End-toEnd In-memory Graph Analytics

Jure Leskovec (@jure)

Including joint work with Rok Sosic, Deepak Narayanan, Yonathan Perez, et al.

Background & Motivation

My research at Stanford:

- Mining large social and information networks
- We work with data from Facebook,Twitter, LinkedIn, Wikipedia, StackOverflow

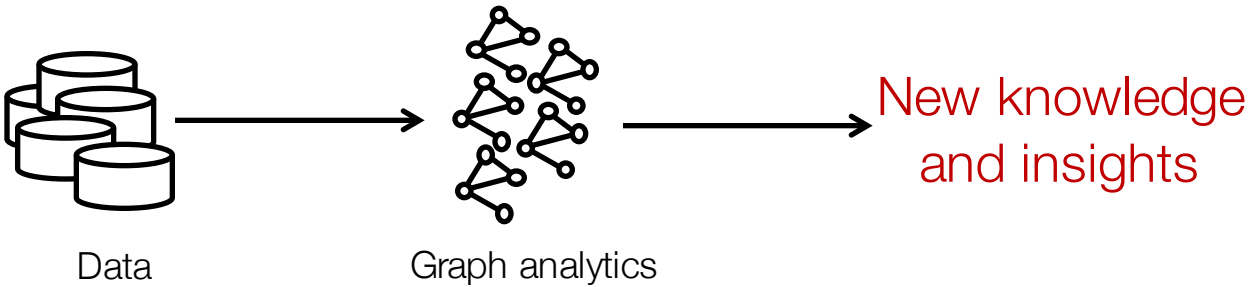
Much research on graph processing systems but we don't find it that useful…

Why is that? What tools do we use?
What do we see are some big challenges?

Some Observations

- We do not develop experimental systems to compete on benchmarks
  - BFS, PageRank, Triangle counting, etc.
- Our work is
  - Knowledge discovery: Working on new problems using novel datasets to extract new knowledge
  - And as a side effect developing (graph) algorithms and software systems

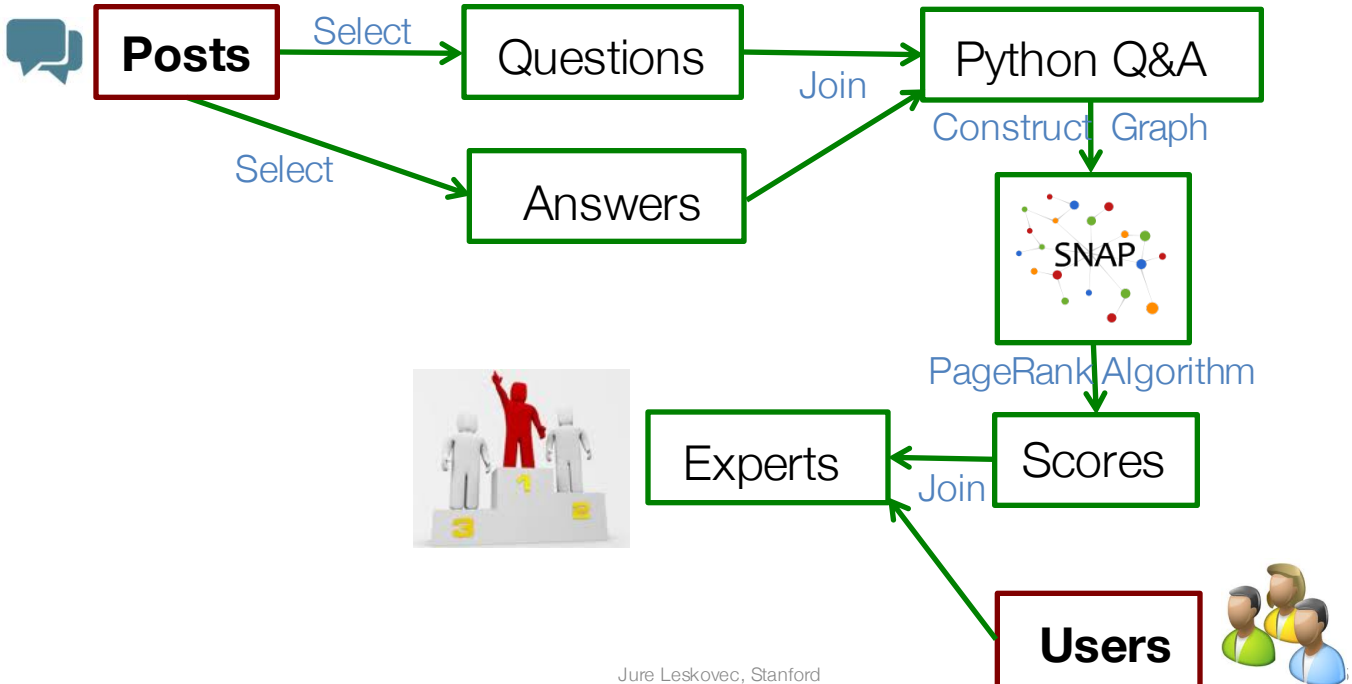
End-to-End Graph Analytics



Need end-to-end graph analytics system that is flexible, scalable, and allows for easy implementation of new algorithms.

Typical Workload

- Finding experts on StackOverflow:



Observation

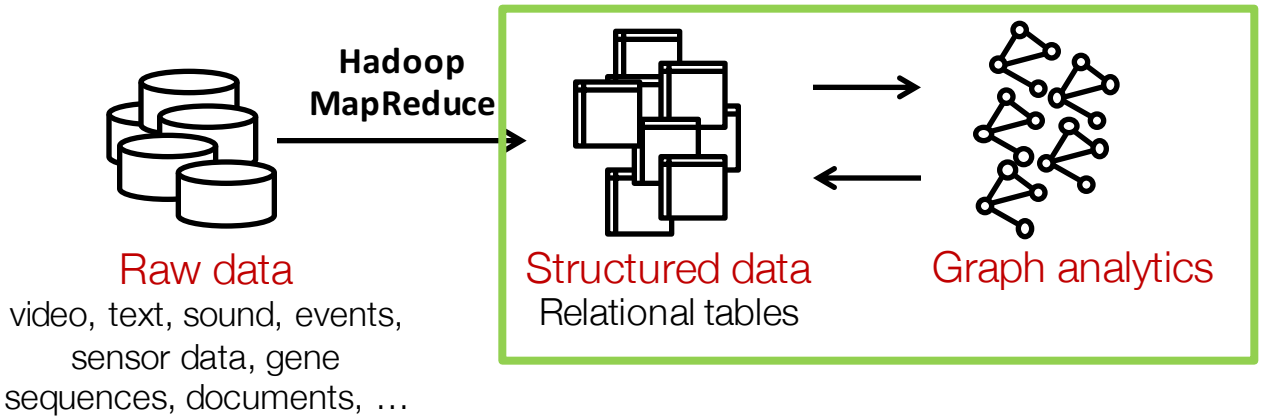
Graphs are never given!

Graphs have to be constructed from input data! (graph constructions is a part of knowledge discovery process)

Examples:

- Facebook graphs: Friend, Communication, Poke, Co-tag, Co-location, Co-event
- Cellphone/Email graphs: How many calls?
- Biology: P2P, Gene interaction networks

Graph Analytics Workflow



- Input: Structured data
- Output: Results of network analyses
  - Node, edge, network properties
  - Expanded relational tables
  - Networks

Plan for the Talk: Three Topics

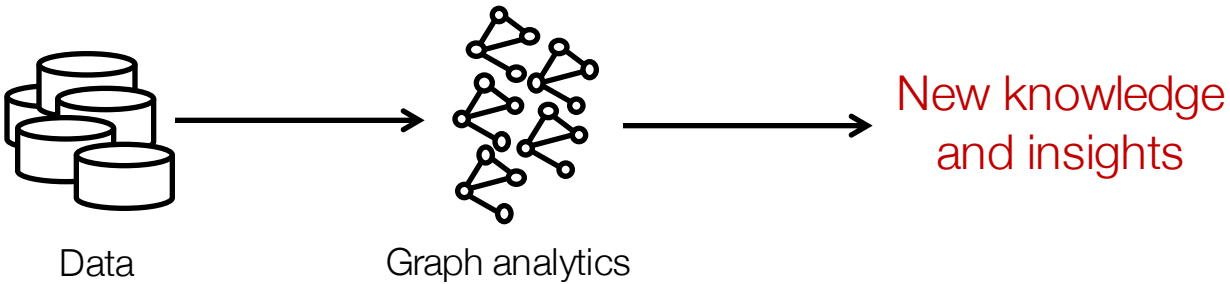
- SNAP: an in-memory system for end-to-end graph analytics
  - Constructing graphs from data
- Multimodal networks
  - Representing richer types of graphs
- New graph algorithms
  - Higher-order network partitioning
  - Feature learning in networks

SNAP
Stanford Network Analysis Platform

SNAP: A General Purpose Network Analysis and Graph Mining Library.
R. Sosic, J. Leskovec. ACM TIST 2016.

