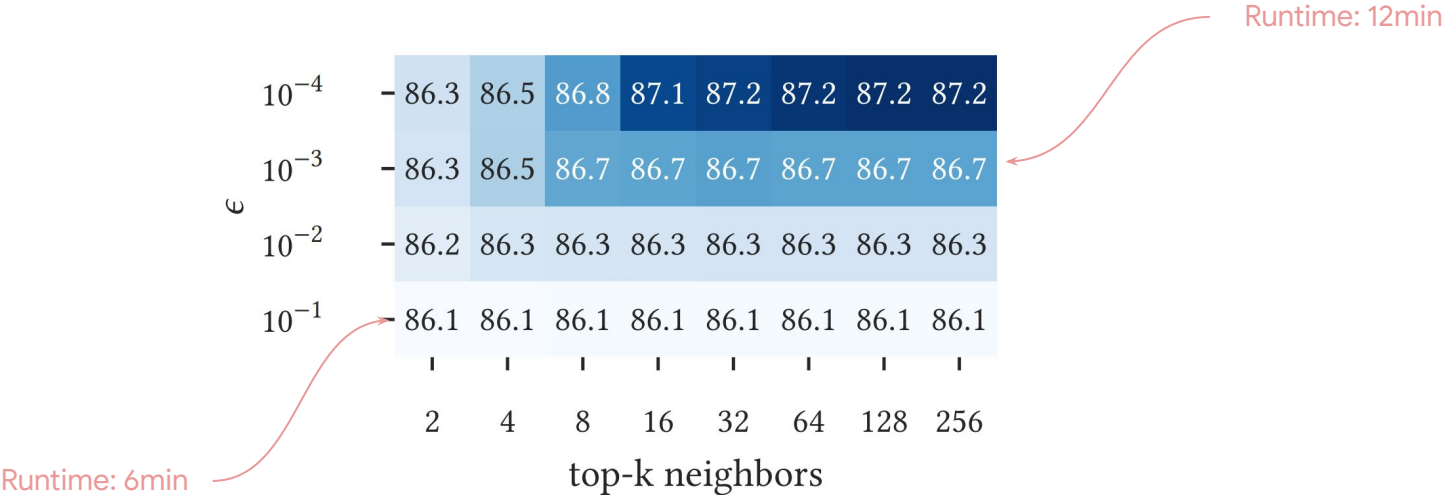
PPRGo works best on real world graphs. Most academic graphs are tiny -- a few hundred thousand nodes is hardly a reasonable comparison point.

We created a novel dataset, the MAG Scholar Citation Dataset. With 12M nodes and 173M edges, we start getting close to Google scale.

PPRGo finishes training on this dataset in < 2 minutes. It's the only method that actually finishes.

	<b>MAG-Scholar-C</b>		
	Time	Mem.	Acc.
Cluster-GCN	>24h	-	-
SGC	>24h	-	-
APPNP	-	OOM	-
PPRGo ( $\epsilon = 10^{-4}, k = 32$ )	98.6(17)	24.51(4)	69.3(31)
PPRGo ( $\epsilon = 10^{-2}, k = 32$ )	89(5)	24.49(5)	63.4(29)

# PPRGo: Distributed MAG -- TradeOffs



**(b) Setting with a large number of labeled nodes (105415 nodes, 1%)**

# PPRGo: Distributed MAG -- Efficiency

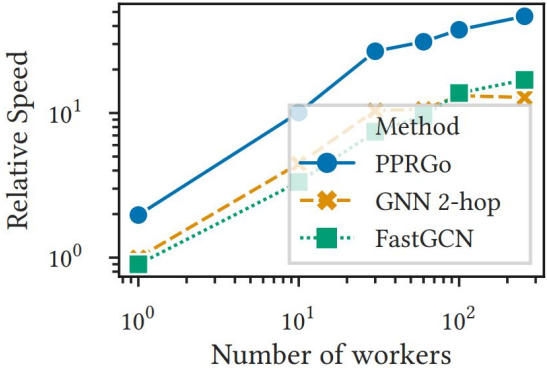


Figure 4: Relative speed in terms of number of gradient-update steps per second on the MAG-Scholar-F graph compared to the baseline method (GNN 2-hop, single worker). Both axes are on a log scale. PPRGo is consistently the fastest method and can best utilize additional workers.

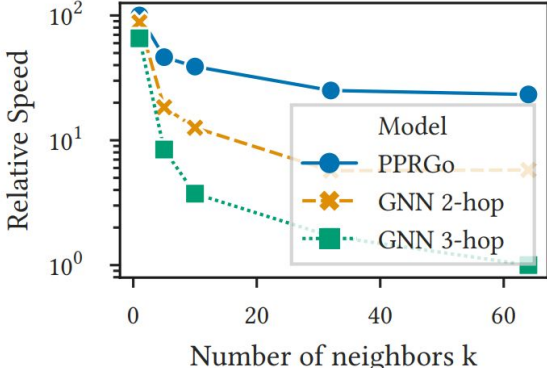
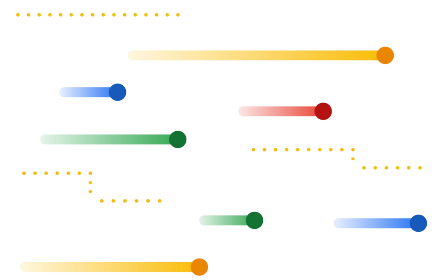


Figure 5: Relative speed comparison (num. gradient updates per second) between PPRGo and multi-hop models for different values of k on MAG-Scholar-F. Distributed training.



# Conclusions

Graph Convolutional Networks are powerful because they let us incorporate node neighborhoods, but they do so in an expensive, scattershot way.

PPRGo gives us the benefit of large neighborhood learning with the speed of a single hop GNN in a trivially distributable manner.

PPRGo can operate on actual large scale graphs, including Google scale graphs. It was the only learning tool that successfully completed the MAG dataset, the largest public academic graph that we are aware of to date.

# Citations

## Based on Work By:

Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, Stephan Günnemann. **Scaling Graph Neural Networks with Approximate PageRank**. KDD'20. <https://arxiv.org/abs/2007.01570>

Reid Andersen, Fan Chung, Kevin Lang. **Local Graph Partitioning using PageRank Vectors**. <http://www.leonidzhukov.net/hse/2015/networks/papers/andersen06localgraph.pdf>

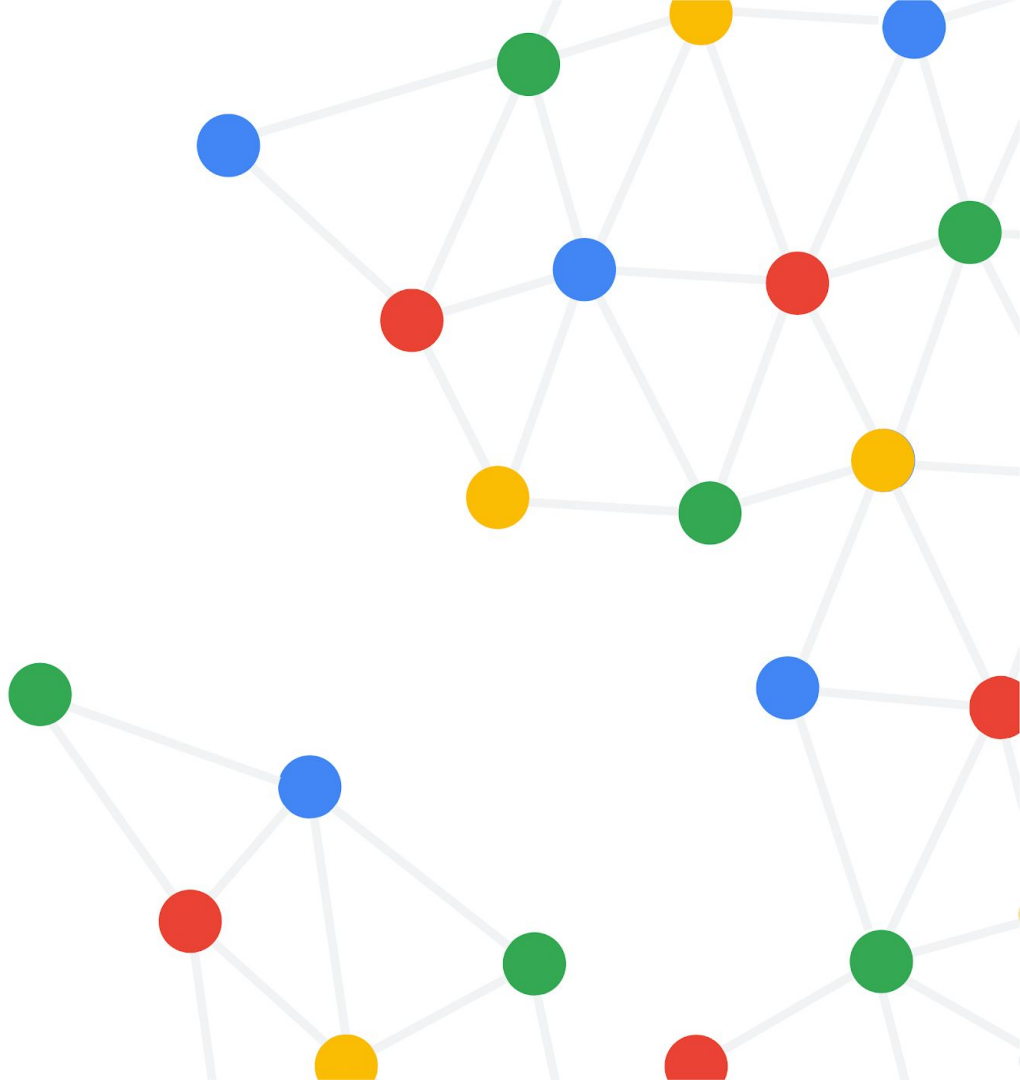
## ICONS:

Mind: <https://thenounproject.com/search/?q=deep+learning+graph&i=1705433>

## OTHER:

CC3.0: <https://creativecommons.org/licenses/by/3.0/us/legalcode>

CC2.5: <https://creativecommons.org/licenses/by-nc/2.5/>







# Learning Multiple Embeddings

Alessandro Epasto

Based on: Epasto, Perozzi. “Is a Single Embedding Enough? Learning Node Representations that Capture Multiple Social Contexts” **WWW 2019**

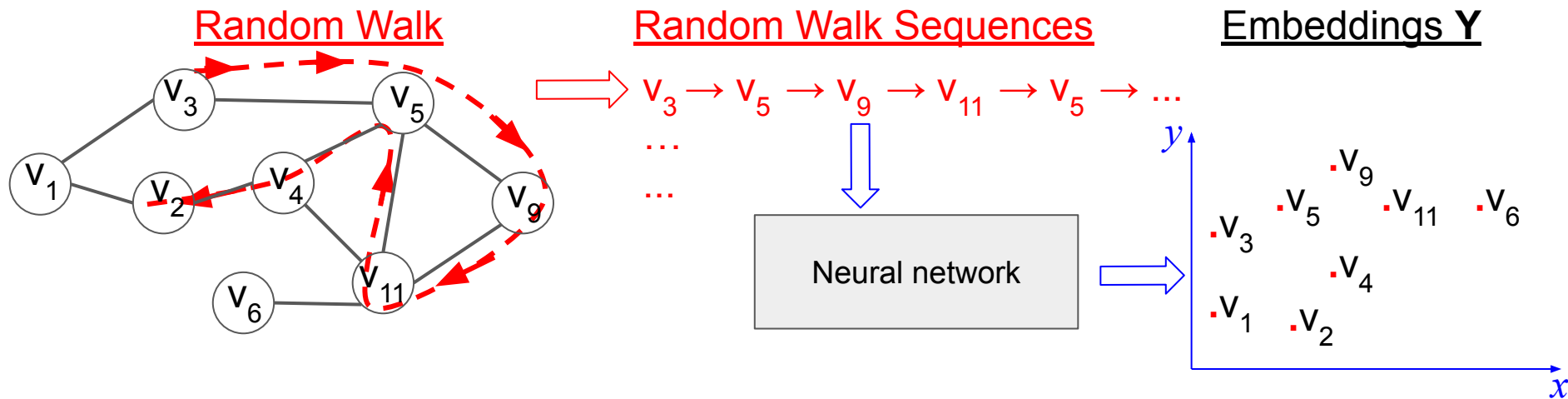
# Node Embeddings -- Why do we need them?

Graphs contain discrete information (nodes, edges).

