Graph Analytics and Machine Learning problems

Alex Dimakis

based on joint works with
I. Mitliagkas, M. Borokhovich, C. Caramanis
E. Elenberg, K. Shanmugam

UT Austin

Thank you for coming to my talk. We will include some notes on the slides here.

Structure analytics and ML problems

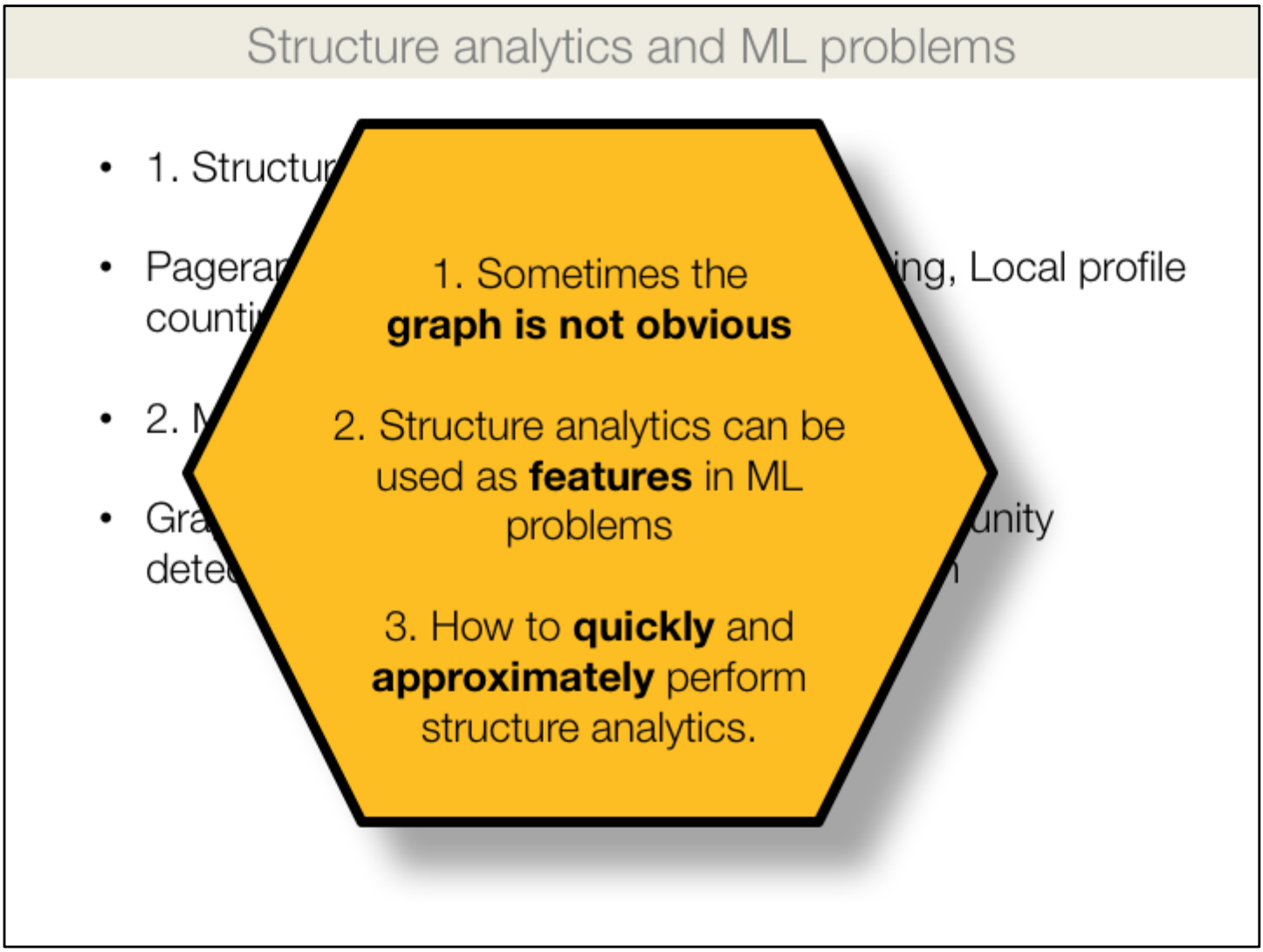
- 1. Structure analytics:
- Pagerank, Triangle counting, Profile counting, Local profile counting, Low-rank approximation
- 2. Machine Learning problems:
- Graph classification, Vertex Clustering, Community detection, Node classification, Role identification, finding influential vertices, finding sinks of epidemics, link prediction

In this talk, we connect two things: The first is Structure Analytics which means basic graph structure counting problems. For example finding the Pagerank of each vertex on the graph or counting the total number of triangles in the graph. Some of these measurements produce a single number for the whole graph and some others a number per vertex (or edge).

The second aspect are various machine learning problems on graphs. For example Graph classification: you are given a bunch of graphs and a label for each one. Example: for a chemical compound we extract a graph indicating its structure and we have a label if it is cancerogenic or not. There are a few standard datasets like that. For graph classification problems someone gives us a new graph and we need to classify it as cancerogenic or not.

A classification problem can apply to vertices of the graph, not the whole graph. For example, the facebook graph, is this particular user a real user or a fake profile.

In this overview talk we will see how structure analytics can relate to ML problems on graphs.



I have three things I would like you to walk out with:

1. For some problems, the graph may not be obvious.
   I will show two types of examples for this in a bit.

2. Counting the number of triangles in a graph may sound like a useless thing to do.
   I will argue that structure analytics (like triangle counts) can be used as features for vertex or graph classication tasks.
   They are basically features that capture the local geometry of a graph.

3. In my lab, our research project focuses on quickly approximating structure analytics for big graphs and showing they are useful as features.