RINGO: Interactive Graph Analytics on Big-Memory Machines Y. Perez, R. Sosic, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, J. Leskovec. SIGMOD 2015.

End-to-End Graph Analytics



- **S**tanford **N**etwork **A**nalysis **P**latform (SNAP)
  General-purpose, high-performance system for analysis and manipulation of networks
  - C++, Python (BSD, open source)
  - http://snap.stanford.edu
- Scales to networks with hundreds of millions of nodes and billions of edges

Desiderata for Graph Analytics

- Easy to use front-end
  - Common high-level programming language
- Fast execution times
  - Interactive use (as opposed to batch use)
- Ability to process large graphs
  - Billions of edges
- Support for several data representations
  - Transformations between tables and graphs
- Large number of graph algorithms
  - Straightforward to use
- Workflow management and reproducibility
  - Provenance

Data Sizes in Network Analytics

Number of Edges	Number of Graphs
<0.1M	16
0.1M – 1M	25
1M – 10M	17
10M – 100M	7
100M – 1B	5
> 1B	1

- Networks in Stanford Large Network Collection
  - http://snap.stanford.edu
  - Common benchmark Twitter2010 graph has 1.5B edges, requires 13.2GB RAM in SNAP

Network of all Published research

Entity	#Items	Size
Papers	122.7M	32.4GB
Authors	123.1M	3.1GB
References	757.5M	14.4GB
Affiliations	325.4M	15.3GB
Keywords	176.8M	5.9GB
Total	**1.9B**	**104.1GB**

- Microsoft Academic Graph

All Biomedical Research

Dataset	#Items	Raw Size
DisGeNet	30K	10MB
STRING	10M	1TB
OMIM	25K	100MB
CTD	55K	1.2GB
HPRD	30K	30MB
BioGRID	64K	100MB
DrugBank	7K	60MB
Disease Ontology	10K	5MB
Protein Ontology	200K	130MB
Mesh Hierarchy	30K	40MB
PubChem	90M	1GB
DGIdb	5K	30MB
Gene Ontology	45K	10MB
MSigDB	14K	70MB
Reactome	20K	100MB
GEO	1.7M	80GB
ICGC (66 cancer projects)	40M	1TB
GTEx	50M	100GB

Total: 250M entities, 2.2TB raw data

Availability of Hardware

Could all these datasets fit into RAM of a single machine?

Single machine prices:

- Server 1TB RAM, 80 cores, $25K
- Server 6TB RAM, 144 cores, $200K
- Server 12TB RAM, 288 cores, $400K

In my group we have 1TB RAM machines since 2012 and just got a 12TB RAM machine

Dataset vs. RAM Sizes

- KDNuggets survey since 2006 surveys: "What is the largest dataset you analyzed/mined?"
- Big RAM is eating big data:
  - Yearly increase of dataset sizes: 20%
  - Yearly increase of RAM sizes: 50%

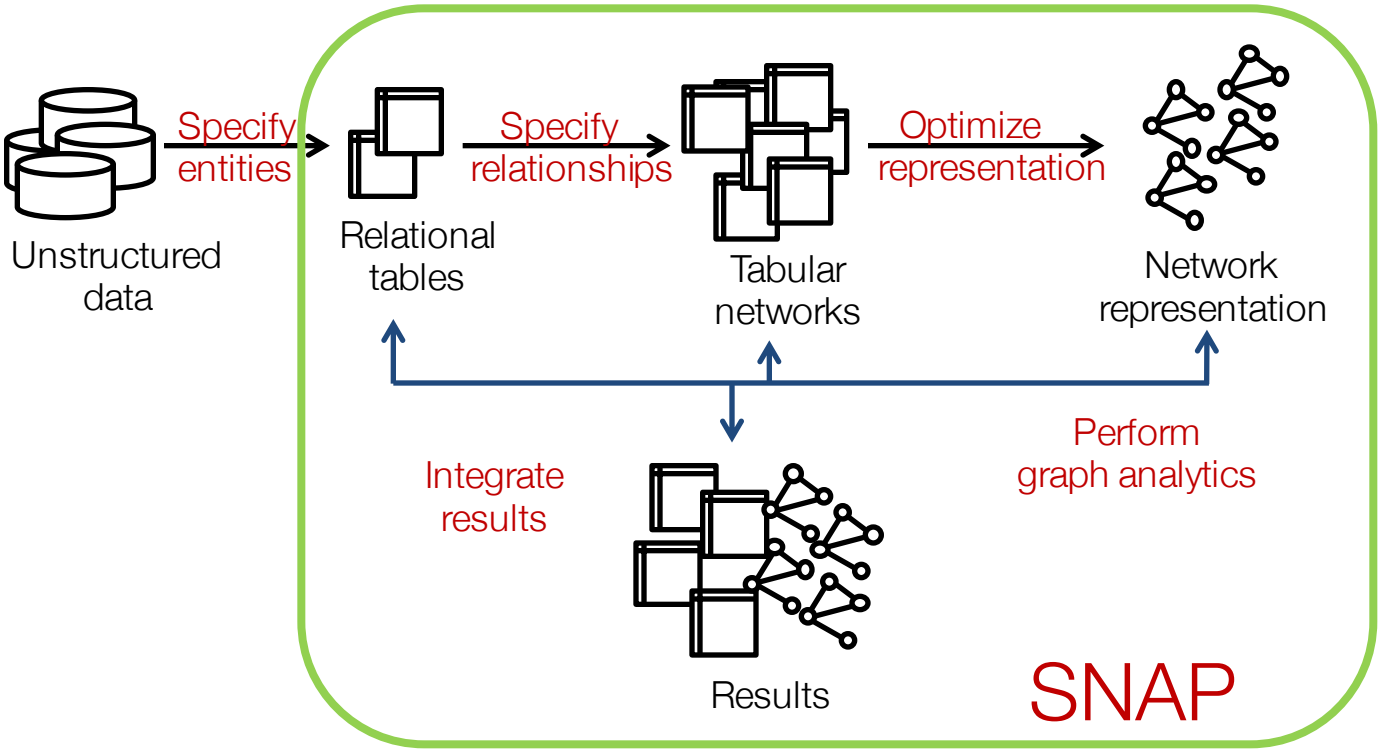
Bottom line: Want to do graph analytics? **Get a BIG machine!**

Trade-offs

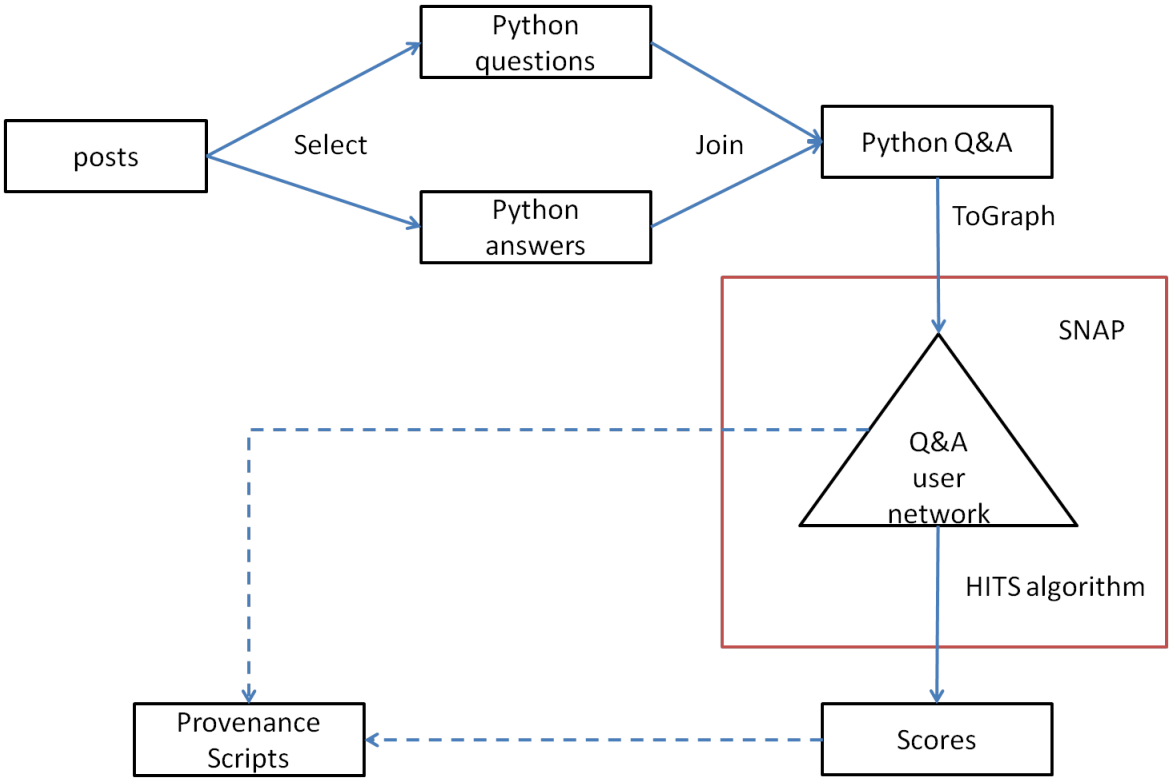
Option 1	Option 2
Standard SQL database	Custom representations
Separate systems for tables and graphs	Integrated system for tables and graphs
Single representation for tables and graphs	Separate table and graph representations
Distributed system	Single machine system
Disk-based structures	In-memory structures