- Most modern Machine Learning (ML) techniques operate on *continuous* inputs.
- Graph Embeddings are *continuous* representation of graphs.
- Useful for various problems, including:
  - Node Classification
  - Edge Classification
  - Link Prediction

# Review: Node Embedding via Random Walks

Deepwalk [2]: simulate random walks then encode them with neural network.

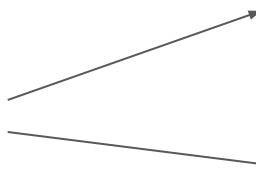


[2] Perozzi et al. DeepWalk: Online Learning of Social Representations. KDD 2014

# Is a single embedding enough?

In **Natural Language Processing** there are disadvantages to using a single embedding to represent a word.

“Lets sit by the **bank**”



A financial institution?



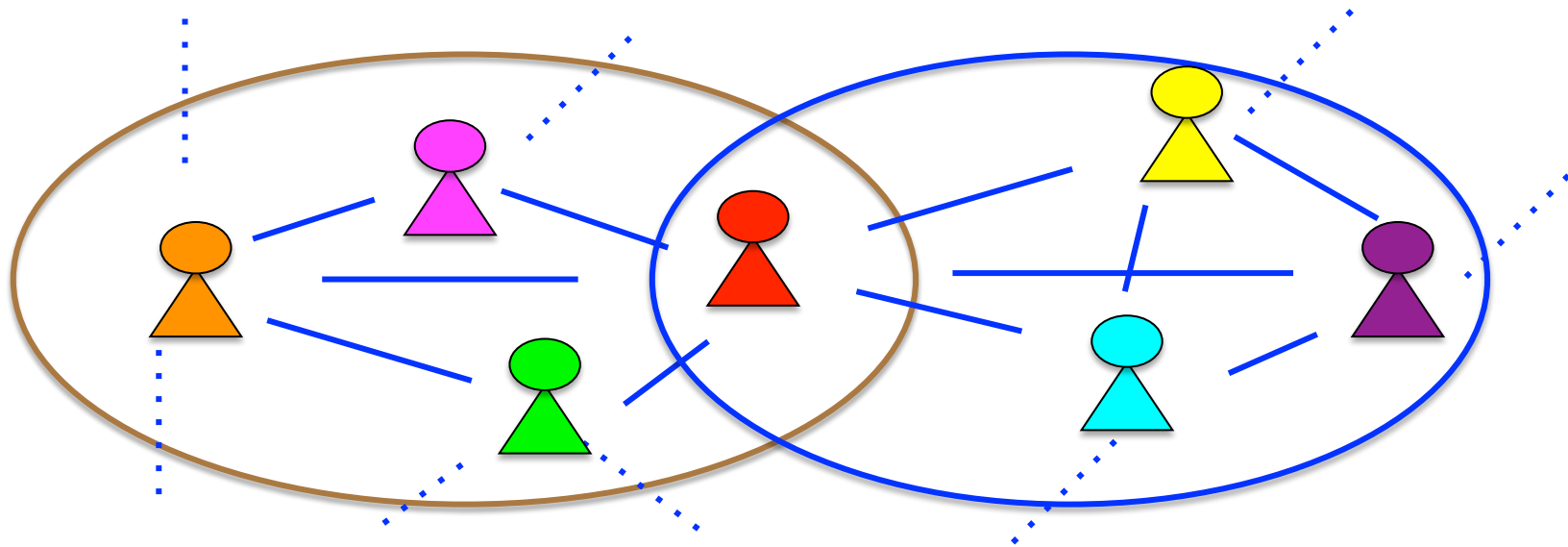
A river side?



**Main observation:** Graphs have this problem too!

In a social network, nodes belong to multiple overlapping communities

More connections with outside than with inside



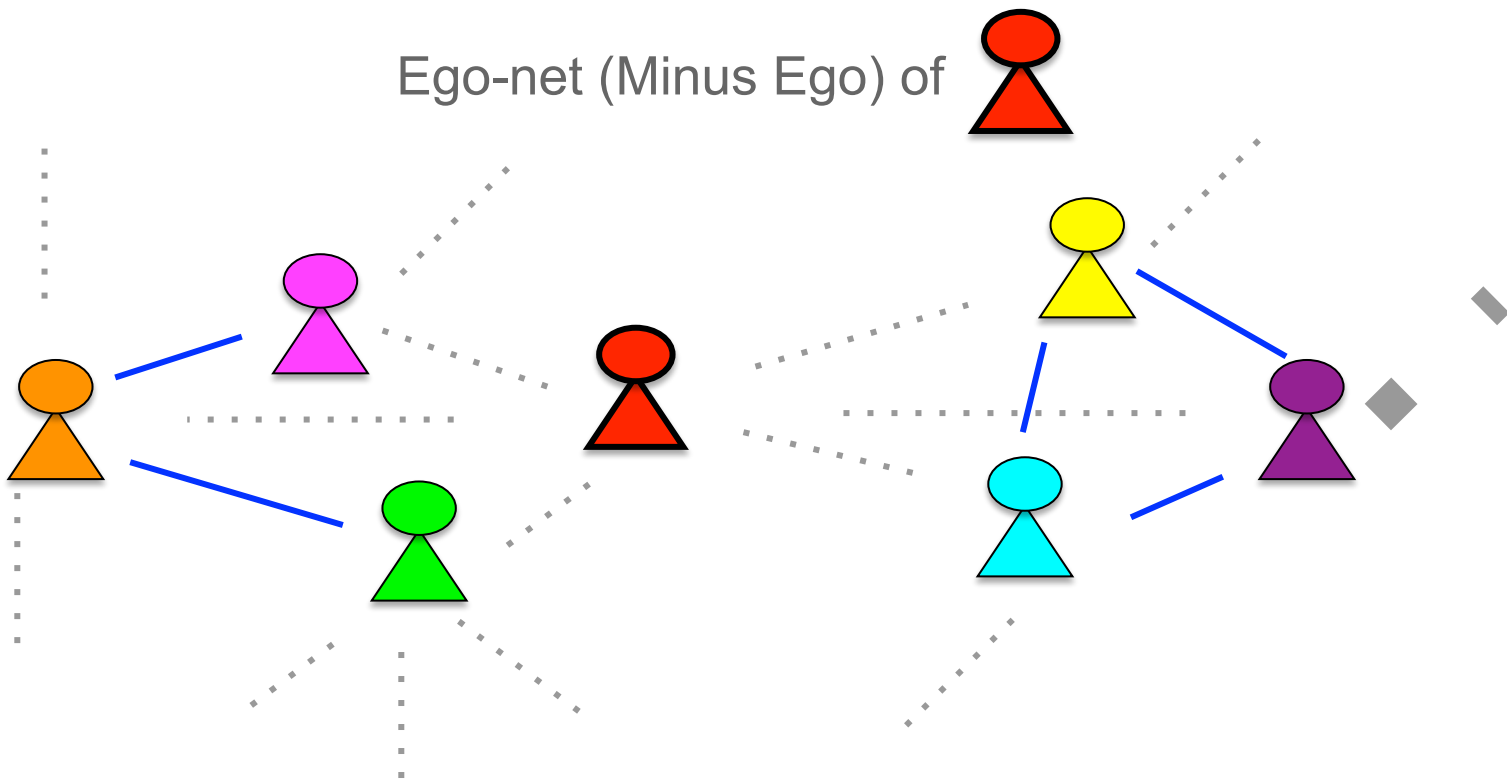
Communities **overlap** heavily.

**Large cut**

Random walks **will cross** the community boundaries **very often**. Each node has **many roles** and belongs to **many communities** that the random walk will explore partially.

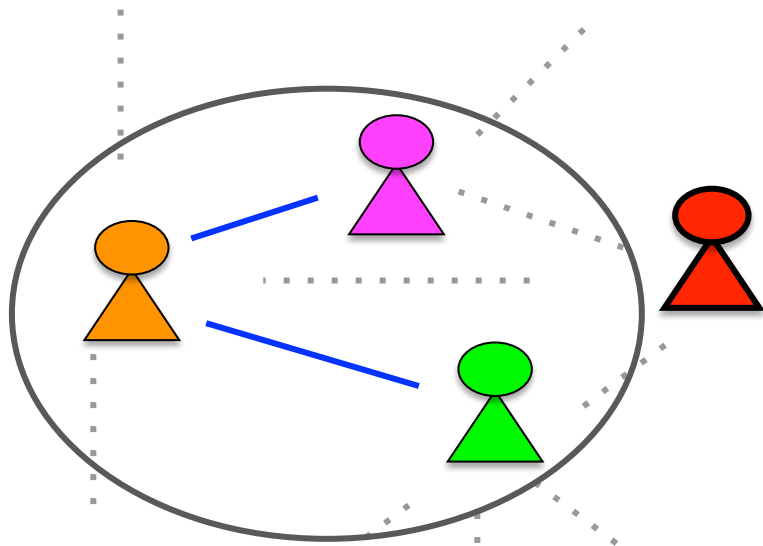
# Cluster Locally, Embed Globally

**Solution:** Community structure is more **clear** at the **microscopic** level of node-centric structures called **ego-networks**. Analyzing the ego-nets allows to disentangle the communities.

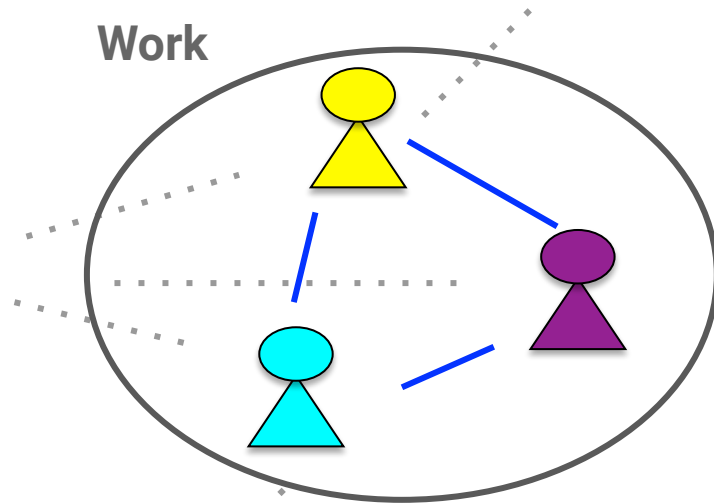


The **Ego-net (minus Ego)** of node  $u$ , is defined as the induced subgraph on  $\{N(u)\}$ .

Family



Work



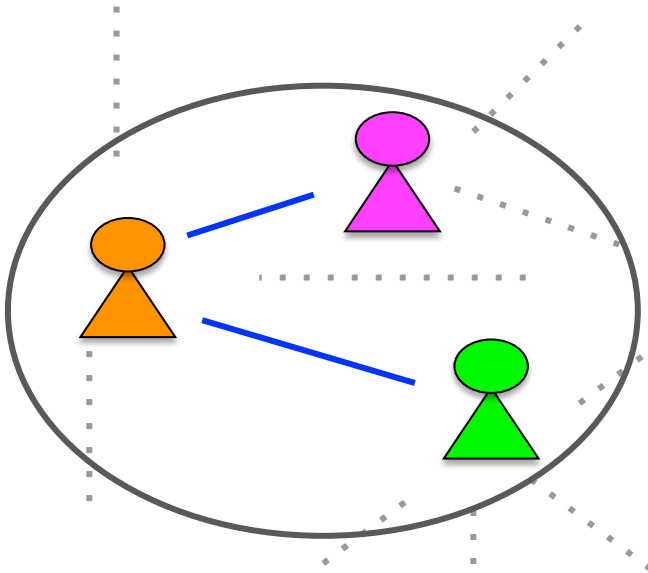
**Intuition:** while communities overlap, usually there is a **single context** in which **two** neighbors interact.



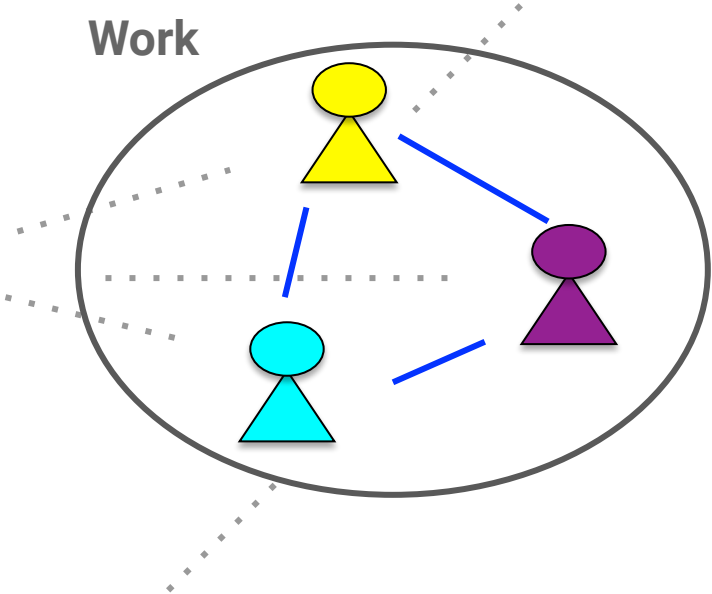
The persona graph, a novel graph concept based on ego-net analysis with applications in overlapping graph clustering, graph embeddings and more.