1. Sometimes the **graph is not obvious**

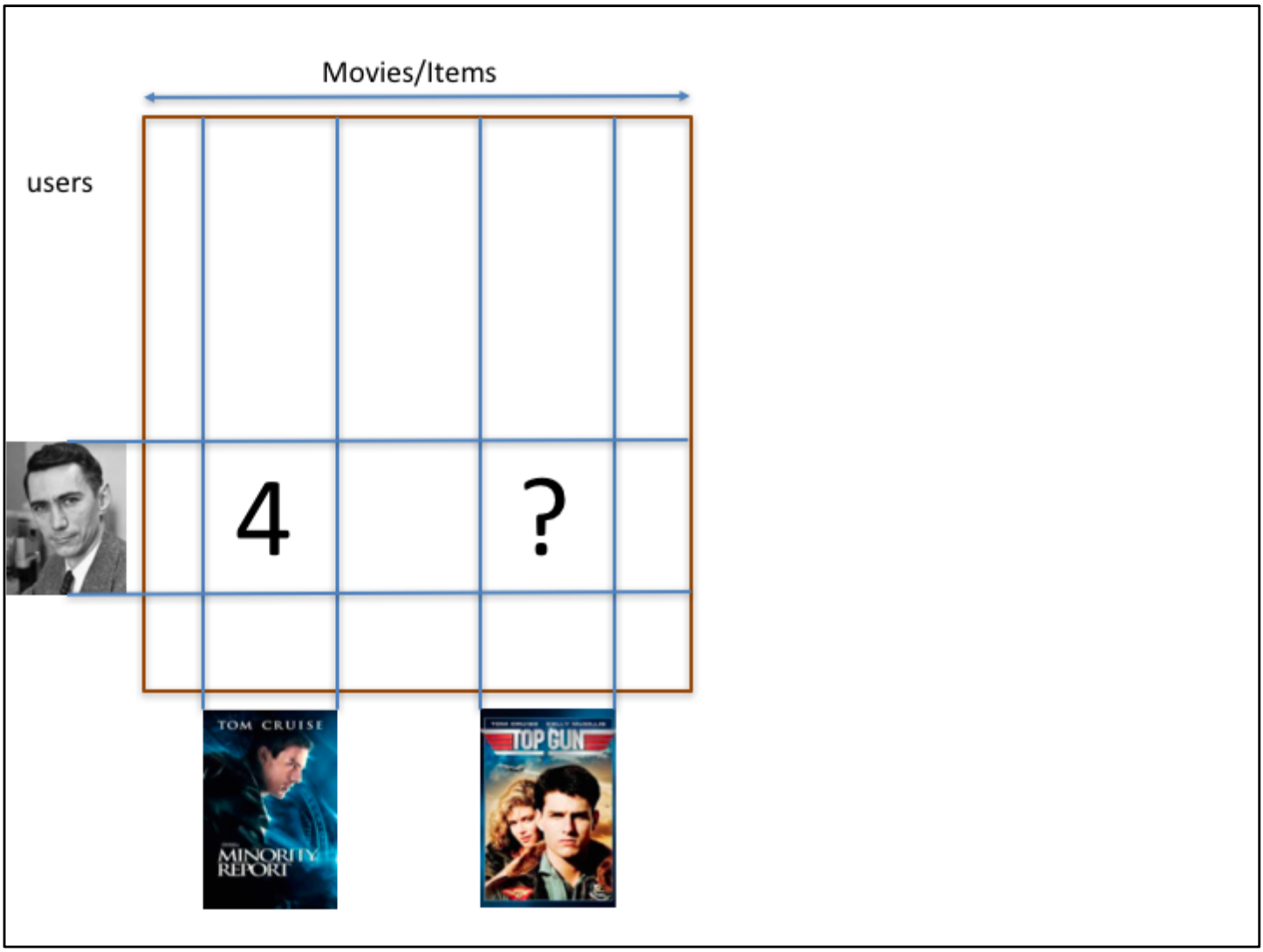
1a. it can be *a way to write an ML computation*
or 1b. *part of the modeling process*

example 1a: collaborative filtering/recommendations

example 1b: hotel bookings

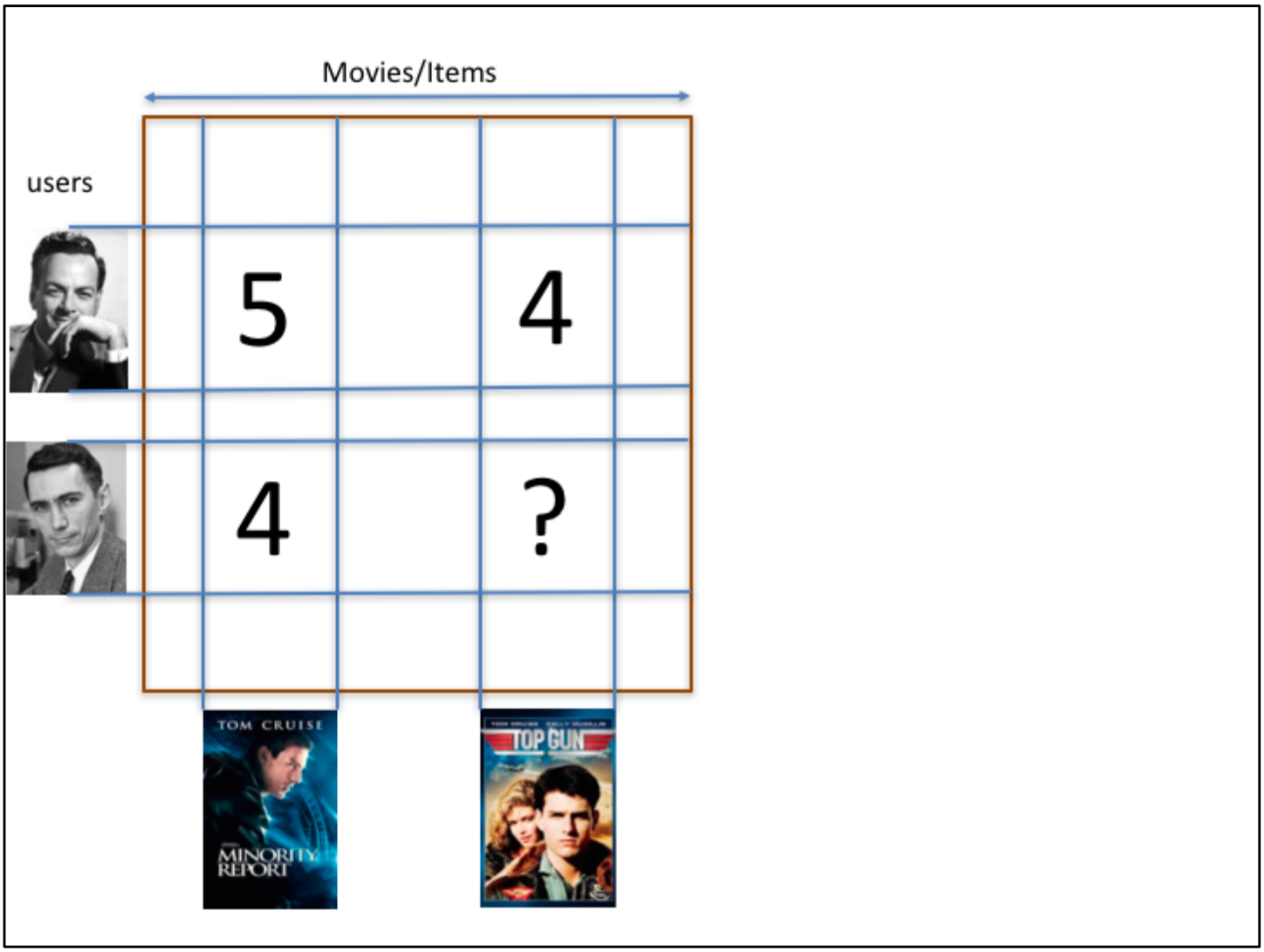
Lets start with our first point:

The graph may not be obvious in two ways:
1a: You may want to use the graph to assist the parallelization of your computation, even if there is no obvious graph in the original problem. We will see an example of that.
1b. The graph may be something you create as a data scientist, i.e. be part of the modeling process. We will see an example of that also.



Lets see an example of 1a.
Lets remember how low-rank matrix completion (aka Collaborative filtering) works for recommendations, in this simple example:
We have a matrix of users by movies (or more generally, items to recommend).
Our friend Claude here liked the movie ‘Minority report’ since he gave it a score of 4/5. We would like to predict if would like the movie ‘Top Gun’.