SNAP

Graph Analytics: SNAP



Experts on StackOverflow



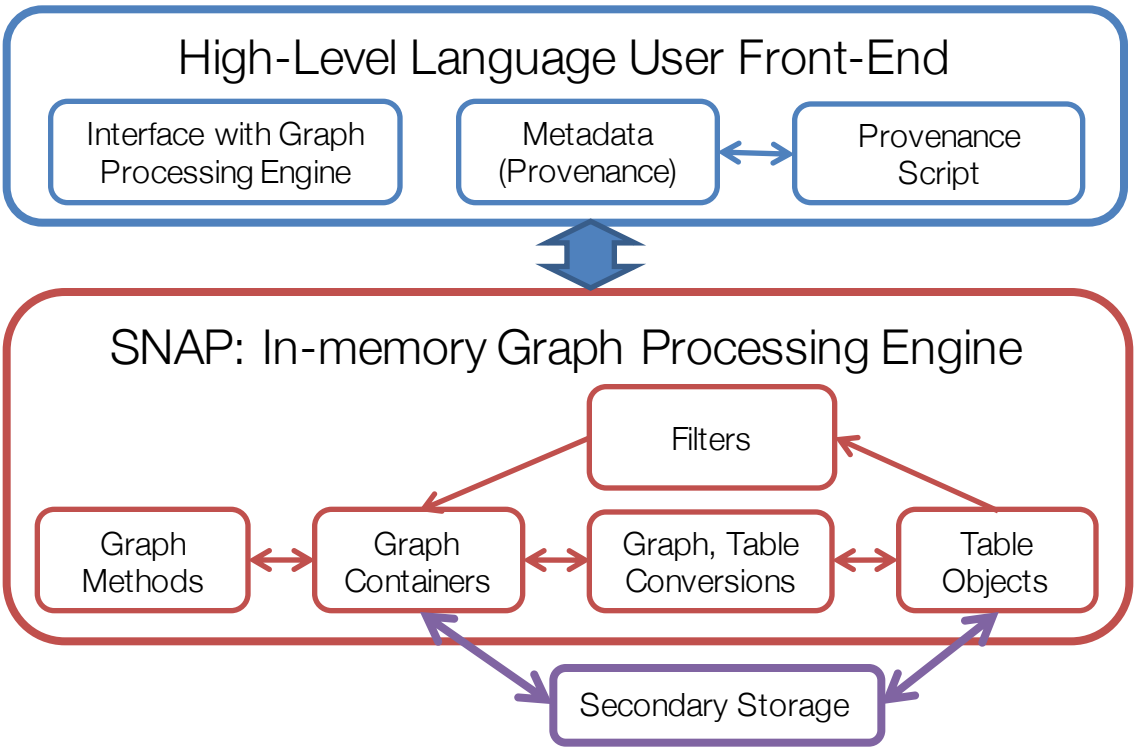
Graph Construction in SNAP

- SNAP (Python) code for executing finding the StackOverflow example

```
P = ringo.LoadTable(schema,'posts.tsv')
JP = ringo.Select(P,'Tag=Java')
Q = ringo.Select(JP,'Type=question')
A = ringo.Select(JP,'Type=answer')
QA = ringoJoin(Q,A,'AnswerId','PostId')
G = ringo.ToGraph(QA,'UserId.1','UserId.2')
PR = ringo.GetPageRank(G)
S = ringo.ToTable(PR,'UserId','Score')
ringo.Save(S,'scores.bin')
```

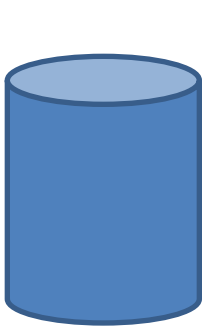
RINGO: Interactive Graph Analytics on Big-Memory Machines Y. Perez, R. Sosic, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, J. Leskovec. SIGMOD 2015.

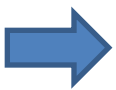
SNAP Overview



Graph Construction

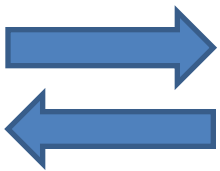
Input data must be manipulated and transformed into graphs

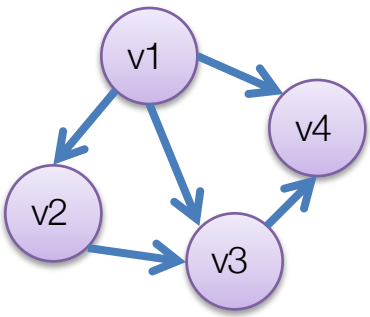




Src	Dst	...
v1	v2	...
v2	v3	...
v3	v4	...
v1	v3	...
v1	v4	...

Table data structure





Graph data structure

Creating a Graph in SNAP

Four ways to create a graph:

Nodes connected based on

(1) Pairwise node similarity

(2) Temporal order of nodes

(3) Grouping and aggregation of nodes

(4) The data already contains edges as source and destination pairs

Creating Graphs in SNAP (1)

Similarity-based: In a forum, connect users that post to similar topics

- Distance metrics
  - Euclidean, Haversine, Jaccard distance
- Connect similar nodes
  - SimJoin, connect if data points are closer than some threshold
  - How to get around quadratic complexity
    - Locality Sensitive Hashing

Creating Graphs in SNAP (2)

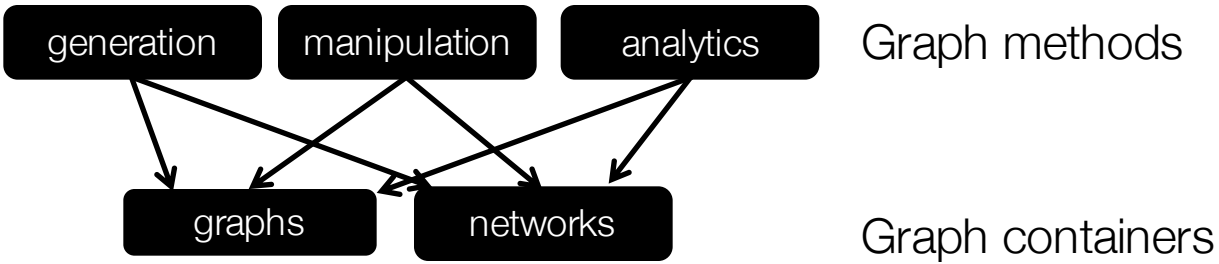
Sequence-based: In a Web log, connect pages in an order clicked by the users (click-trail)

- Connect a node with its *K* successors
  - Events selected per user, ordered by timestamps
  - NextK, connect *K* successors

Creating Graphs in SNAP (3)

- Aggregation: Measure the activity level of different user groups
  - Edge creation
    - Partition users to groups
    - Identify interactions within each group
    - Compute a score for each group based on interactions
  - Treat groups as super-nodes in a graph

Graphs and Methods

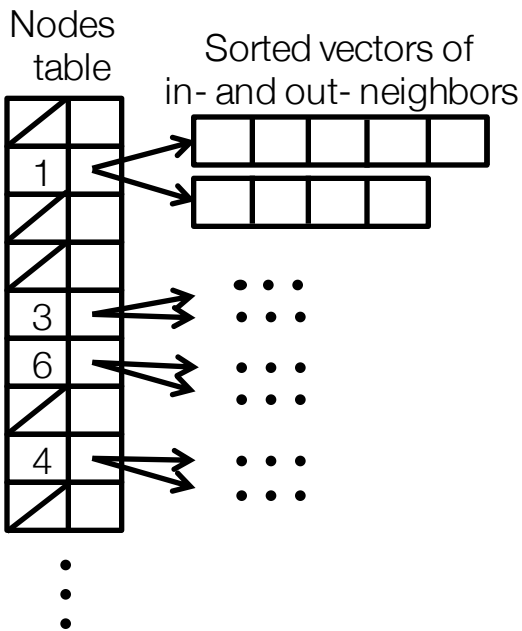


- SNAP supports several graph types
  - Directed, Undirected, Multigraph
- >200 graph algorithms
- Any algorithm works on any container