- **Highly flexible**, allows use of **any non-overlapping** algorithm
- **Scalable** (tens of billions of nodes and edges)
- Provable theoretical guarantees for graph clustering

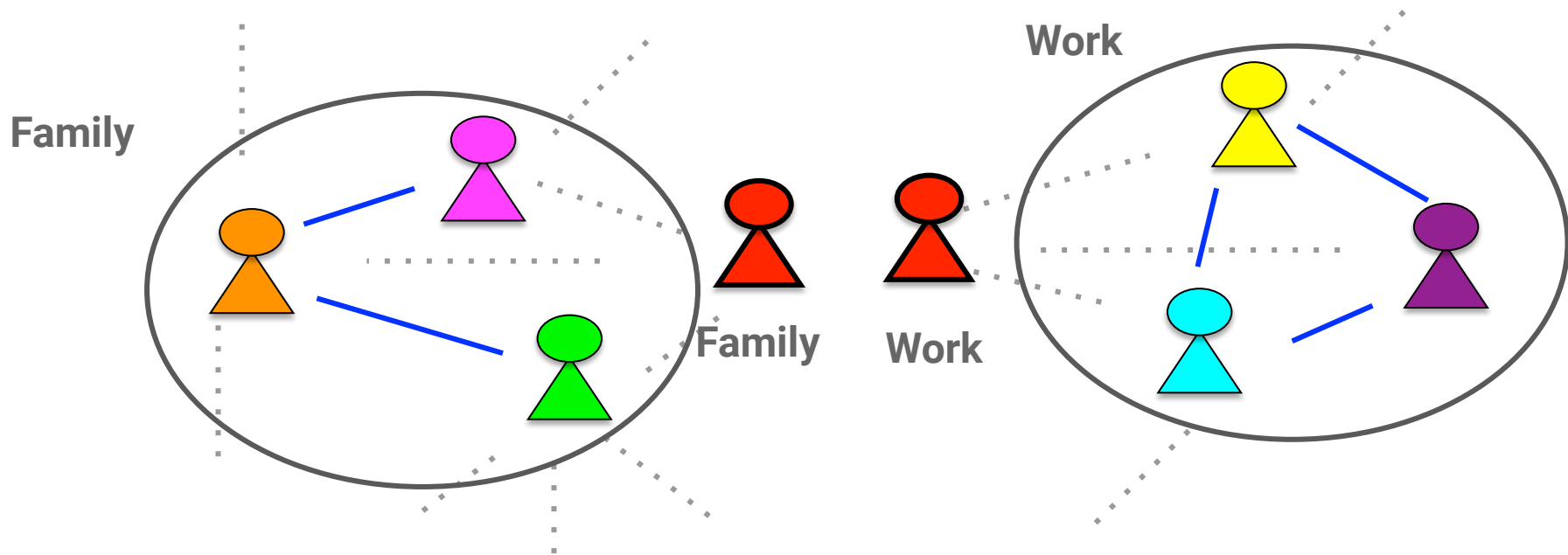
Family



Work



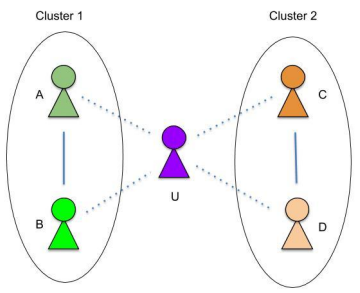
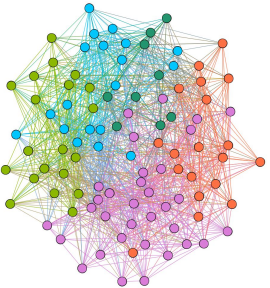
Intuition: the red node is *actually* **two** nodes which we call the persona nodes of the node.



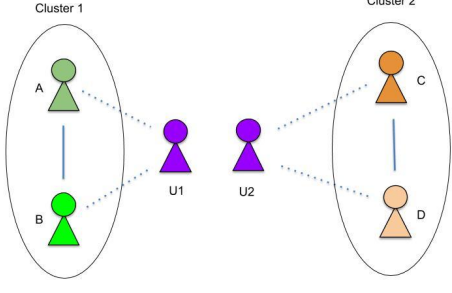
We create a Persona Graph where these two nodes are separated and we split the edges of the original node among the persona nodes.

More formally the persona graph proceeds in the following steps:

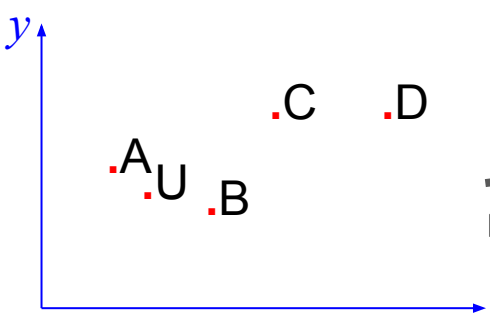
1. Create the ego-net of each node
2. Partition each ego-net with *any* non-overlapping clustering algorithm
3. Create the persona graph
4. Analyze the persona graph (e.g. embed the nodes)
5. Map the results of the persona graph to the original graph



Ego-Net Analysis

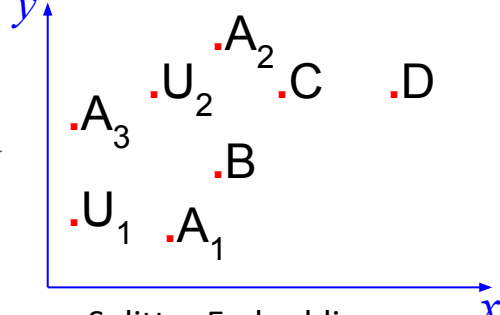


Persona Graph



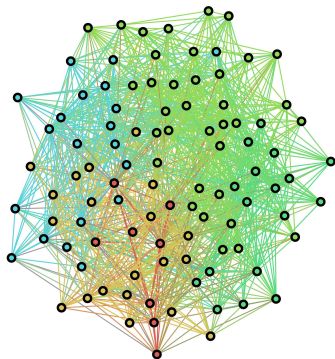
Normal Embedding

Regularization

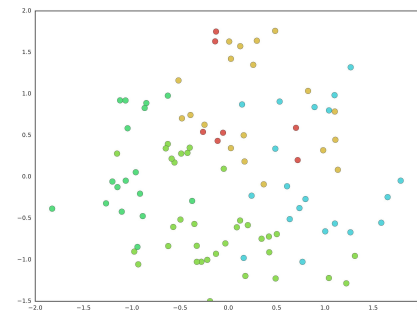


Splitter Embedding

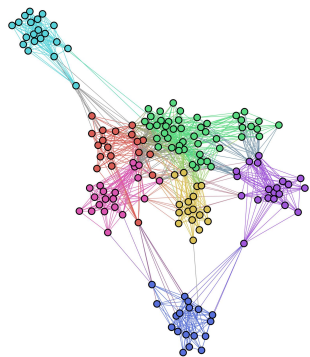
Original Graph



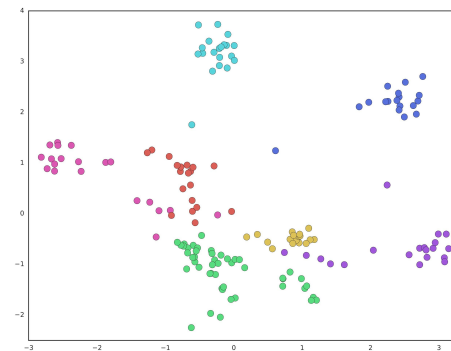
node2vec



Persona Graph

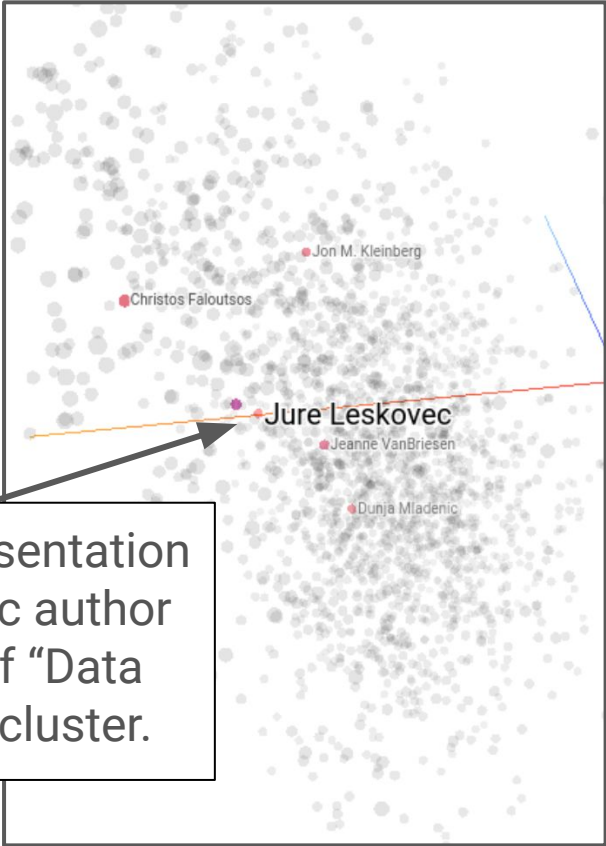


splitter

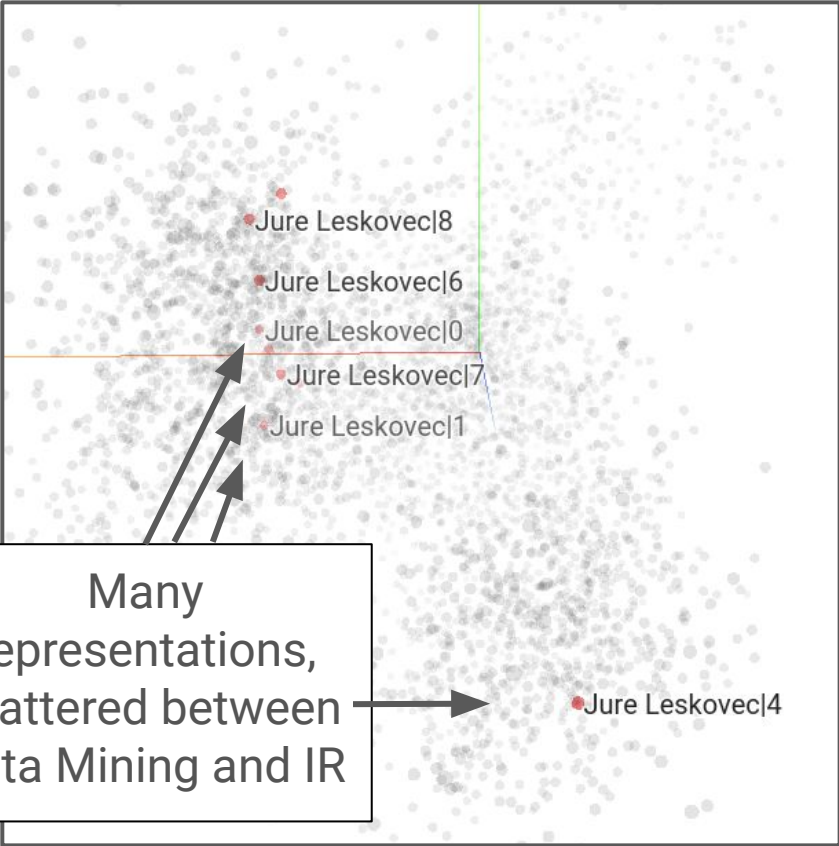


Dataset	$d_{max} = 16\bar{p}$	Best EigenMaps	Best Node2Vec	Best DNGR	Best Asymmetric	Best M-NMF	SPLITTER $d = 16$
soc-epinions	48.5	†	0.726	†	0.700	†	<b>0.974</b>
wiki-vote	64.0	0.613	0.643	0.630	0.702	0.932	<b>0.952</b>
ca-HepTh	38.2	0.802	0.886	0.868	0.885	<b>0.912</b>	0.897
ca-AstroPh	40.5	0.824	0.934	0.939	0.942	0.966	<b>0.972</b>
ppi	79.5	0.737	0.733	0.769	0.813	0.840	<b>0.869</b>

We use the simple max aggregation of dot products for link prediction using persona embeddings.