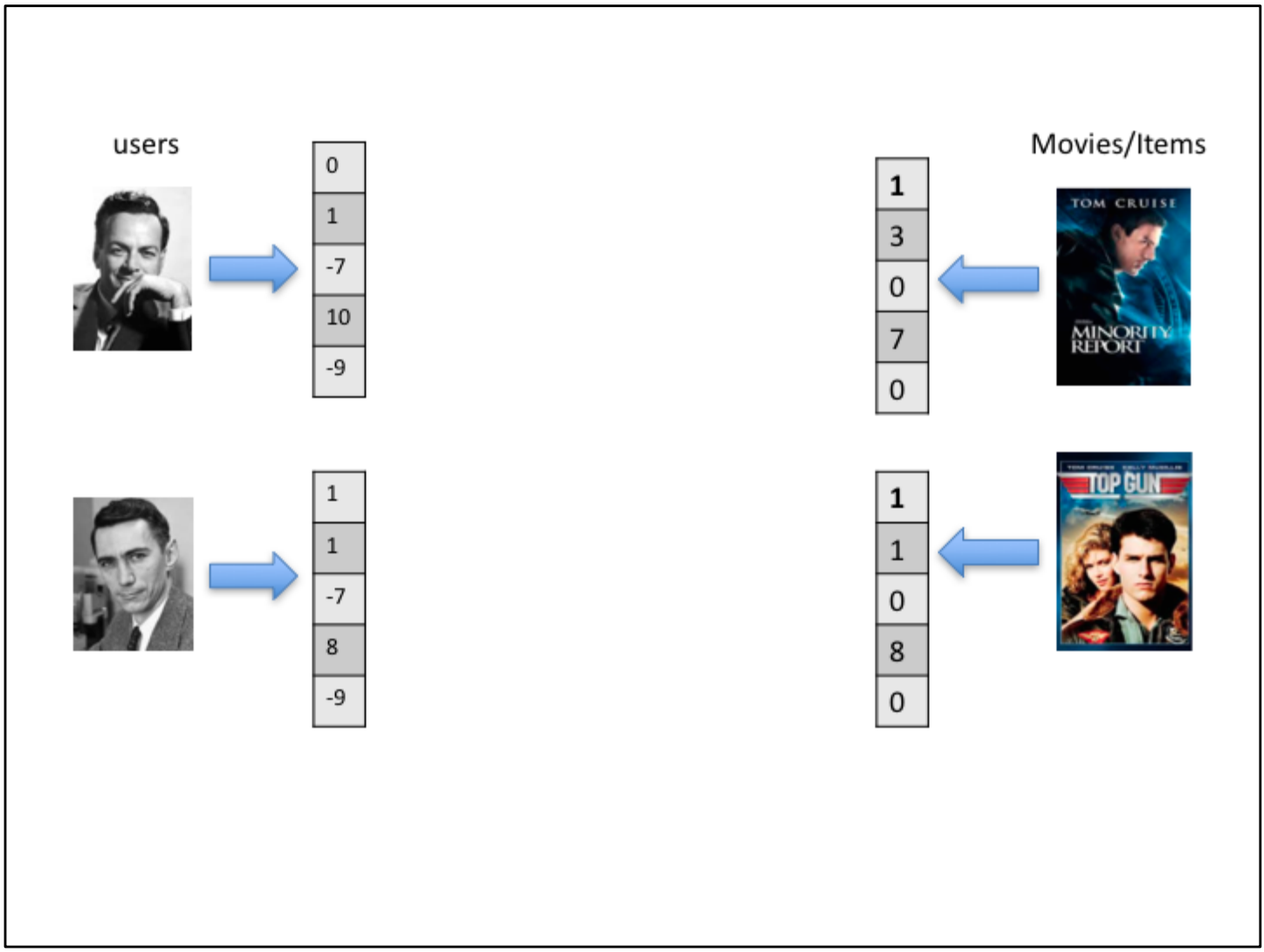
Unfortunately we do not have this observation.



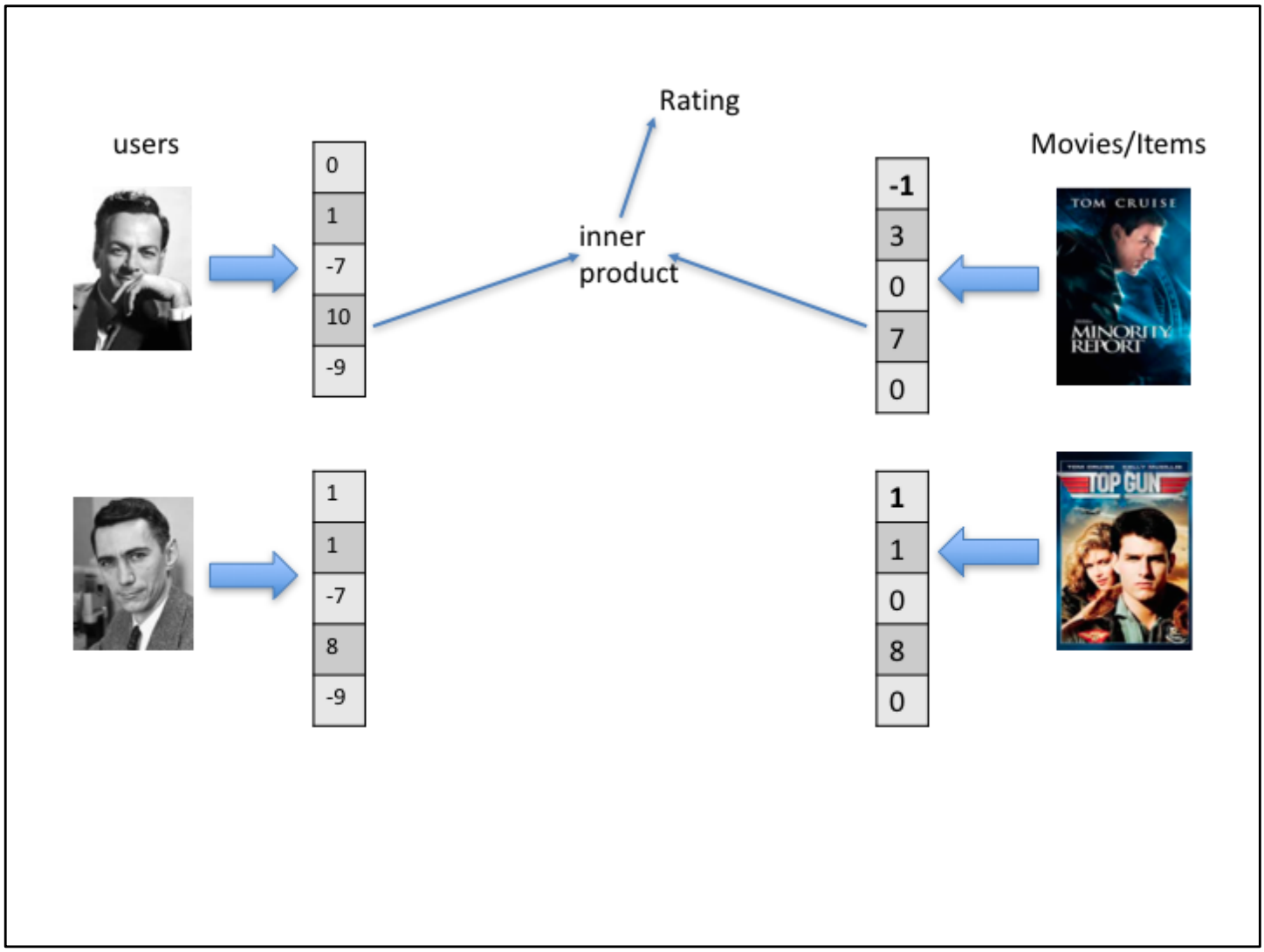
Fortunately this other user, Richard has watched both ‘Minority report’ and Top Gun and he liked both.

The key idea in all missing data approaches is to extrapolate from similar users.

Low-rank matrix approximation does this in a principled way.



Here is the idea:
We will try to find a vector for every user and a vector for every movie. These vectors will live in some low-dimensional space, say 5 dimensions only, in this example.