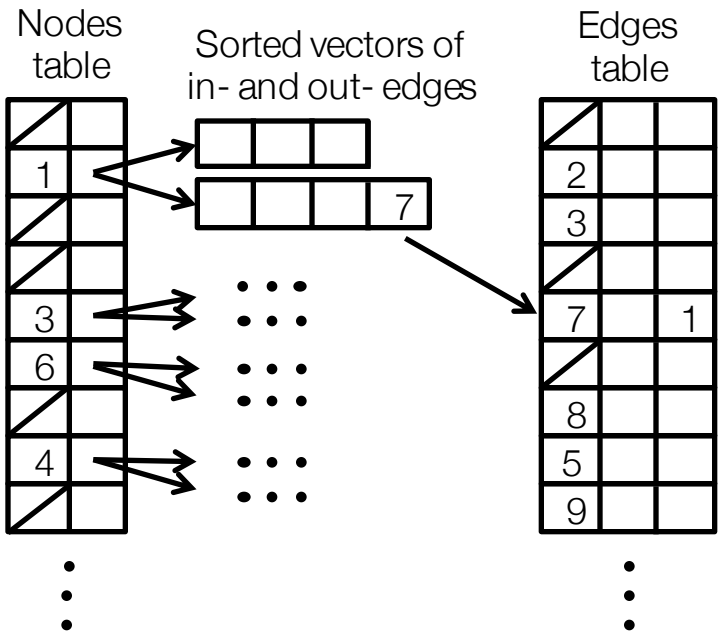
SNAP Implementation

- High-level front end
  - Python module
  - Uses SWIG for C++ interface
- High-performance graph engine
  - C++ based on SNAP
- Multi-core support
  - OpenMP to parallelize loops
  - Fast, concurrent hash table, vector operations

Graphs in SNAP



Directed graphs in SNAP



Directed multigraphs in SNAP

Experiments: Datasets

Dataset	LiveJournal	Twitter2010
Nodes	4.8M	42M
Edges	69M	1.5B
Text Size (disk)	1.1GB	26.2GB
Graph Size (RAM)	0.7GB	13.2GB
Table Size (RAM)	1.1GB	23.5GB

Benchmarks, One Computer

Algorithm Graph	PageRank LiveJournal	PageRank Twitter2010	Triangles LiveJournal	Triangles Twitter2010
Giraph	45.6s	439.3s	N/A	N/A
GraphX	56.0s	-	67.6s	-
GraphChi	54.0s	595.3s	66.5s	-
PowerGraph	27.5s	251.7s	5.4s	706.8s
SNAP	**2.6s**	**72.0s**	**13.7s**	**284.1s**

Hardware: 4x Intel CPU, 64 cores, 1TB RAM, $35K

Published Benchmarks

System	Hosts	CPUs host	Host Configuration	Time
GraphChi	1	4	8x core AMD, 64GB RAM	158s
TurboGraph	1	1	6x core Intel, 12GB RAM	30s
Spark	50	2		97s
GraphX	16	1	8X core Intel, 68GB RAM	15s
PowerGraph	64	2	8x hyper Intel, 23GB RAM	3.6s
SNAP	**1**	**4**	**20x hyper Intel, 1TB RAM**	**6.0s**

Twitter2010, one iteration of PageRank

SNAP: Sequential Algorithms

Algorithm	Runtime
3-core	31.0s
Single source shortest path	7.4s
Strongly connected components	18.0s

LiveJournal, 1 core

SNAP: Sequential Algorithms

Algorithm	Time (s)	Implementation
In-degree	14	1 core
Out-degree	8	1 core
PageRank	115	64 cores
Triangles	107	64 cores
WCC	1,716	1 core
K-core	2,325	1 core

- Benchmarks on the citation graph: nodes 50M, edges 757M

SNAP: Tables and Graphs

Dataset	LiveJournal	Twitter2010
Table to graph	8.5s 13.0 MEdges/s	81.0s 18.0 MEdges/s
Graph to table	1.5s 46.0 MEdges/s	29.2s 50.4 MEdges/s

Hardware: 4x Intel CPU, 80 cores, 1TB RAM, $35K

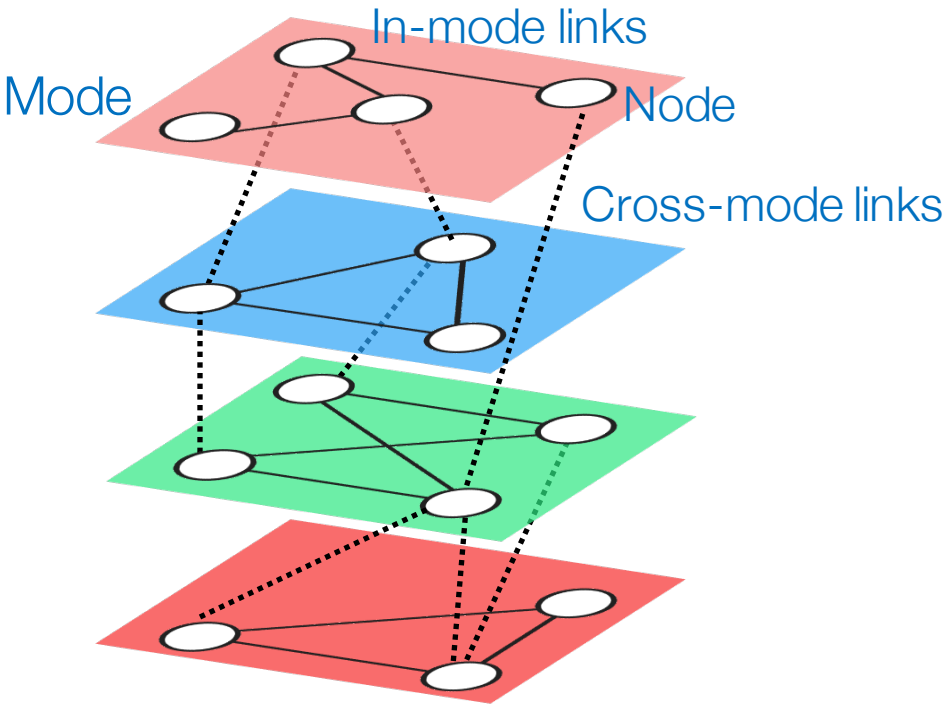
SNAP: Table Operations

Dataset	LiveJournal	Twitter2010
Select	<0.1s 575.0 MRows/s	1.6s 917.7 MRows/s
Join	0.6s 109.5 MRows/s	4.2s 348.8 MRows/s
Load graph	5.2s	76.6s
Save graph	3.5s	69.0s

Hardware: 4x Intel CPU, 80 cores, 1TB RAM, $35K

Multimodal Networks:
A network of networks

Multimodal Networks

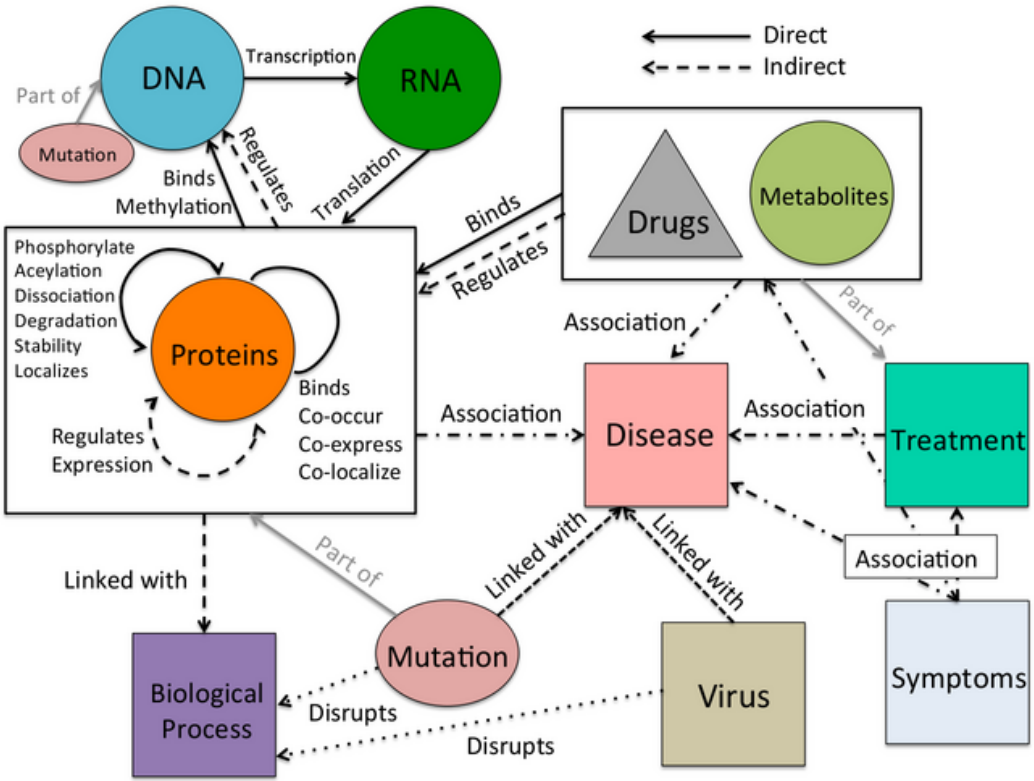


Network of networks

Why multimodal networks?