One representation for prolific author inside of "Data Mining" cluster.



Many representations, scattered between Data Mining and IR



1. How do the multiple meanings relate to each other?
  
2. How can we use them in different tasks?
  - a. Link prediction
  - b. Node classification → ???

# Algorithms, systems and scalability

Martin Blais, Jakub Łącki

Graph Neural Networks in  
Tensorflow

Graph algorithms in the  
distributed setting

Multi-core parallel graph  
clustering



# Graph Neural Networks in TensorFlow (a.k.a. “*Graph Tensor*”)

Martin Blais (blais@google.com)

# Motivation

Common infrastructure for building GNNs on TensorFlow.

Why?

- Consulting with many internal teams within the company working with GNNs, we realized that a significant portion of development time was spent on **data representation**.
- After our second system, we realized the **right shape** that the third version should have (and this is it).

# Goal

*“Build the ultimate toolkit for building and training GNN models on very large graphs on top of TensorFlow.”*

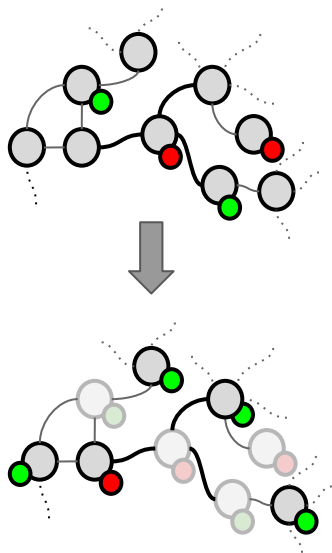
- Supports many model types, graph types, arbitrary feature shapes
- **Scalable and distributed by default**
- Handles irregular representation, sampling and I/O out of the box
- Integrates well with [TensorFlow](#)

*This is a preview; we're working on an open source release*

# Types of Models

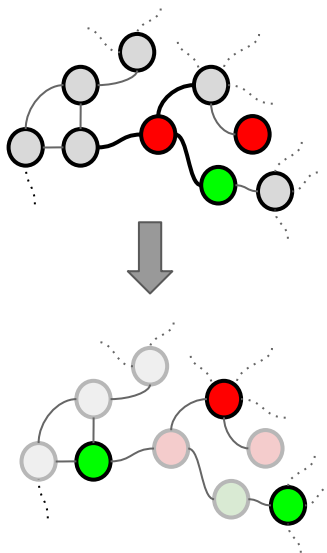
## Supervised models

Classification from surrounding neighborhood features



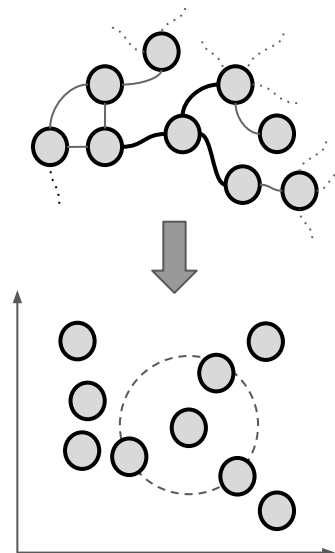
## Semi-supervised models

Learn from propagating labels from the neighborhood



## Unsupervised models

Train node-level embeddings to describe the structural role of the data (e.g. DGI)

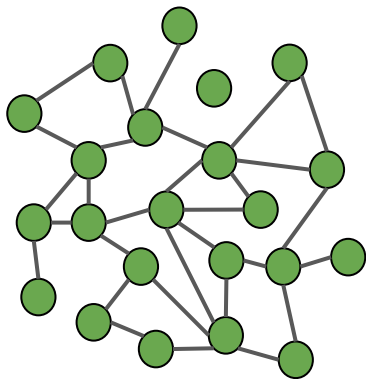


# Types of Graphs

## Homogeneous models

One type of node

One type of edge

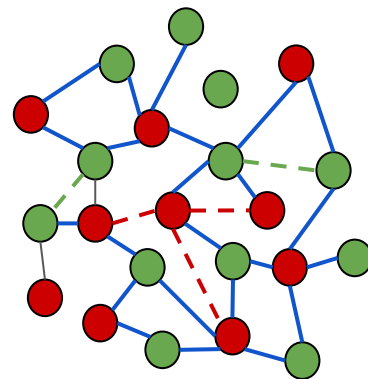
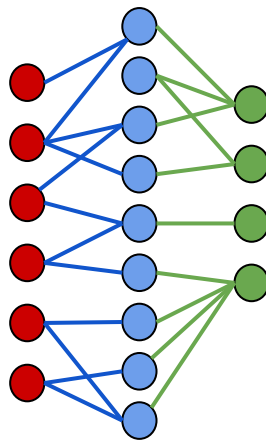


## Heterogeneous models

Multiple types of nodes

Multiple types of edges

Directed or undirected



# Types of Features

Features can be attached to

- Node sets
- Edge sets
- Graph

Arbitrary shapes

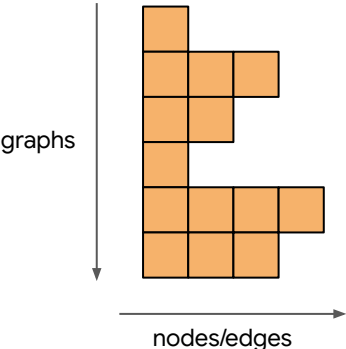
- scalar
- dense
- ragged

- Provides RaggedTensors
- Labels are just features
- Supports latent nodes with no features

(Multiple features per set)

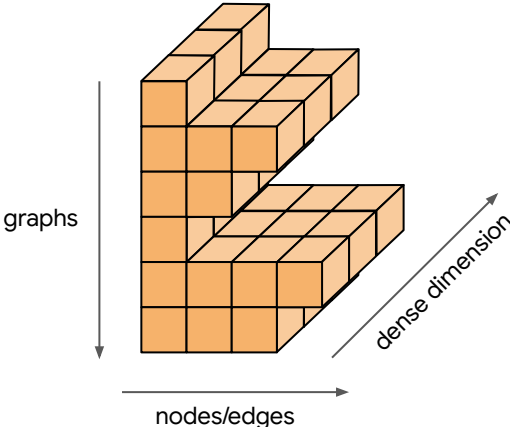
**scalar features**

e.g. edge weights



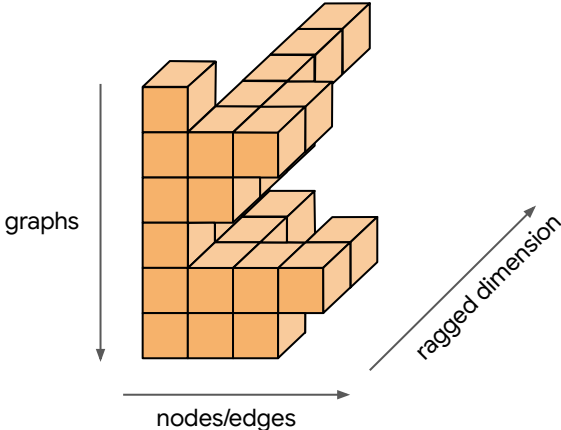
**features with rank > 1**

e.g. embeddings



**variable-shaped features**

e.g. sentences of words





# Graph Schema

**nodes**

**edges**

```
node_sets {
  key: "user"
  value {
    description: "An end user who watches videos."

    features {
      key: "account_age"
      value: {
        description: "The number of days since account was created."
        dtype: DT_INT64
      }
    }
  }
}
```

```
node_sets {
  key: "video"
  value {
    description: "Unique video content."

    features {
      key: "title"
      value: {
        description: "The title of the video (bag of words)."
        dtype: DT_STRING
        shape { dim { size: -1 } }
      }
    }
    features {
      key: "days_since_upload"
      value: {
        description: "The number of days since upload."
        dtype: DT_INT64
      }
    }
  }
}
```

```
edge_sets {
  key: "watches"
  value {
    description: "Watches of videos by users."
    source: "user"
    target: "video"
  }
}
```

```
edge_sets {
  key: "co-watch"
  value {
    description: "Co-watch similarity graph between users."
    source: "user"
    target: "user"

    features {
      key: "similarity"
      value: {
        description: "The Jaccard similarity of the video sets between users."
        dtype: DT_FLOAT
      }
    }
  }
}
```

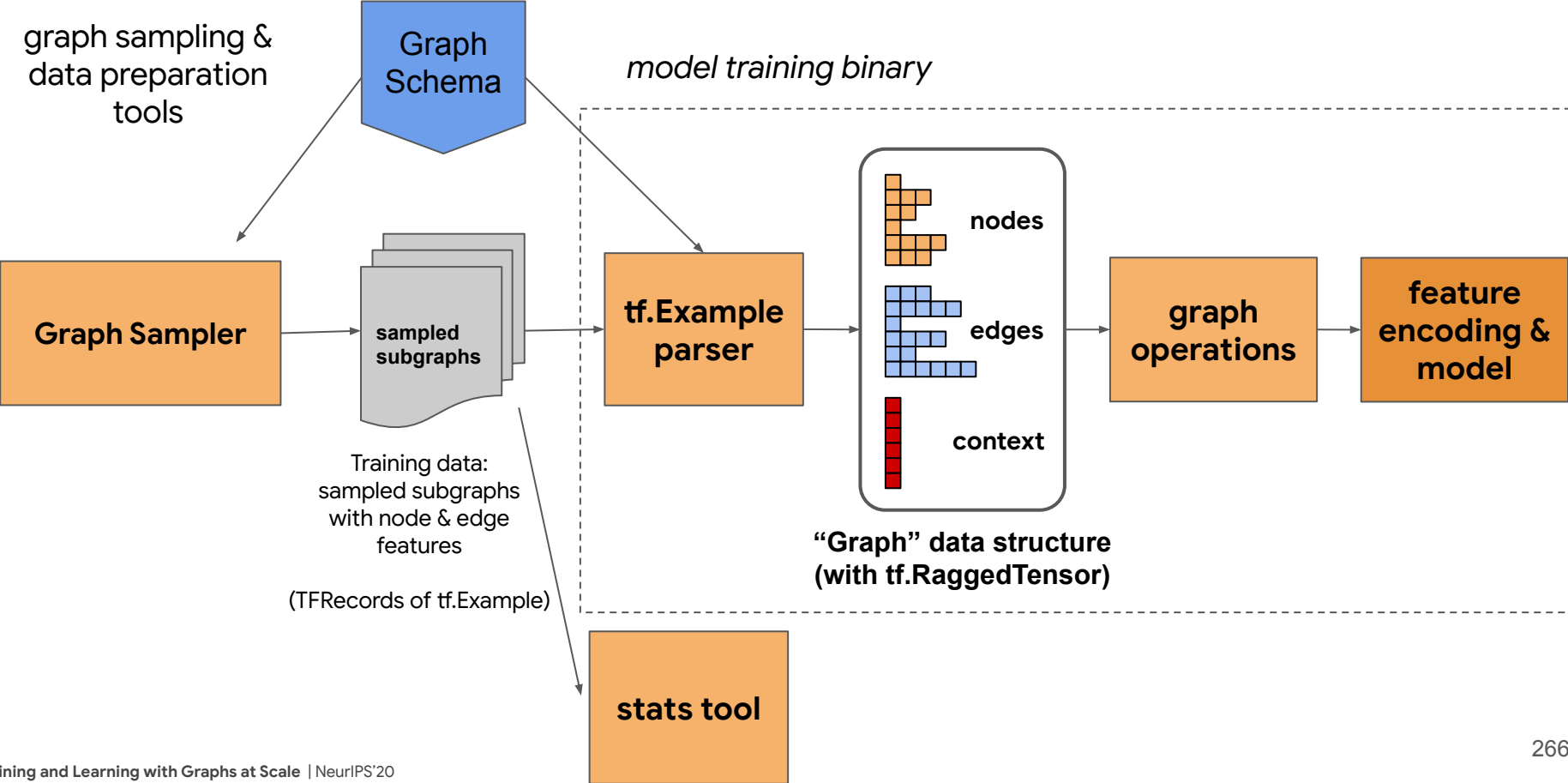
```
context {
  features {
    key: "label_class"
    value: {
      description: "A label, ground truth."
      dtype: DT_STRING
    }
  }
}
```