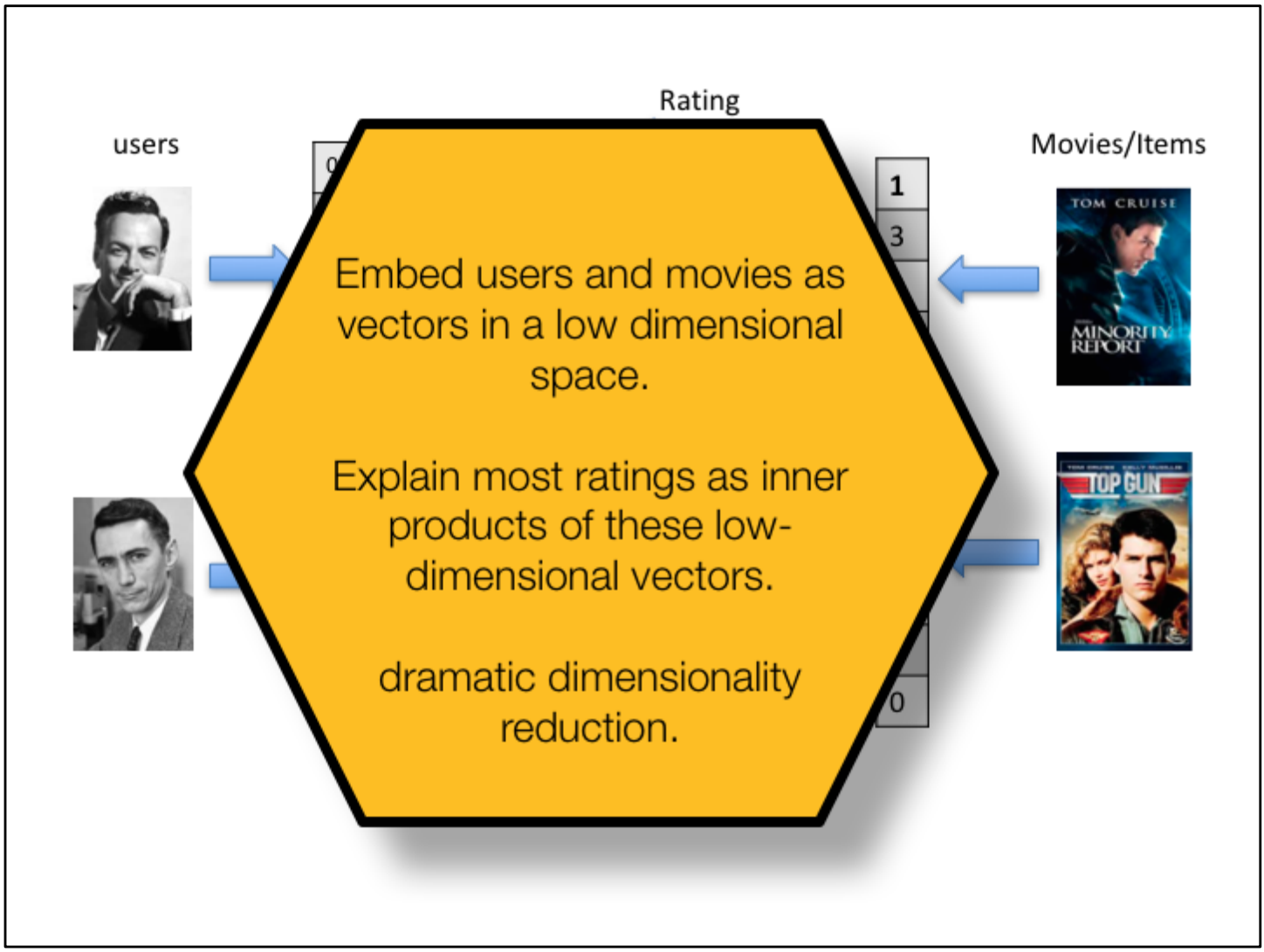


Our goal is to find such vectors so that the dot product of these two vectors produces the correct rating, for all the ratings we observe.

We are therefore trying to map users and movies to vectors so that the ratings are the inner products. In our example, the inner product (aka dot product) of
[0,1,-7,10,-9] , [-1,3,0,7,0] = 0*1 + 1*3 + 0*(-7)+ 10*7 + (-9)*0 = 73. We just multiply every coordinate with its corresponding one and add the result.

Why is this model reasonable? For every user, their vector represents how much they like different 'dimensions' of the movie. For example, for Richard, the first coordinate may represent how much he likes comedies (0, so doesn't care)
the second may be SCI-FI-ness (1 so he likes SCI-FI a bit), etc. The fourth coordinate may represent how much he likes Tom cruise in his movies, and 10 is a lot.

On the movie side, every coordinate tells us how much this dimension is present in this movie. So Minority report has -1 on comedy (so it is not) , 3 on SCI-FI-ness which is somewhat, and a 7 on its TomCruiseness dimension.



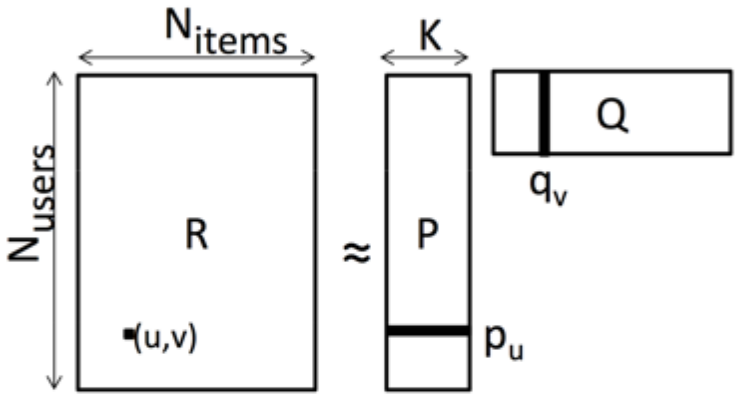
There are two magical things about this low-rank matrix model:

1. We do not have to know these magical dimensions.
2. the optimization algorithm will discover them automatically, by trying to fit vectors that explain the ratings (TomCruiseness is certainly one of them).

3. It actually works really well with very few dimensions.

4. Most of us think we have unique taste in things. However, 20 or 30 numbers can fully describe our taste in movies (and predict if we would like any movie, surprisingly well).

Collaborative Filtering/ Recommendation systems

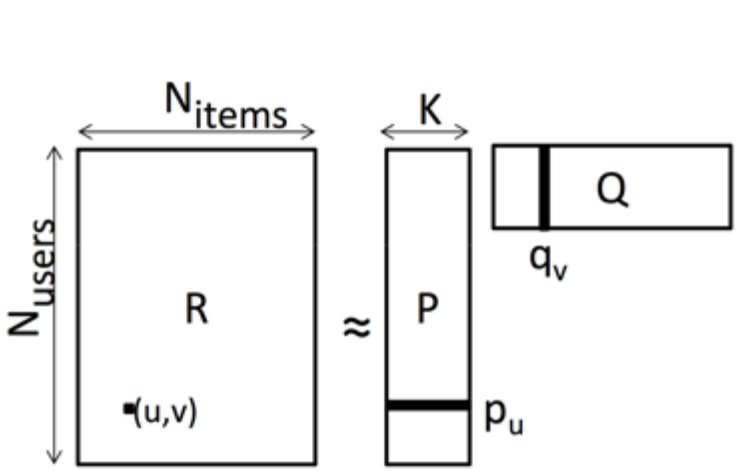


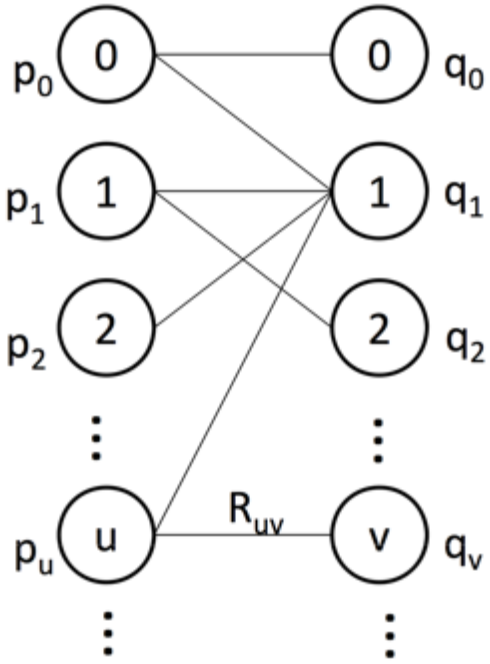
Theoretical guarantees for the convergence of Alternating minimization can be given under conditions for R:
Low-rank matrix completion using alternating minimization by P. Jain, P. Netapalli and S. Sanghavi. STOC 2013