- Can encode additional semantic structure than a “simple” graph
- Many naturally occurring graphs are multimodal networks
  - Gene-drug-disease networks
  - Social networks,
  - Academic citation graphs

Multimodal Network Example



Challenges

Multimoral network requirements:

- Fast processing
  - Efficient traversal of nodes and edges
- Dynamic structure
  - Quickly add/remove nodes and edges
    - Create subgraphs, dynamic graphs, …
- Tradeoff
  - High performance, fixed structure
  - Highly flexible structure, low performance

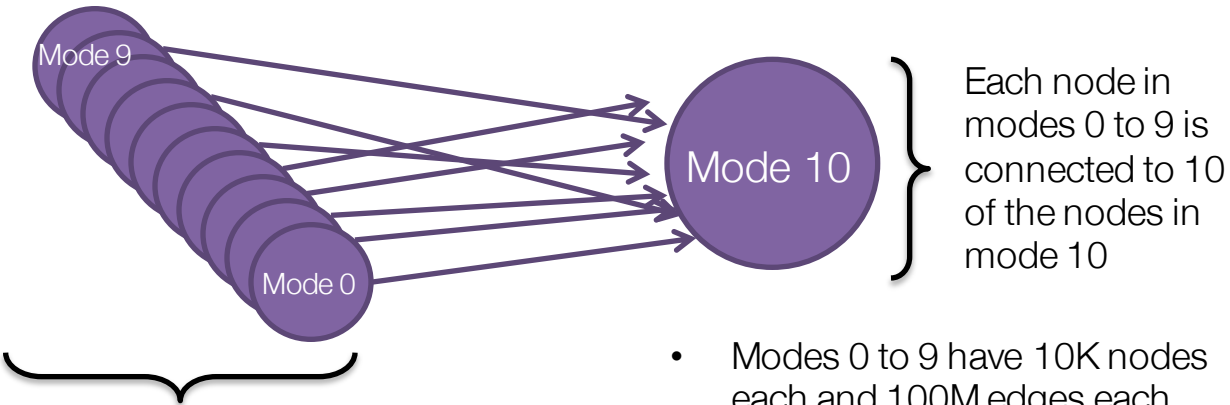
Piggyback on a Graph

Why can't we just piggyback extra information onto a regular graph?

- Want to ensure that per-mode information is easily accessible as a unit
- Want more fine-grained control as to where certain vertex and edge information resides
- Want indexes that allow for easy random access

Piggyback mode information

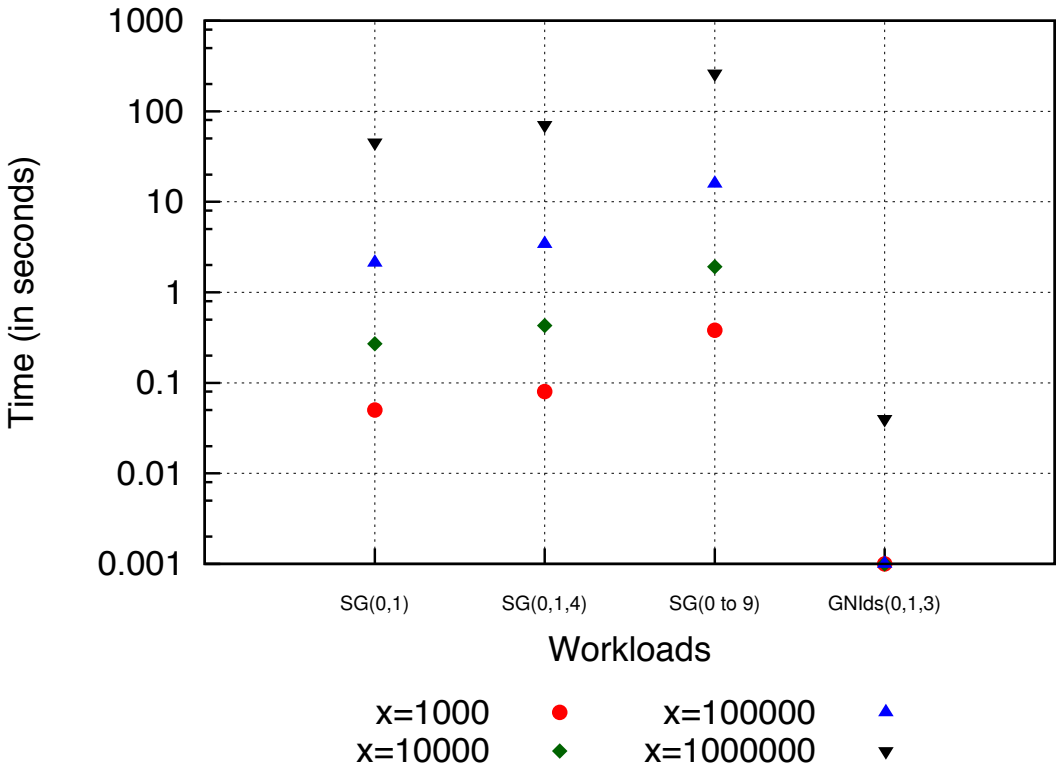
Benchmark multimodal graph:



- Modes 0 to 9 have 10K nodes each and 100M edges each
- Mode 10 has *X* nodes
- Each node in modes 0 to 9 is connected to all nodes in mode 10

X controls randomness of redundant edges (while the output size is fixed)

Experiment



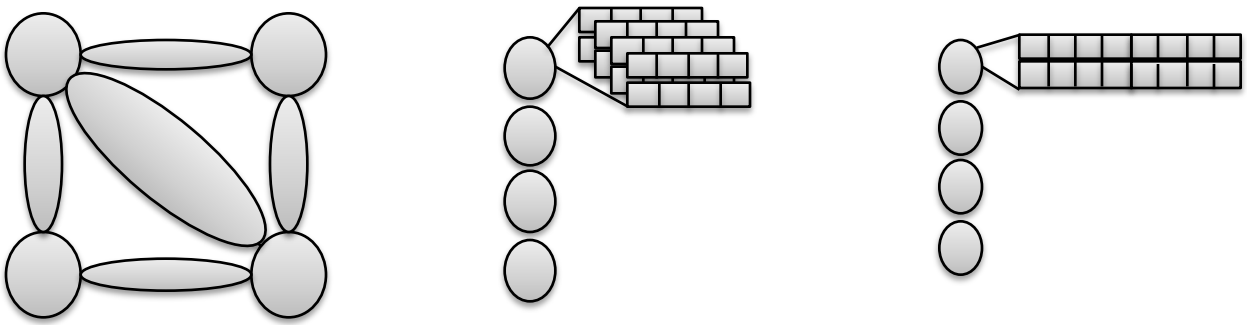
For X=1M, graph has 10.1B edges

Extract subgraph on given modes

How to be faster?