**source/target**

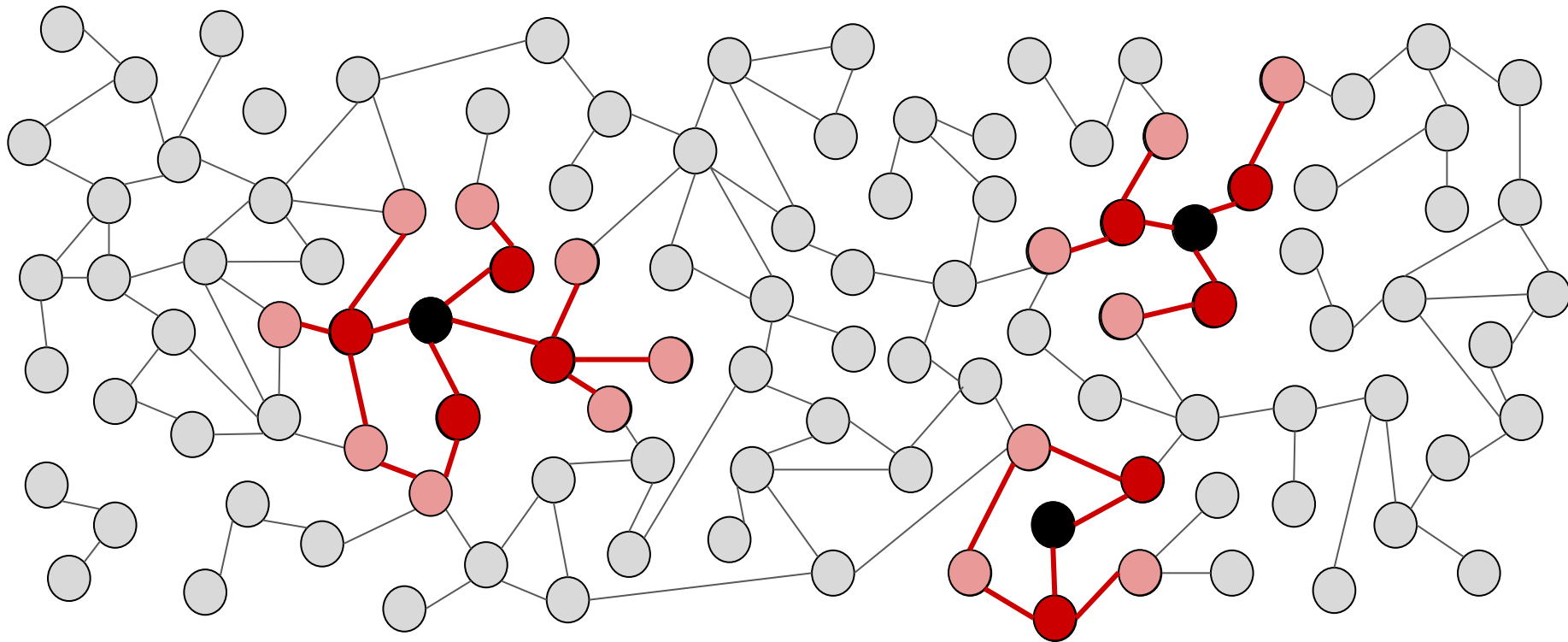
**feature declarations**

**graph**

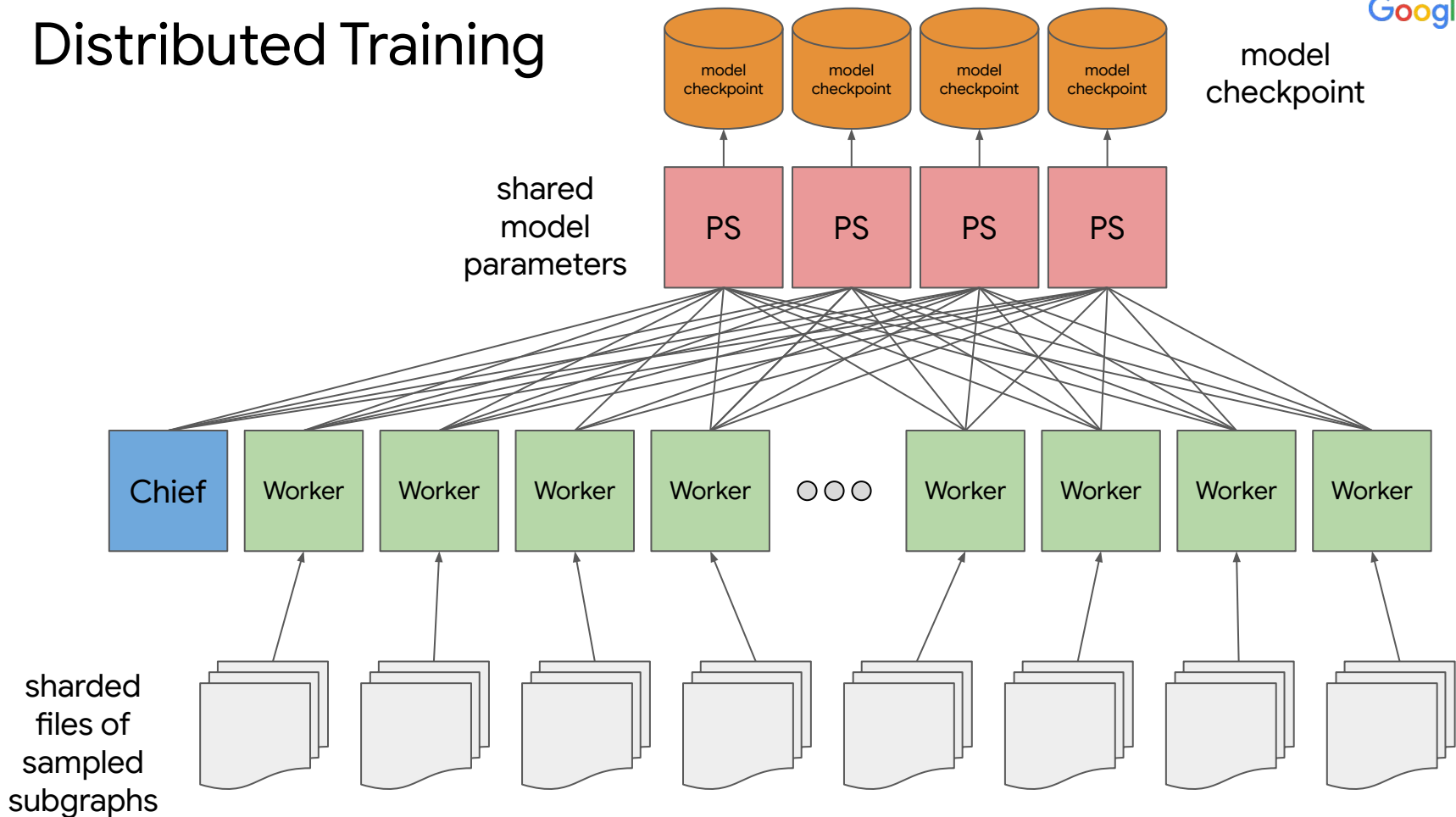
# Library Overview



# Scalability via Subgraphs on Receptive Field



# Distributed Training



# Training Data Preparation: Graph Sampling

Two common scenarios:

## 1. A graph is provided

- a. Apply region growing algorithm sampling over edges to produce subgraphs.

## 2. There is no graph;

- a. Sample references between entities from **relational tables** (i.e., a database).  
This is very common in building heterogeneous models.
- b. Build a graph (see [GRALE](#) paper), then → Process using graph sampler → (1).

External tools:

- Small graphs that fit in-memory: **Conversion from [NetworkX](#)**
- **Scalable sampler using Apache Beam** (runs on Cloud Dataflow / Spark)
- Custom converters for **public datasets**, e.g. [OGB](#) → research
- A **builder API** to implement your own encoders

# Utility Library Functions

- Extract sparse adjacency matrix
- Insert self-edges
- Convert to undirected
- Mask out some nodes, mask out some edges
- Extract seed node mask
- Simple convolution (gather + segment reduction (e.g, max, sum))
- Insert self-attention layers

... and more

# Integrations

The TF GNN library is **agnostic to model API**; integrates with

- DeepMind [GraphNets](#): Consume graph tensors → adapt to GraphsTuple
- Google [Neural Structured Learning](#) (with GraphNets)
- **TF GNN API** (own API) — Based on MPNN paper [[Gilmer 2017](#)]

The “Graph” container object is a **TF Extension Type** (a composite tensor):

- It can be passed around **Keras** layers.
- It supports **batching**, unbatching/flattening and serialization and tf.data.
- Supports custom hardware (TPUs)



# Graph algorithms in the distributed setting

Jakub Łącki

Based on joint work with Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Vahab Mirrokni, Warren Schudy, and Michał Włodarczyk



# Graphs are big



Hyperlink2012

200B edges

Web graph  
Google (2018)

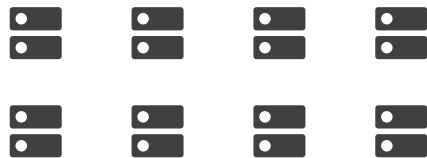
6.5T edges

Human brain

>100T connections

*How to mine graphs with billions / trillions of edges?*

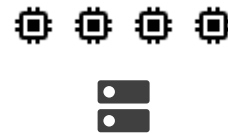
# Two approaches to processing large graphs



Distributed

up to ~10k machines  
up to ~10k CPUs  
many terabytes of RAM

**This talk**

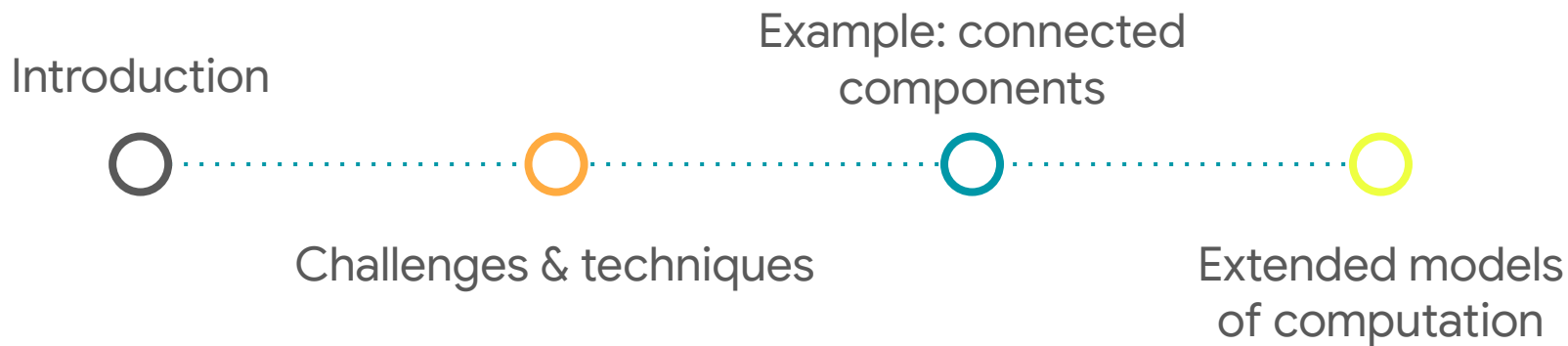


Multi-core parallel

1 machine  
100s logical CPUs  
~1TB RAM

**Next talk**

# Agenda



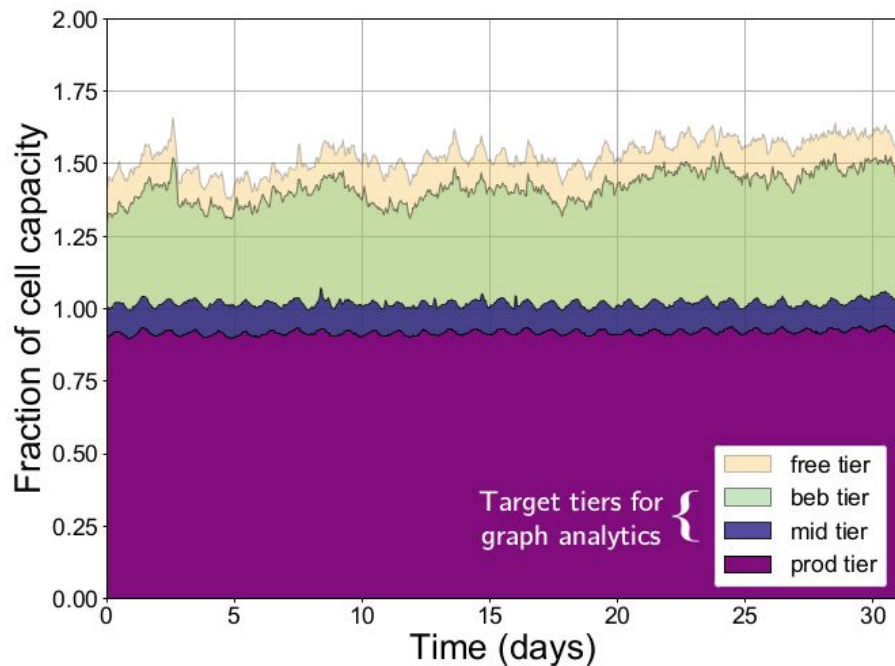
# Running in a shared datacenter

## Goals:

- Speed & scalability
- High reliability
- **Low cost**

Use whatever capacity the top tier jobs don't use:

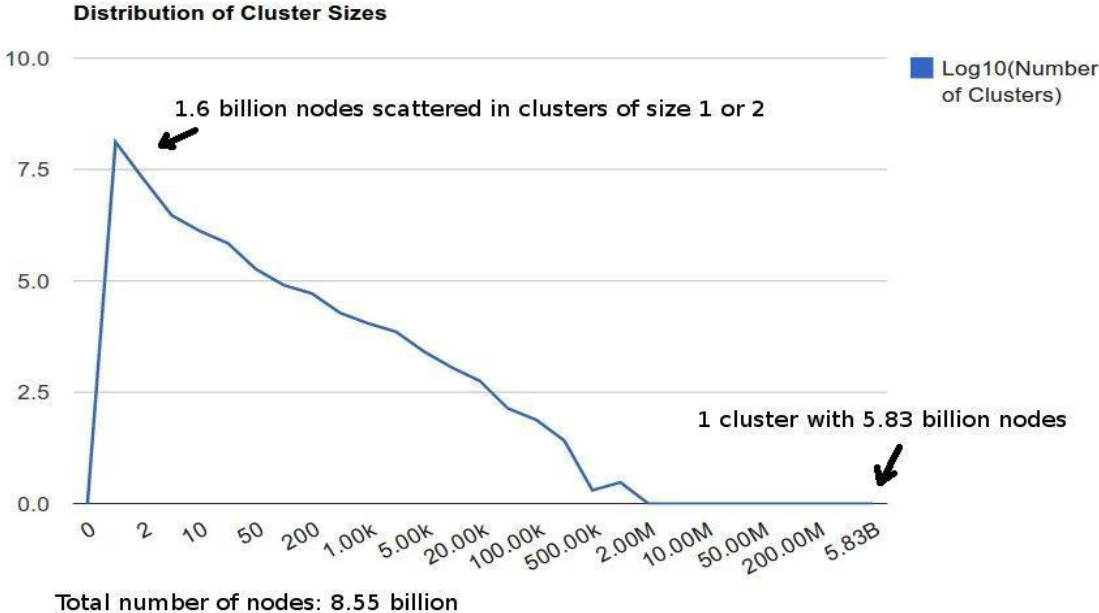
- **Very common** failures due to preemptions
- All data must be saved to disk
- Lower resource cost



*Borg: the Next Generation.* Tirmazi, Barker, Deng, Haque, Qin Hand, Harchol-Balter, Wilkes, EuroSys'20.

# Challenges of big graphs - data skew

Cluster sizes in a web graph (8.5B nodes, 700B edges)

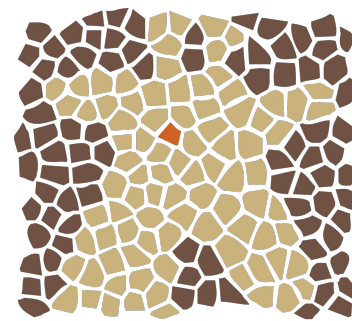


# Distributed computation frameworks

- Many popular frameworks
  - MapReduce / Hadoop  
[DeanG, OSDI'04]
  - Pregel / Giraph  
[MalewiczABDHLC, SIGMOD'10]
  - Beam / Flume / Cloud Dataflow  
[AkidauBCC+, VLDB'15]
  - ...
- High-level abstraction over distributed setting
- Fault tolerance
  - Computation (mostly) in synchronous rounds
  - Different checkpointing strategies
- All provide a very similar model of computation



beam

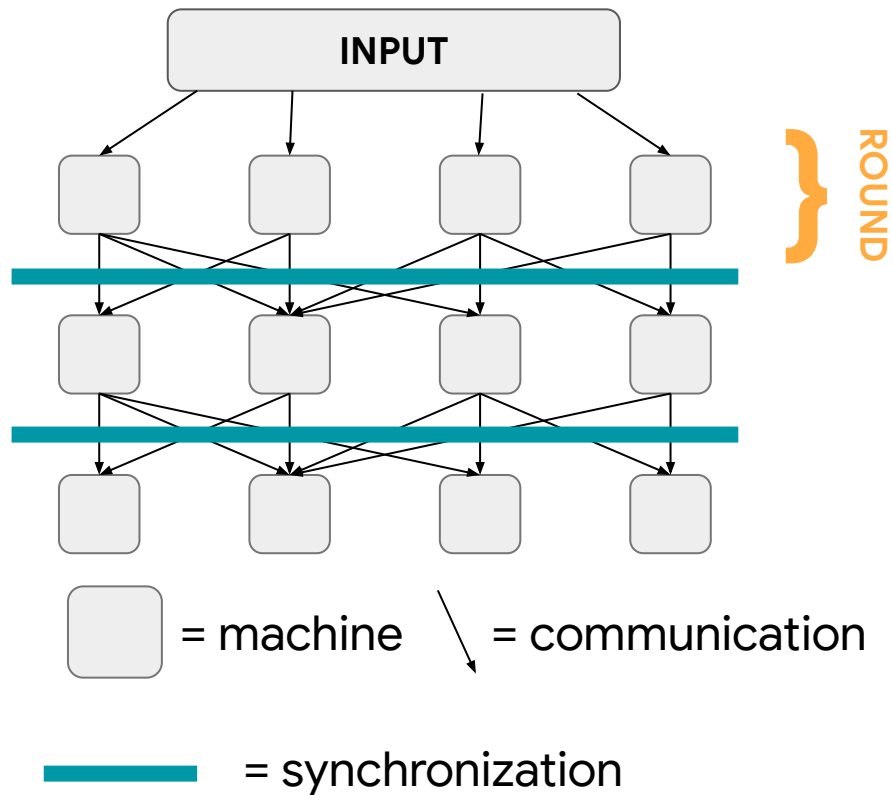


A P A C H E  
G I R A P H



# Distributed computation in practice

- Computation in synchronous rounds
- In each round, a machine:
  1. Receives messages from previous rounds
  2. Performs arbitrary computation
  3. Sends messages to other machines
- All communicated data saved to persistent storage (fault tolerance)



# Distributed computation - desirable features



## Running time

Low number of rounds  
(ideally  $O(1)$  or  $O(\log n)$ )

Each machine takes  
near-linear time in the  
input size



## Load balancing

No machine is  
overloaded



## Communication

Linear communication per  
round

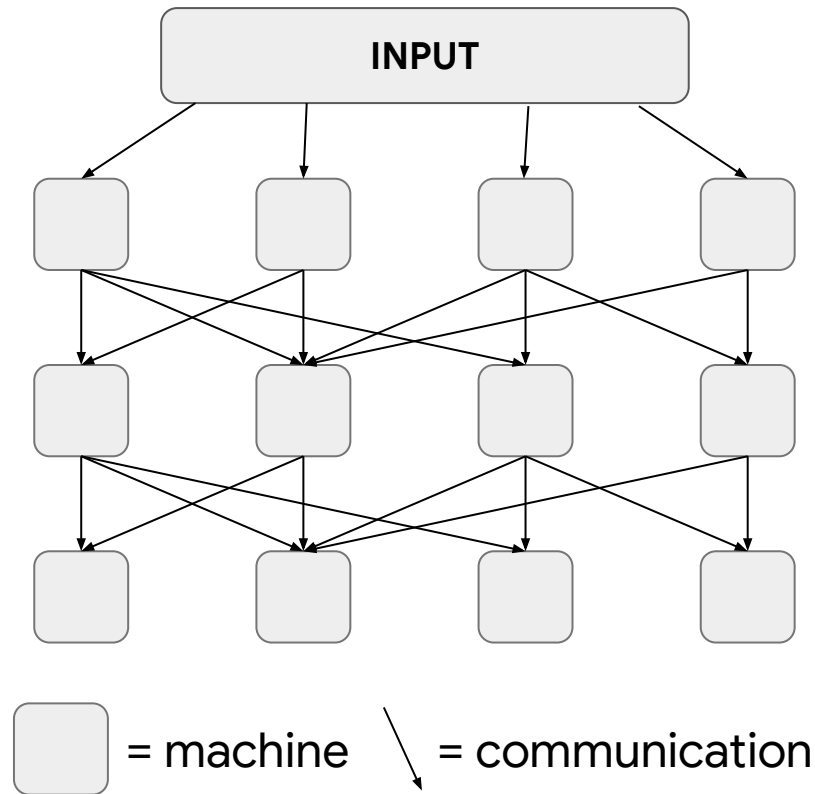
Communication balanced  
among machines



# MPC model

*A Model of Computation for MapReduce.* KarloffSV, SODA'10

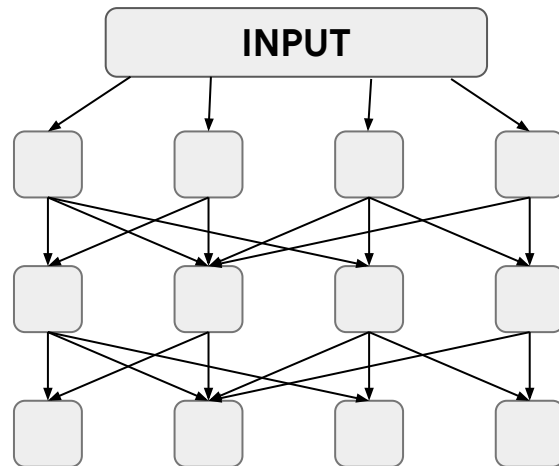
- MPC = Massively Parallel Computation
- Input of size  $N$
- $M$  machines with space  $S$
- $N \approx M \cdot S$ 
  - Machines can (barely) store the input
- $S = N^\epsilon$  for some  $\epsilon \in (0, 1)$ 
  - Each machine can see a small fraction of the input



# MPC model

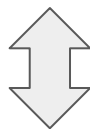
*A Model of Computation for MapReduce.* KarloffSV, SODA'10

- **Key restriction (load balancing)**
  - each machine sends/receives data of size  $O(S)$  in each round
- **Goal**
  - Minimize #rounds

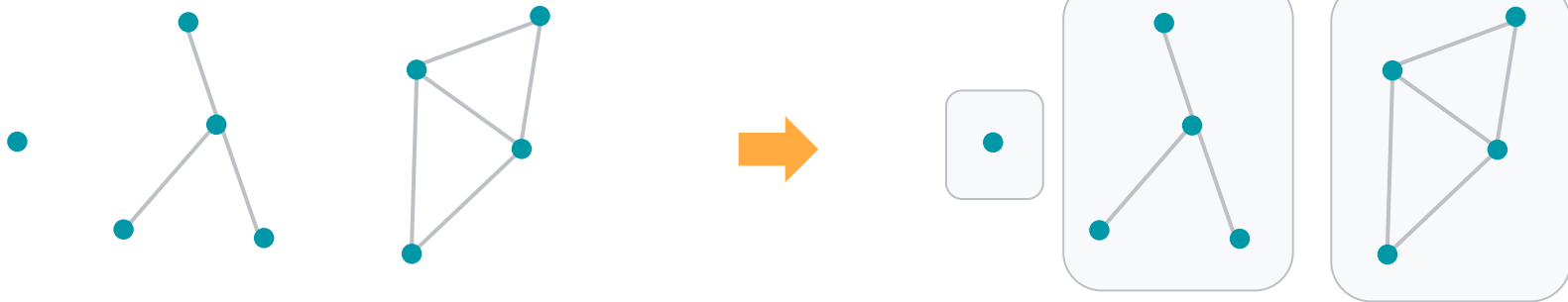


# Example problem: connected components

Two vertices in the same connected component

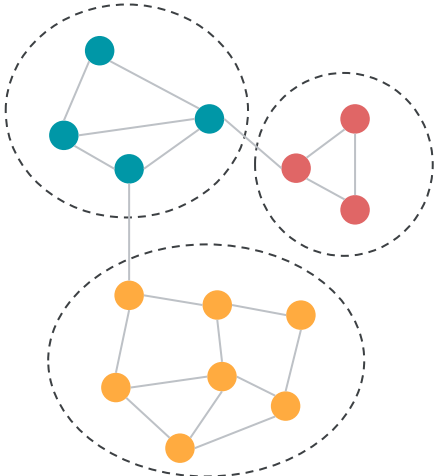


There is a path connecting them

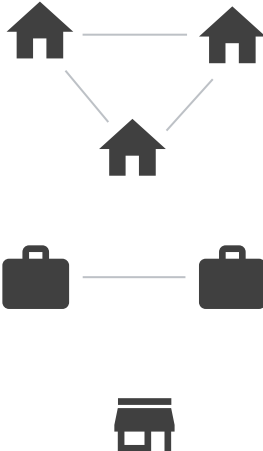


# Connected components - applications

(Hierarchical) Clustering



Deduplication



Building block for other algorithms



## Connected components is the most popular graph computation

The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. Sahu, Mhedhbi, Salihoglu, Lin, Özsu, VLDB'18