So we have a computational problem that is written in this compact linear algebra way. Every observated rating R(u,v) should be approximately equal to the dot product of the row pu and the column qv. This means we are trying to approximate this incomplete ratings matrix R by a matrix or rank K. We can write a non-convex optimization problem for this and approximate its solution using alternating minimization or gradient descent.

Alternating minimization says: if I fix one of the two factors (say Q), what is the optimal P? We are typically choosing here to approximate each entry in squared error i.e. $(R(u,v) – pv^* qv)^2$ and we sum all these errors for all our observations. If you think about it, finding the optimal P for a fixed Q is a least squares problem. This is because R(u,v) and qv are fixed and these are just noisy linear equations in the entries of P.
So we find that P, fix it, and optimize over Q. We then repeat this process and hope it converges to a good answer. This answer will discover TomCruiseness and all the other factors hidden in our ratings.

Collaborative Filtering





This is all great, but what does this have to do with graphs ?

Well this computational problem can be very naturally written as a graph problem in a language like Pregel or Graphlab.

We create a bipartite graph with one vertex per row of P and one per column of Q. The state of each vertex is the vector in K dimensions that we are looking for.
Each observation R(u,v) corresponds to an edge on this graph. We can initialize the q's randomly and with a vertex program update each pu to approximately explain its observations (edges).
This is one least squares problem for each pu.

This will naturally parallelize this computation over a graph framework.

So the point (1a) is that the graph is created here to express the computation of the ML problem

in hand and make parallelization and scaling easy.

This is true for several ML problems when there is sparsity.

1. Sometimes the **graph is not obvious**

1a. it can be *a way to write an ML computation*
or 1b. *part of the modeling process*

~~example 1a: collaborative filtering/recommendations~~

example 1b: hotel bookings

We will now show an example for point 1b. How a graph can be the part of a modeling process.

Hotel recommendations

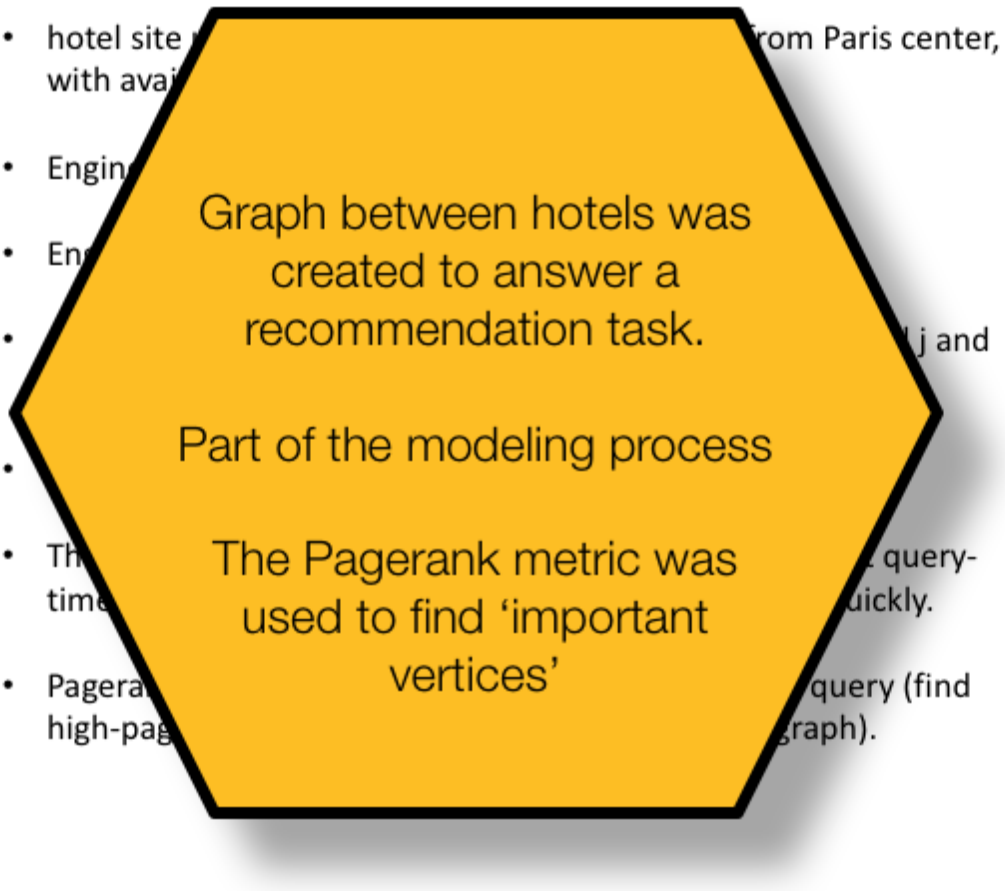
- hotel site user asks: find hotels, within 10 miles from Paris center, with availability on these dates.
- Engine finds 1000 hotels, must recommend 10-20.
- Engine forms a graph where vertices are hotels
- Add edge i → j if a previous user has seen both hotels i and j and preferred j
- Show 10-20 hotels with highest Pagerank to user
- There are many applications where graphs are formed at query-time and high Pagerank vertices must be discovered quickly.
- Pagerank cannot be pre-computed. Depends on the query (find high-Pagerank `Greeks in Austin' on the facebook graph).

We can recommend a hotel using its high pagerank in graph of preferences.

This preference graph is something we chose to create.

This simple method works quite well because Pagerank will prefer vertices (hotels) with a lot of incoming vertices, i.e. a lot of people preferring them. Further, If someone preferred hotel X over a good hotel,
this is a stronger signal that hotel X is great (and this is also captured by Pagerank: your importance is how many people point to you, and the more important they are , even better).

Hotel recommendations



Our storyline

1. ~~Sometimes the **graph is not obvious**~~
   - ~~1a. it can be a way to write an ML computation~~
   - ~~1b. it can be a part of the modeling process~~

2. Structure analytics can be used as **features** in ML problems

3. How to **quickly** and **approximately** perform structure analytics.

We now move to our second point. How to use structure as features.

Structure analytics as features

- Triangle counting: the number of triangles in the graph.
- Local triangle counting on a vertex v: the number of triangles that v participates in.
- How can this be used as a feature ?
- Consider the vertex classification problem: is this vertex a fake facebook user vs a real one.
- Fake fb accounts send requests and add friends at random.
- Real users have a higher local triangle count compared to fake.
- A classifier can exploit this. What about more structure information?

A fake profile participates in fewer triangles compared to a real user.

This is because your friends are quite likely to know each other, but a fake user typically just sends out random invitations to people that are more likely unconnected.

So the local triangle count of a vertex is a useful feature for that vertex.

Obviously, we will have other non-graph features like how often people login, their Ips etc and these can be more useful. But the point is that structure features can be useful.