- Remember: Everything is in memory so don't need to worry about disk
- Desirable properties:
  - Stay in cache as much as possible as memory accesses are expensive in comparison
    - I.e., we want good memory locality
  - Cheap index lookups that allow us to avoid having to look at the entire data structure

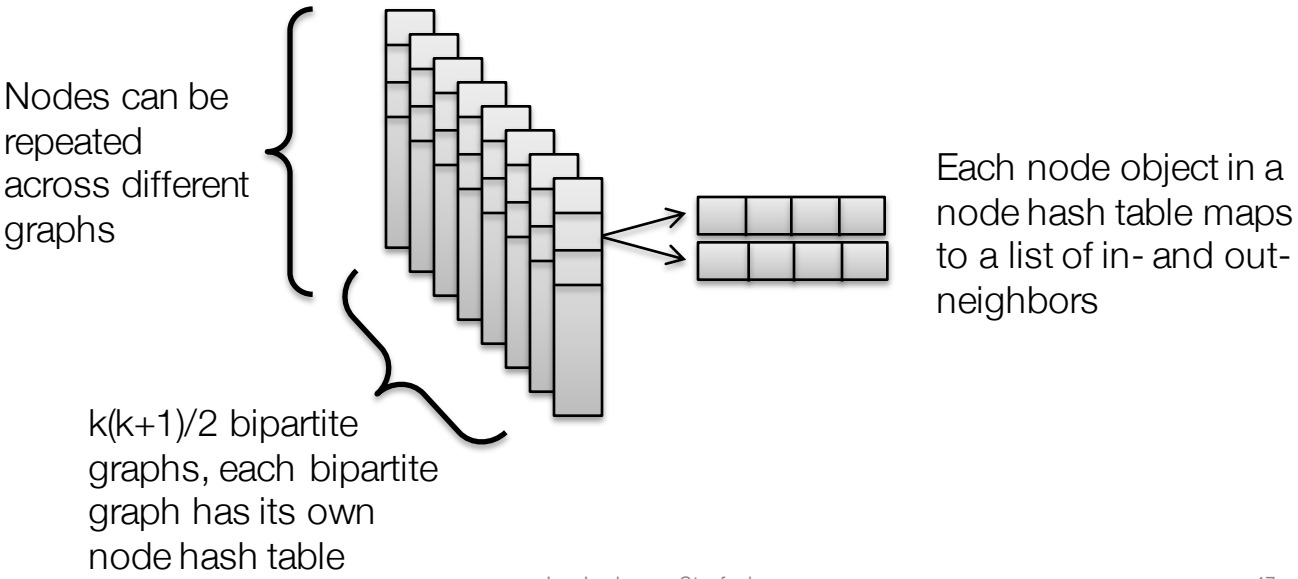
Multimodal Networks



- Idea 1: Represent the multimodal graph as a collection of bipartite graphs
- Idea 2: Consolidate node hash tables
- Idea 3: Consolidate adjacency lists

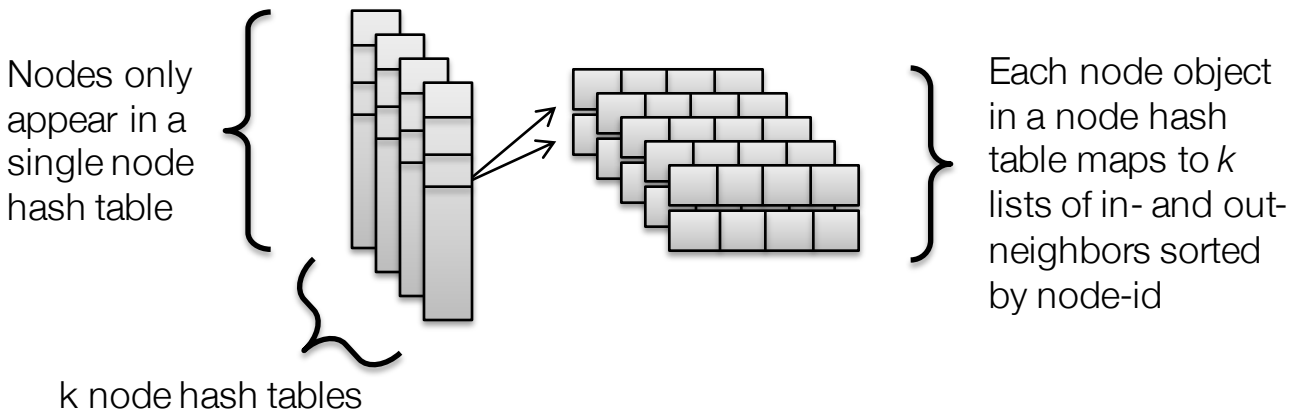
Idea 1: BGC

BGC (Bipartite Graph Collection): Collection of per-mode bipartite graphs



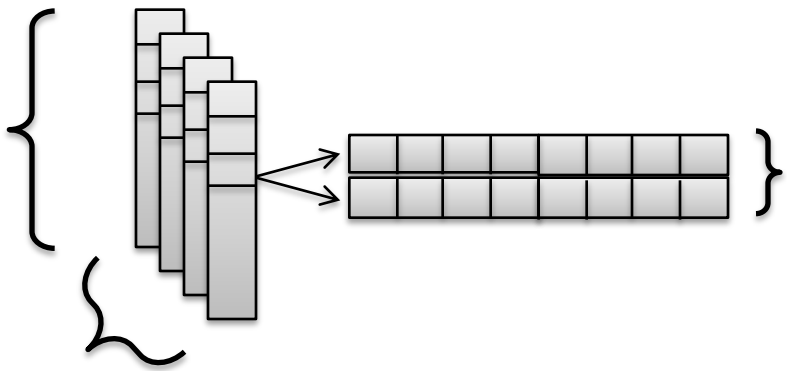
Idea 2: Hybrid

Hybrid: Collection of per-mode node hash tables along with individual per-mode adjacency lists



Idea 3: MNCA

MNCA (Multi-node hash table, consolidated adjacency lists): Per-mode node hash tables + big adjacency list

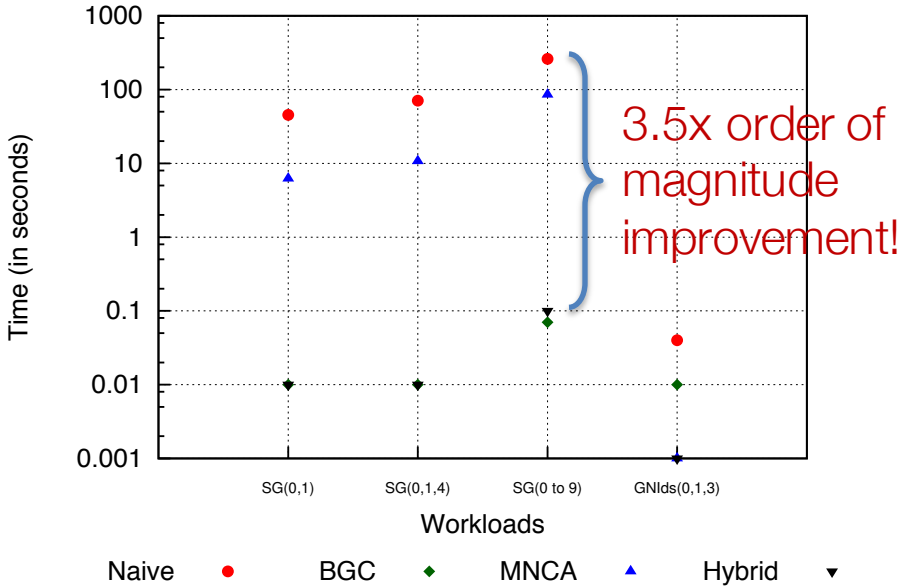


Nodes only appear in a single node hash table

k node hash tables

Each node object in a node hash table maps to a consolidated list of in- and out-neighbors sorted by (mode-id, node-id)

So, how do we do?

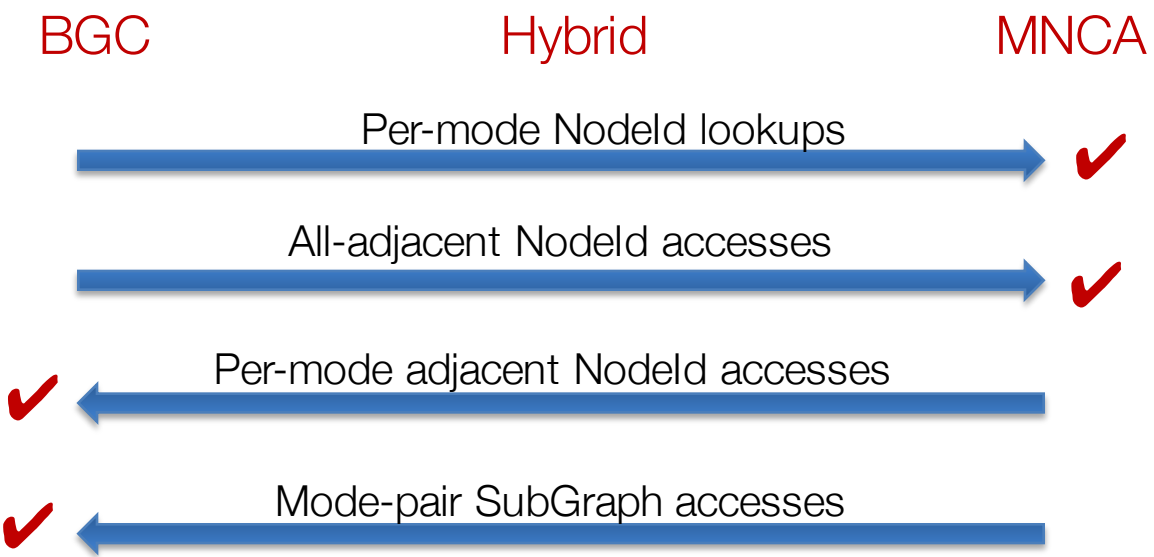


11 modes in total

- 10k nodes in modes 0-9; edges between all nodes
- 1M nodes in mode 10; edges between every node in mode 10 and all other nodes (total of 110B edges)