# Connected components - algorithms

## Efficient implementations

### HashToMin

[RastogiMCS, ICDE'13]

### TwoPhase

[KiverisLMRV, SOCC'14]

### Cracker

[LulliRCDL, ISCC'15]

### LocalContract

[Ł.MW, arxiv]

## Theory algorithms

### $O(\log n)$

[KarloffSV, SODA'10]

### $O(\log \log n)$ in random graphs

[AssadiSW, PODC'19]

### $O(\log_{m/n} \log n \log D)$

[AndoniSSWZ, FOCS'18]

### $O(\log_{m/n} \log n + \log D)$

[BehnezhadDEŁ.M, FOCS'19]

D = graph diameter

# Connected components algorithm

*Connected components at scale via local contractions. Łącki, Mirrokni, Włodarczyk, arxiv*

```
while G has any edges
  for each vertex v
    label(v) := Uniform[0, 1]
    best(v) := neighbor w of v minimizing label(w)
  group nodes by best(v) and merge together
```

- In each iteration, the number of vertices shrinks by a constant factor
  - Algorithm requires  $O(\log n)$  MPC rounds
- Similar algorithm takes  $O(\log \log n)$  rounds in random graphs

## Connected components - relative running times

Graph (#edges)	Orkut (117M)	Friendster (1.8B)	Clueweb (37.3B)	videos (626B)	webpages (6.5T)
<b>New</b>	1.0	1.0	1.0	1.03	1.0
<b>Cracker</b>	1.38	1.16	2.65	1.0	~3.0
<b>TwoPhase</b>	5.77	1.73	1.77		
<b>HashToMin</b>	5.84	20.27			

# Connected components - theory

In the MPC model with  $O(n^\epsilon)$  space per machine:

## Theorem

*Near-Optimal Massively Parallel Graph Connectivity.* Behnezhad, Dhulipala, Esfandiari, Łącki, Mirrokni, FOCS'19

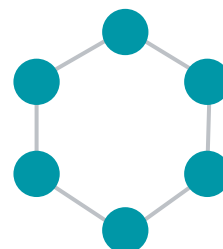
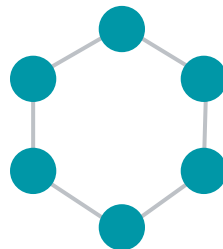
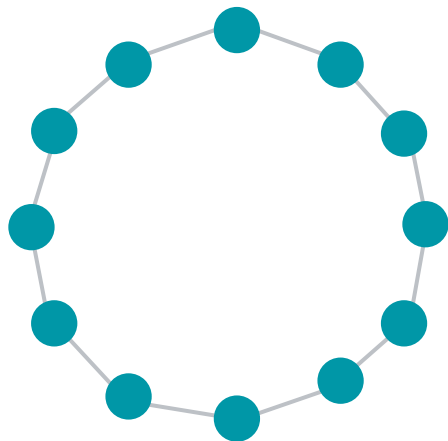
Connected components can be found in  $O(\log_{m/n} \log n + \log D)$  rounds, where  $D$  is the diameter of the input graph.

## Conjecture

Finding connected components requires  $\Omega(\log n)$  rounds.



# Connected components - hard instance

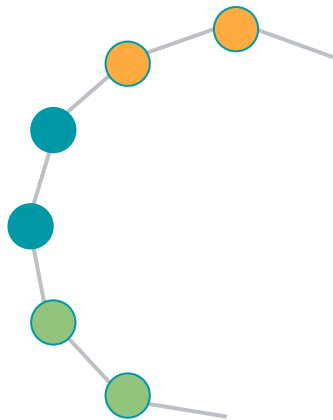


Distinguish between a cycle on  $2n$  nodes and two cycles on  $n$  nodes

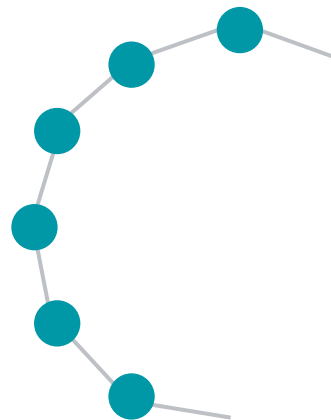
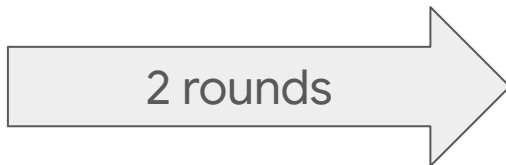
**Conjecture:** requires  $\Omega(\log n)$  rounds in MPC model

# Connected components - main challenge

Intuitive goal: aggregate consecutive nodes on one machine



Each machine has  $k$  nodes



Each machine has  $3k$  nodes

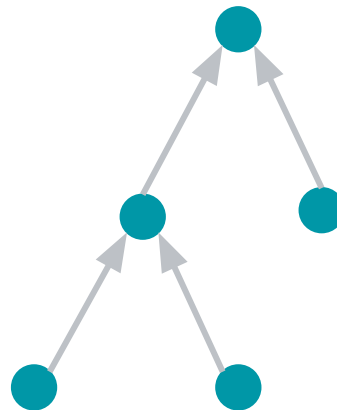
**Workaround:** give machines **random read access** to the graph

# Random read access

- Store the graph in a **distributed hash table**
- Allow machines to read the graph **adaptively** within a round

## Example:

- Input: collection of rooted trees
- A node can find the root of its tree in a single round
- Used in affinity clustering



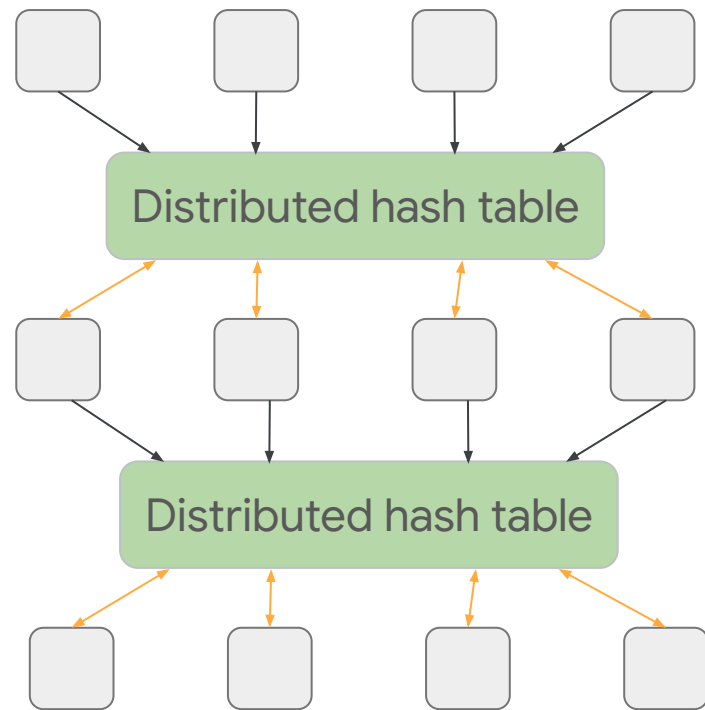
# Adaptive MPC model

Massively Parallel Computation via Remote Memory Access. Behnezhad, Dhulipala, Esfandiari, Łącki, Schudy, Mirrokni, SPAA'19

Modification of MPC. Differences:

- All messages saved to a distributed hash table (DHT)
- In the following round each machine can **adaptively read**  $O(S)$  values from the DHT

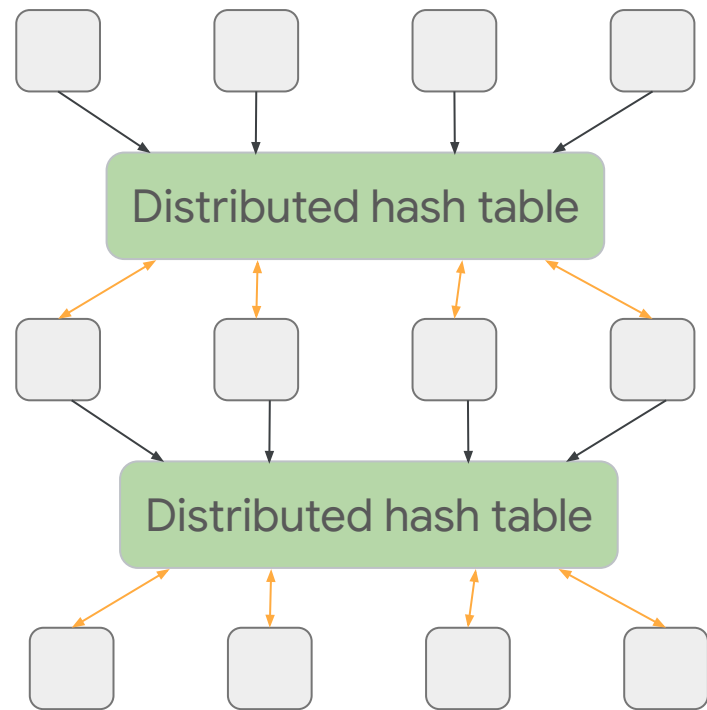
Same bounds on communication



# Adaptive MPC - realism

Is the model realistic?

- Remote read latency?
  - Use hardware support (RDMA)
  - 1-3  $\mu$ s (~20x slower than RAM)
- Fault tolerance?
  - Relies on a fault-tolerant distributed hash table



# Adaptive MPC - theory results

Problem	MPC	AMPC
Maximal Independent Set	$\tilde{O}(\sqrt{\log n})$	$O(1)$
Connectivity	$O(\log D)$	$O(1)$
Minimum Spanning Tree (MST)	$O(\log n)$	$O(1)$
Approximate matching	$\tilde{O}(\sqrt{\log n})$	$O(1)$

## Assumptions

- $n^\epsilon$  space per machine
- Graph has  $n^{1+\epsilon}$  edges
- $D$  = graph diameter

# Adaptive MPC - empirical results

*Parallel Graph Algorithms in Constant Adaptive Rounds: Theory meets Practice.* Behnezhad, Dhulipala, Esfandiari, Łącki, Mirrokni, Schudy, VLDB'20.

Results on 5 graphs of up to 225B edges

Problem	MPC rounds	AMPC rounds	AMPC Speedup
Minimum spanning forest	33-84	5	2.6x - 7.2x
Maximal independent set	8-14	1	2.3x - 3x
Maximal matching	8-16	1	1.16x - 1.7x

# ASYMP: Fault-tolerant Mining of Massive Graphs Asynchronously

*ASYMP: Fault-tolerant Mining of Massive Graphs*. Fleury, Lattanzi, Mirrokni, Perozzi, arxiv.

- Shortest paths - hard to solve in (A)MPC model
- Solution: ASYMP = new framework for message passing algorithms



Asynchronous message passing



Light fault tolerance  
Asynchronous checkpoints



More efficient CPU  
utilization



Impressive performance  
(shortest paths)



How to use for other  
problems?