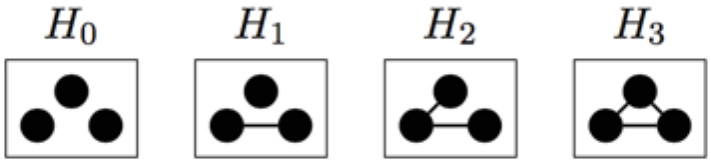
How can we get more geometrical structure information other than triangle counts ?

I.e. Some people who are hubs: connect different cliques, versus people who are just part of one big group, versus other types of geometry in the neighborhood of a vertex.

This may be useful in node classification problems for ad placement, biology applications etc.

the 3-profile of a graph

- Count the induced subgraphs formed by selecting all triples of vertices



Graph profiles are closely related to called Motifs and Graphlets and there is rich literature, e.g.:

J. Ugander, L. Backstrom, M. Park, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In 22nd International World Wide Web Conference, 2013.

D. O'Callaghan, M. Harrigan, J. Carthy, and P. Cunningham. Identifying discriminating network motifs in youtube spam. Feb. 2012

N. Przulj. Biological network comparison using graphlet degree distribution. Bioinformatics, 23(2):177–183, 2007

P. Ribeiro, F. Silva, and L. Lopes. Efficient parallel subgraph counting using g-tries. In IEEE International Conference on Cluster Computing, pages 217–226. Ieee, Sept. 2010.

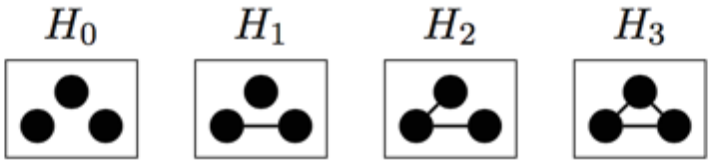
N. Shervashidze, K. Mehlhorn, and T. H. Petri. Efficient graphlet kernels for large graph comparison. In Proc. 20th International Conference on Artificial Intelligence and Statistics, pages 488–495, 2009

We will now introduce the 3-profile of a graph.
Lets consider all possible graphs on 3 vertices. H0 will be called the 'empty', H1 the 'edge', H2 the 'wedge' and H3 the 'triangle'.

Certainly H2 can be rotated, but it would be isomorphic. So we consider non-isomorphic graphs and these four are everything.

the 3-profile of a graph

- Count the induced subgraphs formed by selecting all triples of vertices



Definition

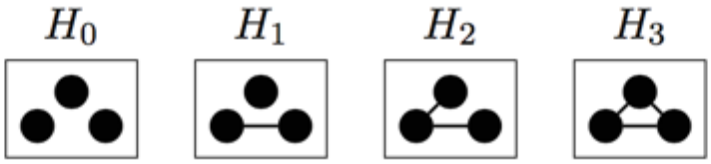
Let n_i be the number of H_i's in a graph G. The vector $\mathbf{n}(G) = [n_0, n_1, n_2, n_3]$ is called the 3-profile of G.

- Always sums to $\binom{|V|}{3}$, the total number of 3-subgraphs

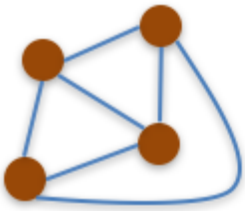
The 3-profile of a graph is a vector of 4 numbers counting how many times each such subgraph appears in the big graph.
n3 is just the triangle count of the graph so the 3-profile is a generalization of triangle counting.

3-profile examples

- Count the induced subgraphs formed by selecting all triples of vertices



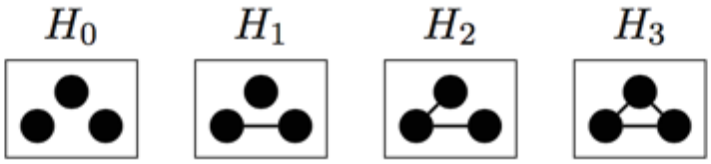
For the graph K_4 the 3-profile is



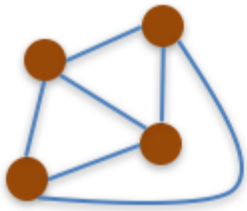
For example lets find the 3-profile of K4, the complete graph on 4 vertices.

3-profile examples

- Count the induced subgraphs formed by selecting all triples of vertices



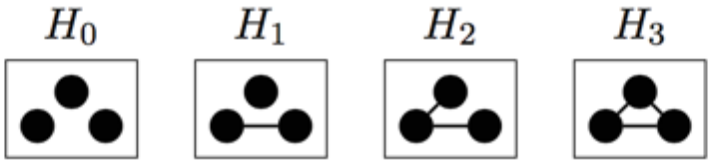
For the graph K_4 the 3-profile is:
[0,0,0,4]



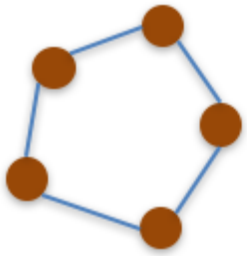
check that you see why it is [0,0,0,4]
i.e. 4 triangles and no other subgraphs.

3-profile examples

- Count the induced subgraphs formed by selecting all triples of vertices



For the graph C_5 the 3-profile is:
[0,5,5,0]



For the C5 (cycle on 5 vertices) the profile is [0,5,5,0].
Sanity check: The sum is 5+5=10 which is 5 choose 3 , i.e. all triplets on 5 vertices.

local 3-profiles

- For each vertex compute its local 3-profile (the number of triangles, wedges *etc* it participates in)
- This is a feature vector for each graph vertex, encoding the local structure of a vertex.
- Can capture if a vertex is a local hub, a part of a clique, etc.
- This is a useful structure feature for many node classification and graph classification tasks.
- How to compute them at scale for each vertex?

Now the interesting question is to compute for every vertex of the graph, its local 3-profile, i.e. how many subgraphs it participates in.

This captures the local structure we wanted.

Our storyline

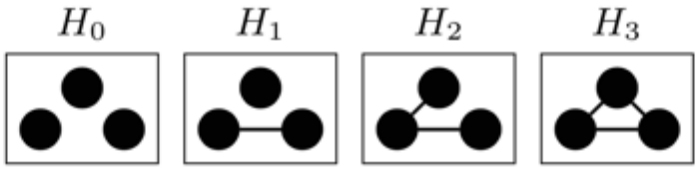
~~1. Sometimes the **graph is not obvious**~~
- ~~1a. it can be a way to write an ML computation~~
- ~~1b. it can be a part of the modeling process~~

~~2. Structure analytics can be used as **features** in ML problems~~

3. How to **quickly** and **approximately** perform structure analytics.
- 3a. Approximate 3-profile and 4-profile counts efficiently
- 3b. Quickly find high pagerank vertices

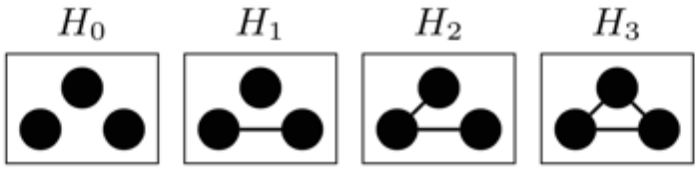
E. R. Elenberg et al. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs, *KDD* 2015
E. R. Elenberg, et al. Distributed Estimation of Graph 4-profiles, *WWW* 2016
I. Mitliagkas, et al. FrogWild! Fast PageRank Approximations on Graph Engines. VLDB 2015

Counting 3-profiles



- Triangle sparsifiers: sub-sample each edge independently
- Unbiased estimator with concentration (Tsourakakis et al.)