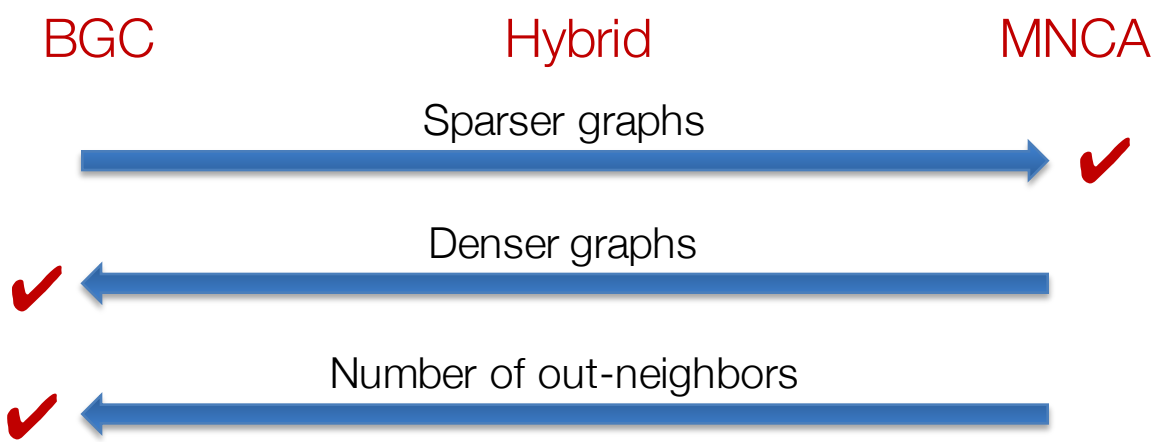
Tradeoffs by Workload

- Workload type:



Tradeoffs by Graph Type

- Graph type:

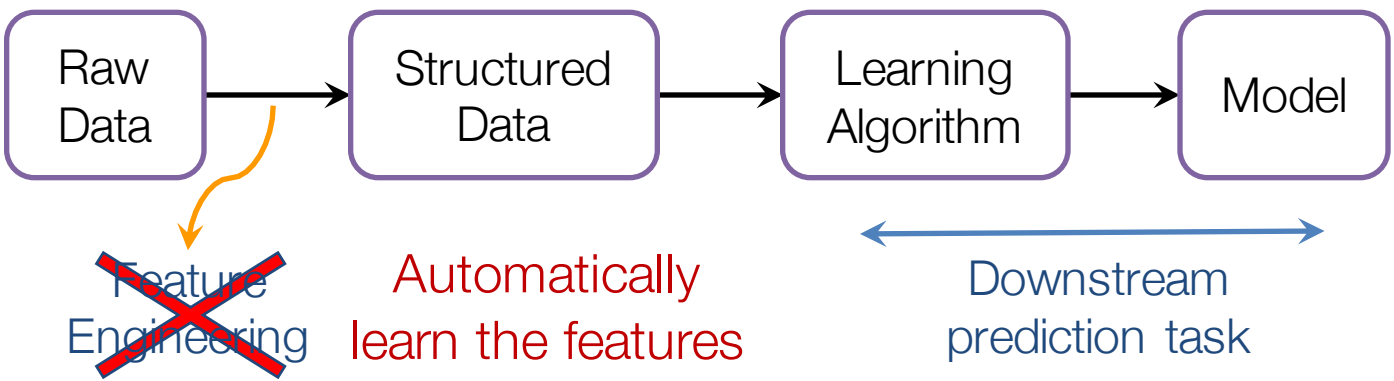


Latest Algorithms: Feature Learning in Graphs

node2vec: Scalable Feature Learning for Networks
A. Grover, J. Leskovec. KDD 2016.

Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: This feature, that feature. Every single time!



Feature Learning in Graphs

Goal: Learn features for a set of objects

Feature learning in graphs:

- Given: $G = (V, E)$
- Learn a function: $f : V \rightarrow \mathbb{R}^d$
  - Not task specific: Just given a graph, learn *f*. Can use the features for any downstream task!

Unsupervised Feature Learning

- Intuition: Find a mapping of nodes to d-dimensions that preserves some sort of node similarity
- Idea: Learn node embedding such that nearby nodes are close together
- Given a node *u*, how do we define nearby nodes?
  - $N_S(u)$ … neighbourhood of *u* obtained by sampling strategy *S*

Unsupervised Feature Learning

- Goal: Find embedding that predicts nearby nodes $N_S(u)$:

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

- Make independence assumption:

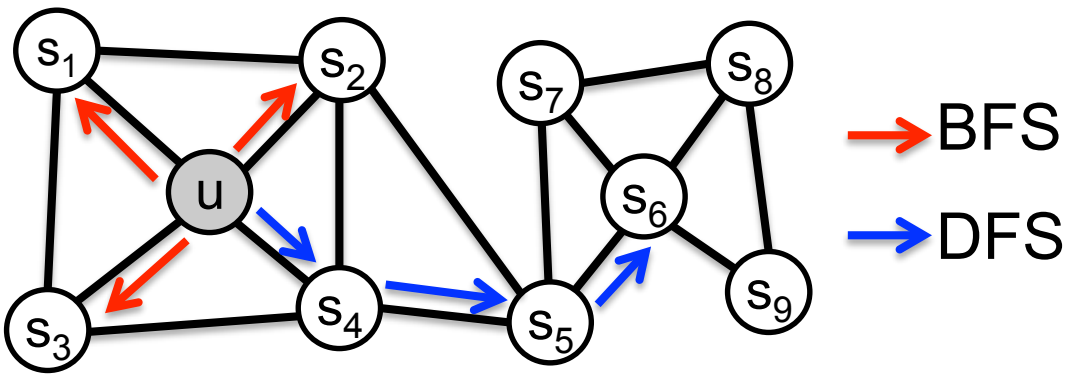
$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u))$$

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

Estimate $f(u)$ using stochastic gradient descent.

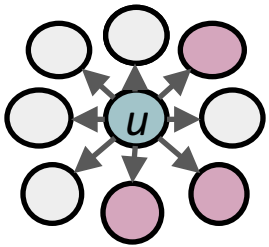
How to determine $N_S(u)$

Two classic search strategies to define a neighborhood of a given node:

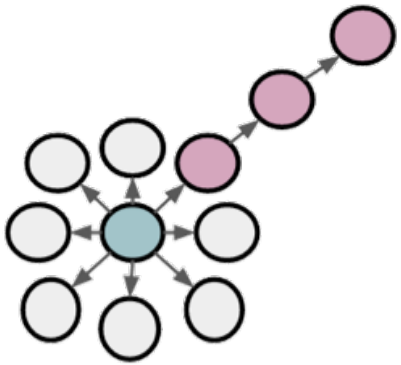


for $|N_S(u)| = 3$

BFS vs. DFS



BFS:
Micro-view of neighbourhood

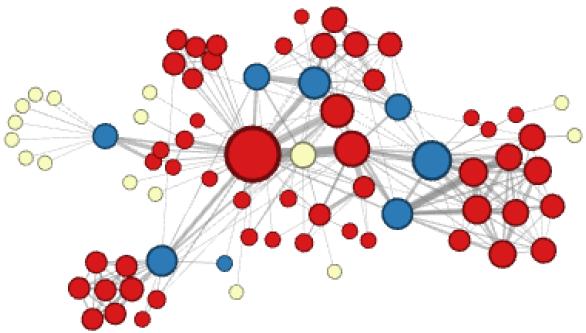


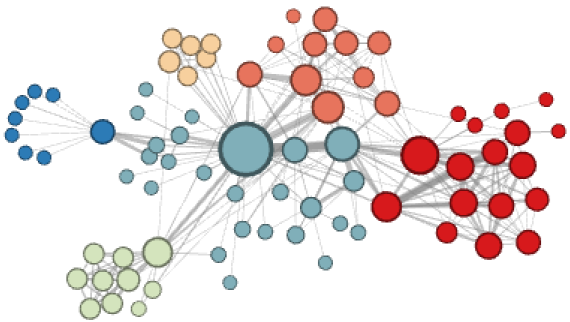
DFS:
Macro-view of neighbourhood

Structural vs. Homophilic equivalence

BFS vs. DFS

Structural vs. Homophilic equivalence



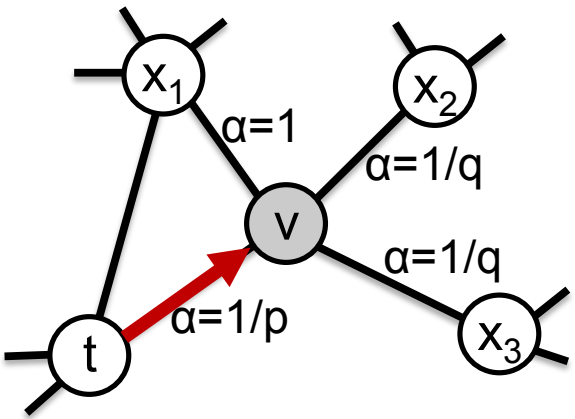


BFS-based:
Structural equivalence
(structural roles)