# Conclusion

- Many interesting problems on the boundary of algorithms & systems
- Distributed systems allow handling graphs with trillions of edges
- Read-only access to the input allows significant speedups
  
- What about large-but-not-huge graphs?
  - Big overhead / resource usage of a distributed system
  - **Next talk:** working with graphs of up to **10B edges** on a single machine





# Multi-core parallel clustering

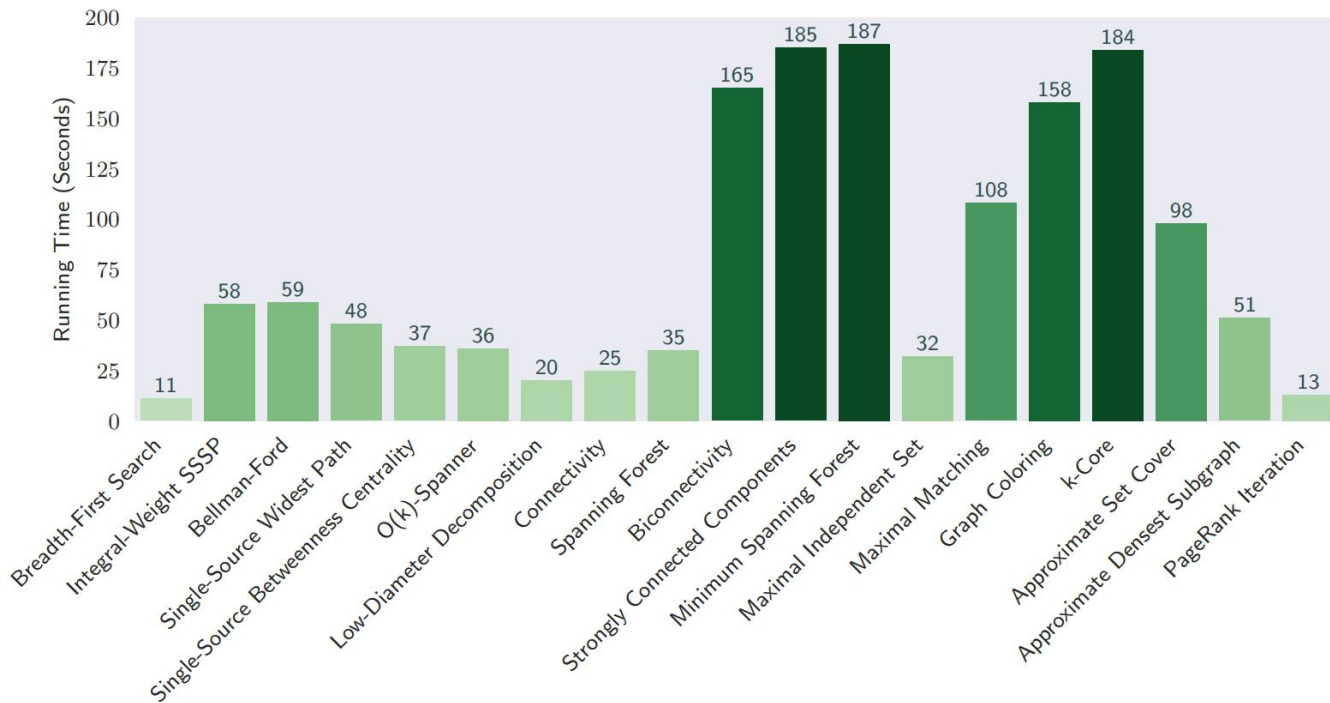
Jakub Łącki

Collaborators: David Applegate, Laxman Dhulipala, David Eisenstat, Heinrich Jiang, Vahab Mirrokni, Jessica Shi

# Goal

***Cluster billion-edge graphs in few minutes on a single machine***

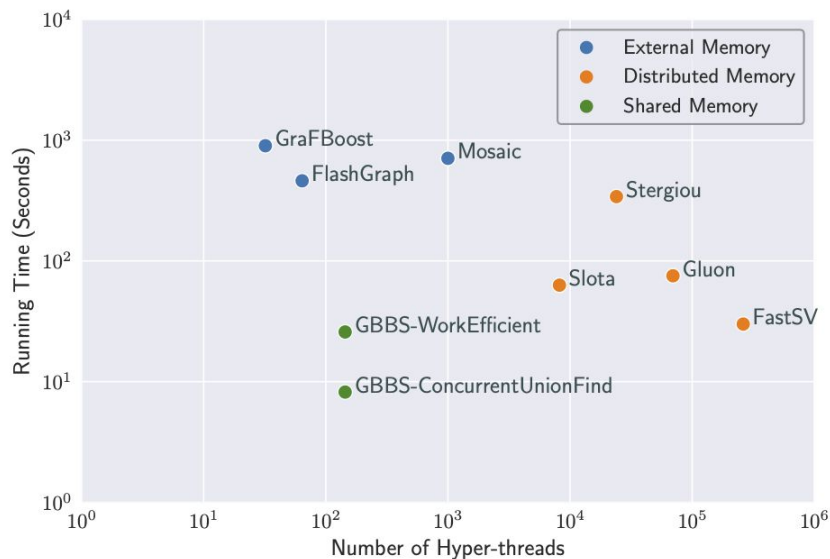
# Multi-core parallel graph algorithms



**Basic graph problems on a 225B-edge graph can be solved in <3 minutes**

*Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable.* Dhulipala, Blelloch, and Shun, SPAA'18

# Multi-core algorithms are fast and cost-effective



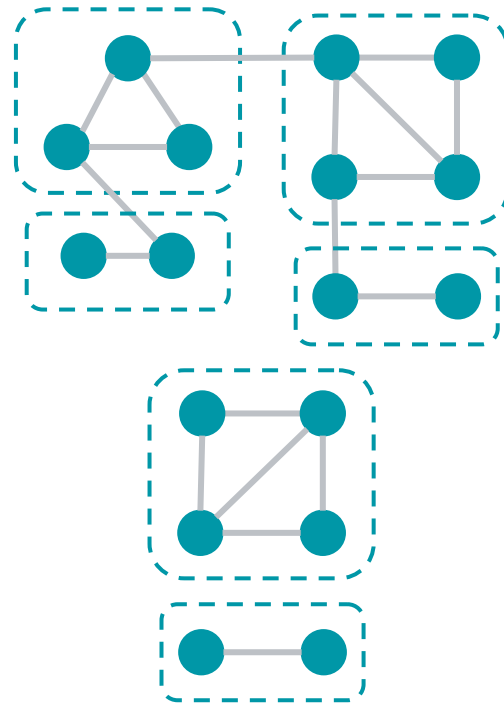
FastSV [Zhang et al. 2020] is run on a Cray XC40 supercomputer.

**Efficient multi-core algorithms can outperform high-end supercomputers at low cost**

*ConnectIt: A Framework for Static and Incremental Parallel Graph Connectivity Algorithms.* Dhulipala, Hong, and Shun, VLDB'21

# Multi-core parallel clustering

- Clustering is a complex problem
  - High running time complexity
  - Input is a **weighted** graph
- We develop parallel clustering algorithms for billion-edge graphs
  - Affinity clustering
  - Correlation clustering
  - Modularity clustering



# GBBS framework

Laxman Dhulipala (MIT), Jessica Shi (MIT), Tom Tseng (MIT), Guy Blelloch (CMU), Julian Shun (MIT)

C++ library for implementing parallel graph algorithms



Graph representation



Parallel graph primitives  
(low level)



Parallel scheduler

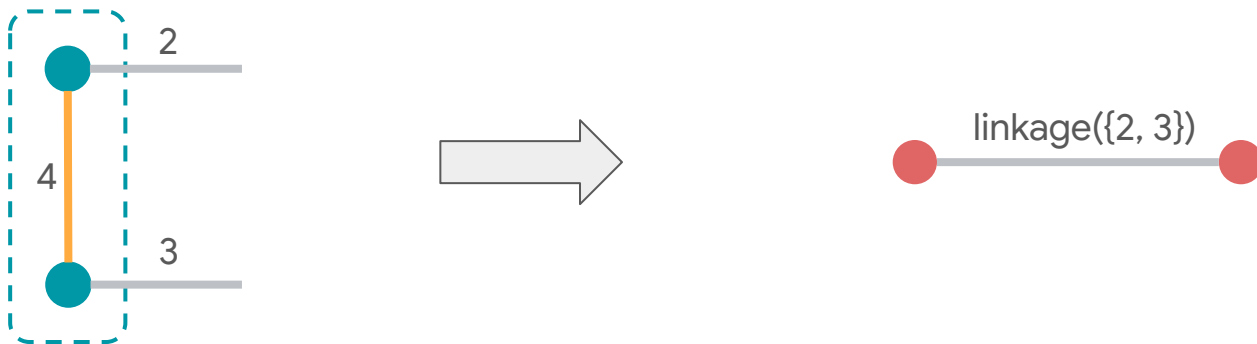


Work-efficient implementations  
polylog depth

<https://github.com/ParAlg/gbbs>

# Affinity clustering

- Clustering of weighted graphs
- Recap: each node in the same cluster as its most similar neighbor
- Parallel algorithm repeats the following steps:
  - a. **Mark** highest-weight incident edge of each node
  - b. Find **connected components** of the marked edges
  - c. Contract each cluster to a single **node**





# Parallel affinity clustering - quality

Graph	Adjusted RAND index			
	Affinity	HAC	DBScan	Modularity
banknote	<b>0.8467</b>	0.4637	0.7903	0.3219
glass	<b>0.8525</b>	0.7852	0.7125	0.5800
images	0.5224	0.5551	0.5053	<b>0.5731</b>
iris	<b>0.8858</b>	0.7455	<b>0.8858</b>	0.5526
letters	<b>0.2890</b>	0.2670	0.1914	0.2480
pageblocks	<b>0.2001</b>	0.0722	0.1481	0.0528
phoneme	0.7324	<b>0.7908</b>	0.7732	0.7678
seeds	<b>0.7329</b>	0.5877	0.7171	0.7066

Datasets from UCI, Kaggle and *The elements of statistical learning*, Friedman, Hastie, and Tibshirani.

# Parallel affinity clustering - performance

#edges	Serial	Parallel	Speedup
117M	110s	26s	4.2x
922M	450s	153s	2.9x
1.8B	5593s	640s	8.7x

← 200 GB RAM

Clustering times (excl. I/O time)

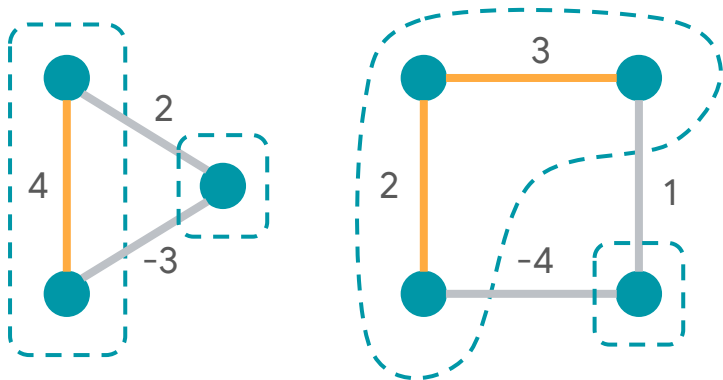
All times from a shared machine in a production cell

# Correlation clustering

Input: graph with positive / negative edge weights

- Positive edge ~ endpoints should be in the same cluster
- Negative edge ~ endpoints should be in different clusters

Objective: maximize sum of edge weights within clusters



Objective = 9

# Correlation clustering algorithm

Start with each node in its cluster

1. Find each node's *best move* (to a cluster based on objective)

2. Find each cluster's *best move* (to merge with a cluster)

Iterate

# Parallel correlation clustering algorithm

Start with each node in its cluster

## 1. Find each node's *best move* (to a cluster based on objective)

Compute best moves in parallel + aggregate new clusters in parallel

## 2. Find each cluster's *best move* (to merge with a cluster)

Compute best moves in parallel + aggregate new clusters in parallel

Iterate

# Parallel correlation clustering - empirical results

Running time (seconds)

#edges	serial	parallel	speedup
380M	413	80	5.1x
950M	1184	301	3.9x

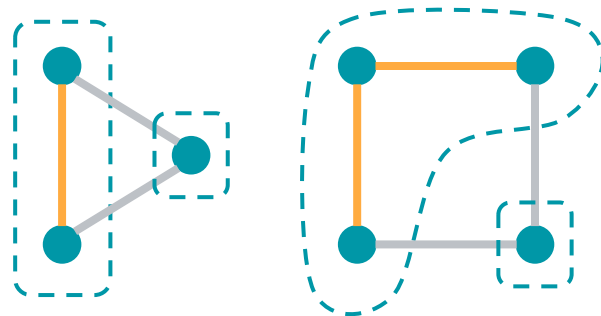
Objective value (scaled by  $10^7$ )

#edges	serial	parallel	difference
380M	2.4464	2.3081	94.3%
950M	4.7052	4.6943	99.7%

# Summary

## Parallel in-memory algorithms

- Can be both **faster** and **cheaper** than distributed algorithms
- Can cluster XB-edge graphs in few minutes
- Speed up clustering **3-9x** compared to serial baselines



Download:

[github.com/google-research/google-research/tree/master/parallel\\_clustering/](https://github.com/google-research/google-research/tree/master/parallel_clustering/)

# Mining and Learning with Graphs *at Scale*

<https://gm-neurips-2020.github.io/>