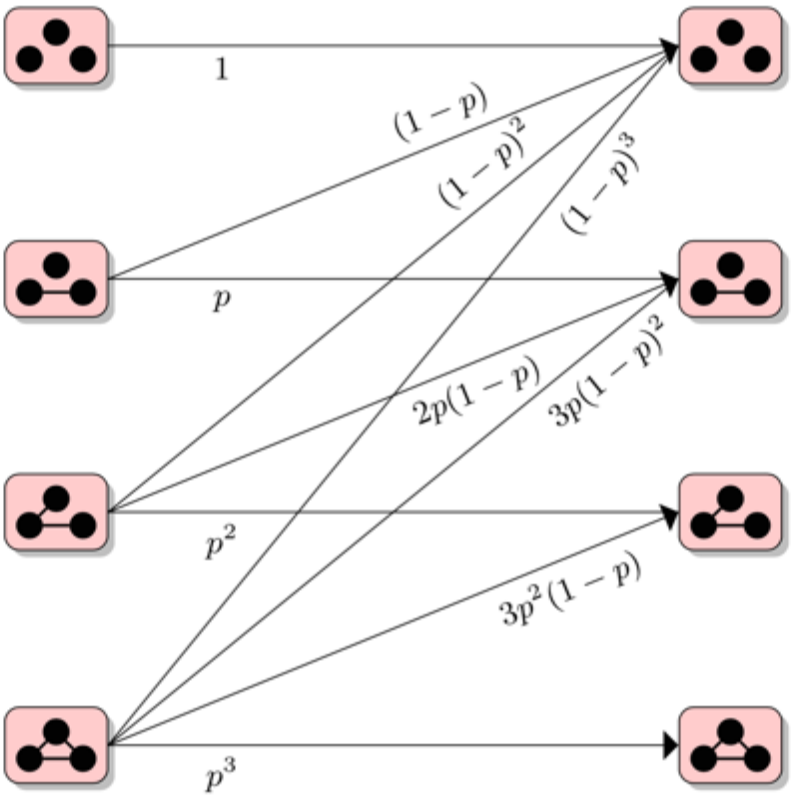
Counting 3-profiles



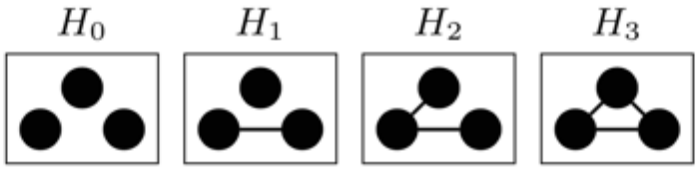
- Triangle sparsifiers: sub-sample each edge independently
- Unbiased estimator with concentration (Tsourakakis et al.)

- (based on: Kim and Vu, Concentration of multivariate polynomials and tis applications, Combinatorica 2000)

- Our result: Introduce 3-profile sparsifiers (can use linear transformation to estimate full 3-profile)


1
$(1-p)$
$(1-p)^2$
$(1-p)^3$
p
$2p(1-p)$
$3p(1-p)^2$
p^2
$3p^2(1-p)$
p^3

Counting 3-profiles



- Triangle sparsifiers: sub-sample each edge independently
- Unbiased estimator with concentration (Tsourakakis et al.)

- Introduce 3-profile sparsifiers (can use linear transformation to estimate full 3-profile)
- Show concentration using Kim-Vu and polynomial decomposition (beyond totally positive polynomials)

- Can count full 3-profile almost same time as counting triangles.

Our storyline

~~1. Sometimes the **graph is not obvious**~~
- ~~1a. it can be a way to write an ML computation~~
- ~~1b. it can be a part of the modeling process~~

~~2. Structure analytics can be used as **features** in ML problems~~

3. How to **quickly** and **approximately** perform structure analytics.
- ~~3a. Approximate 3-profile and 4-profile counts efficiently~~
- 3b. Quickly find high pagerank vertices

E. R. Elenberg et al. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs, *KDD* 2015
E. R. Elenberg, et al. Distributed Estimation of Graph 4-profiles, *WWW* 2016
I. Mitliagkas, et al. FrogWild! Fast PageRank Approximations on Graph Engines. VLDB 2015

pagerank approximation: frogwild

frogwild

- Distributed approximation algorithm for high-pagerank vertices
- Implemented in Graphlab- 7-10x faster than normal Pagerank function
- Provable approximation guarantees for graphs with no bad structure.

PageRank [Page et al., 1999]

Normalized Adjacency Matrix

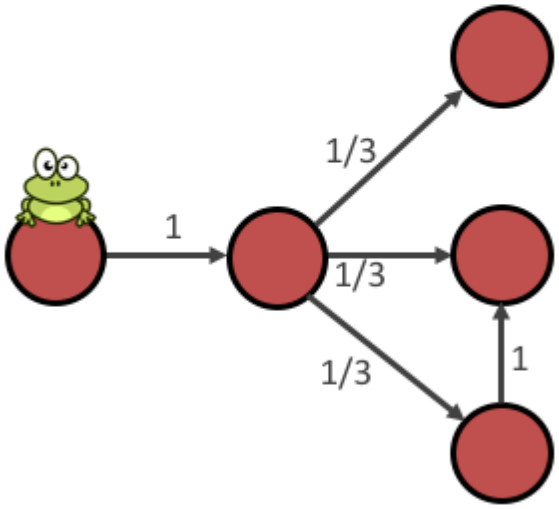
$$P_{ij} = \frac{1}{\mathrm{d}_{\mathrm{out}}(j)}, \quad (j,i) \in G$$

Augmented Matrix $p_T \in [0,1]$

$$Q_{ij} = (1 - p_T)P_{ij} + \frac{p_T}{n}$$

PageRank Vector $\pi = Q\pi$

Power Method $Q^t p^0 \to \pi$



31

Random walk Interpretation

Frog walks randomly on graph

Next vertex chosen uniformly at random

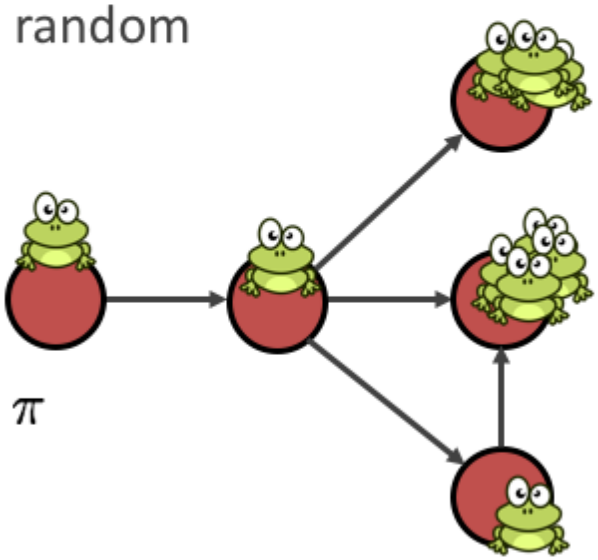
Teleportation

Every step: teleport

w.p. p_T

Sampling after t steps

Frog location gives sample from π



PageRank Vector

Many frogs, estimate vector π

32

Pagerank Approximation

lemma: A few random-walkers (e.g. 600k in 40m vertex graph) will suffice to estimate large entries of π vector whp. (mixing time and Chernoff bounds)

33

Pagerank Approximation

lemma: A few random-walkers (e.g. 600k in 40m vertex graph) will suffice to estimate large entries of π vector whp. (mixing time and Chernoff bounds)

problem 1: teleportations create global traffic. Unsuitable for distributed environments

Pagerank Approximation

lemma: A few random-walkers (e.g. 600k in 40m vertex graph) will suffice to estimate large entries of π vector whp. (mixing time and Chernoff bounds)

problem 1: teleportations create global traffic. Unsuitable for distributed environments

Solution: Start frogs uniformly, run for geometric random time that is $p_{stop} = \frac{1}{T_{mix} + c}$

35

Pagerank Approximation

lemma: A few random-walkers (e.g. 600k in 40m vertex graph) will suffice to estimate large entries of π vector whp. (mixing time and Chernoff bounds)

problem 1: teleportations create global traffic. Unsuitable for distributed environments

Solution: Start frogs uniformly, run for geometric random time that is $p_{stop} = \frac{1}{T_{mix} + c}$

lemma 2: c can be chosen to make this process same as rw for mixing time + teleportations.