DFS-based:
Homophily
(network communities)

Interpolating BFS and DFS

- Biased random walk procedure, that given a node u samples $N_S(u)$



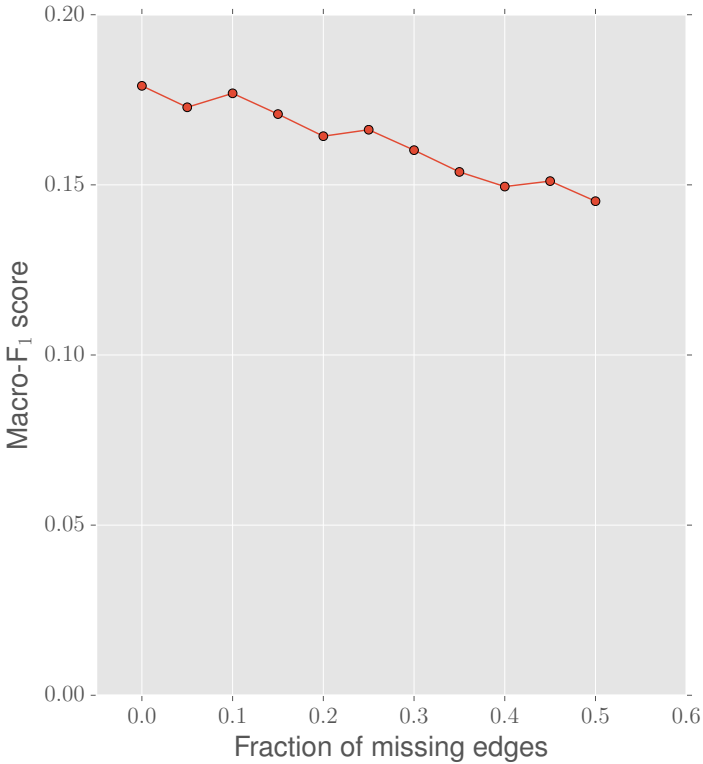
The walk just traversed (t, v) and aims to make a next step.

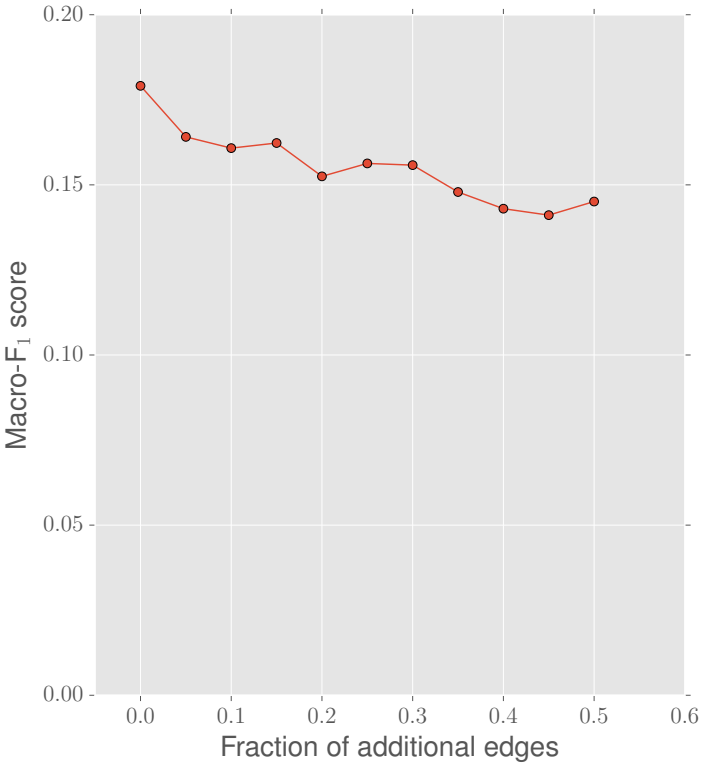
Multilabel Classification

Algorithm	Dataset		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

- Spectral embedding
- DeepWalk [B. Perozzi et al., KDD '14]
- LINE [J. Tang et al.. WWW '15]

Incomplete Network Data (PPI)



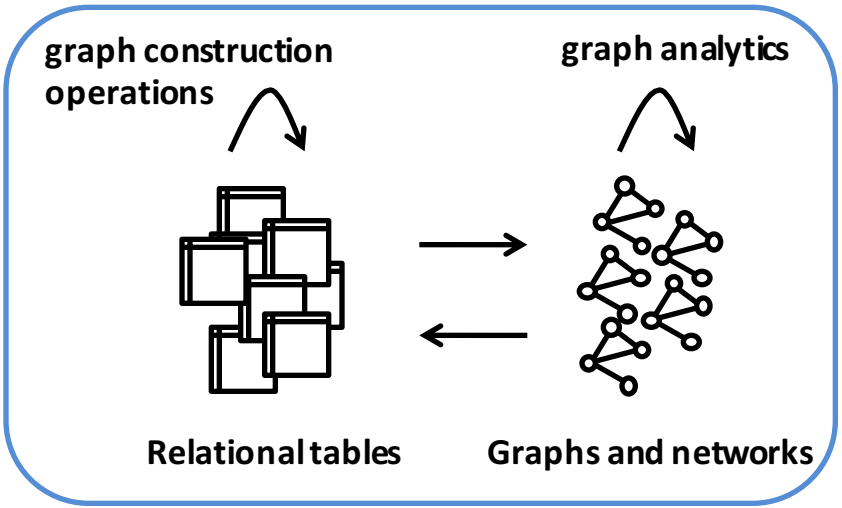


Conclusion

Conclusion

- Big-memory machines are here:
  - 1TB RAM, 100 Cores ≈ a small cluster
  - No overheads of distributed systems
  - Easy to program
- Most “useful” datasets fit in memory
- Big-memory machines present a viable solution for analysis of all-but-the-largest networks

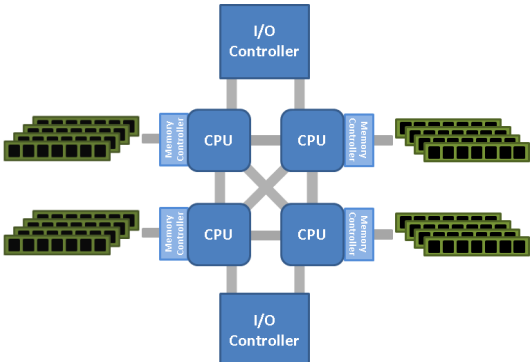
Graphs have to be Built



- Graphs have to built from data
  - Processing of tables and graphs

Multimodal Networks

- Graphs are more than wiring diagrams
- Multimodal network: A network of Networks
- Building scalable data structures
- NUMA architectures provide interesting new tradeoffs



Building Robust Systems

How to get robust performance always?

- Ongoing/future work
  - Better characterize the optimal representation required given workload and graph type
  - Try to dynamically switch representations when nodes get sufficiently high degrees or particular queries become more common
  - Benchmark on real data and real queries

References

- Papers:
  - SNAP: A General Purpose Network Analysis and Graph Mining Library.
    R. Sosic, J. Leskovec. ACM TIST 2016.
  - Ringo: Interactive Graph Analytics on Big-Memory Machines by Y. Perez, R Sosic, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, J. Leskovec. SIGMOD 2015.
  - node2vec: Scalable Feature Learning for Networks. A. Grover, J. Leskovec. KDD 2016.
- Software:
  - http://snap.stanford.edu/ringo/
  - http://snap.stanford.edu/snappy
  - https://github.com/snap-stanford/snap


THANKS!
@jure
http://snap.stanford.edu
Eve Online: Exodus developer CCP publisher CCP
© GameWallpapers.com hosted by JTLnet.com