Final result

Mass captured by top-k set, S, of estimate from N frogs after t steps

$$\pi(S) \geq \mathrm{OPT} - 2\epsilon \qquad \text{w.p. } 1 - \delta$$

where

$$\epsilon < \sqrt{k}\lambda_2^t + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1 - p_S^2)p_\cap(t)\right]}$$

Final result

Mass captured by top-k set, S, of estimate from N frogs after t steps

$$\pi(S) \geq \mathrm{OPT} - 2\epsilon \qquad \text{w.p. } 1 - \delta$$

where

$$\epsilon < \sqrt{k}\lambda_2^t + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1 - p_S^2)p_\cap(t)\right]}$$

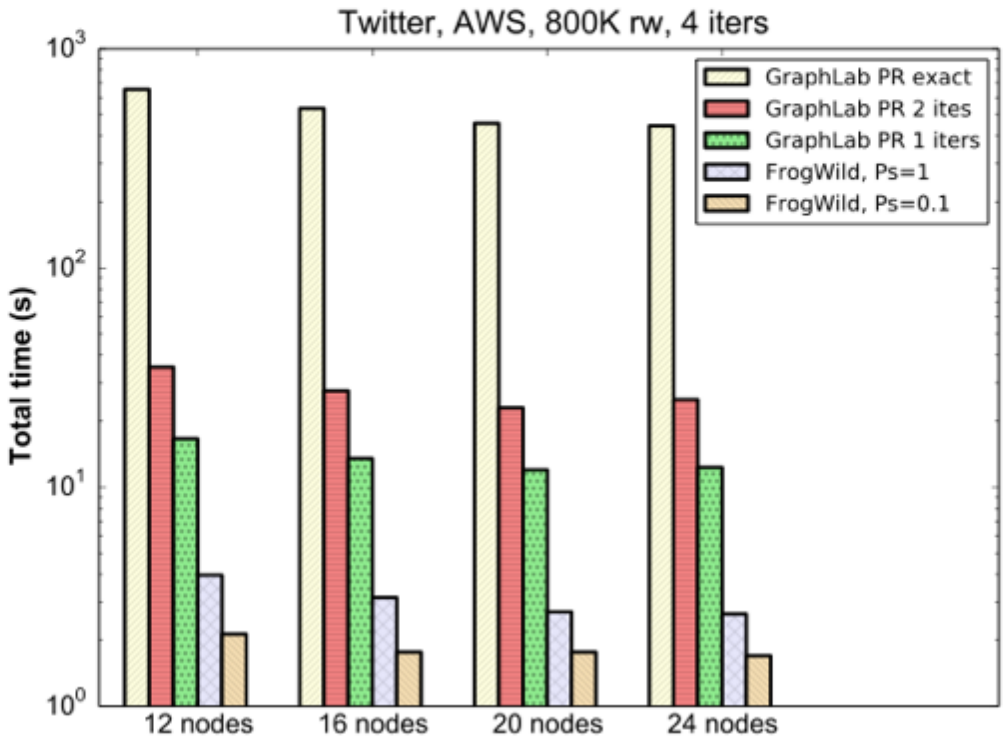
probability two Frogs meet after t steps starting from uniform should be small.

Bad graph: star

Good graph: Complete graph, expander graph

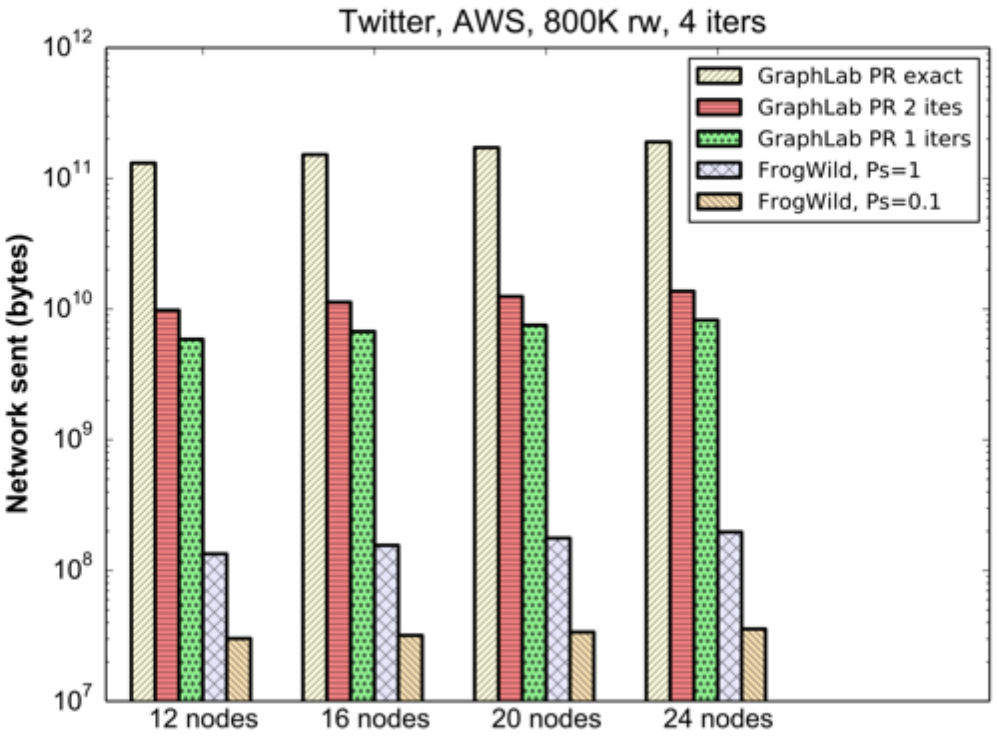
38

Experimental Results
(Twitter graph: 40m vertices 1.4B edges)



39

Experimental Results
(Twitter graph: 40m vertices 1.4B edges)



40

Our storyline is done

1. Sometimes the **graph is not obvious**
   1a. it can be a way to write an ML computation
   1b. it can be a part of the modeling process

2. Structure analytics can be used as **features** in ML problems

3. How to **quickly** and **approximately** perform structure analytics.
   3a. Approximate 3-profile and 4-profile counts efficiently
   3b. Quickly find high Pagerank vertices

Working currently on fMRI applications.

E. R. Elenberg et al. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs, *KDD* 2015
E. R. Elenberg, et al. Distributed Estimation of Graph 4-profiles, *WWW* 2016
I. Mitliagkas, et al. FrogWild! Fast PageRank Approximations on Graph Engines. VLDB 2015

http://users.ece.utexas.edu/~dimakis/

Pointers and links

- Papers
- E. R. Elenberg, et al. Beyond Triangles: A Distributed Framework for Estimating 3-profiles of Large Graphs, *KDD* 2015.
  - https://arxiv.org/abs/1506.06671
- E. R. Elenberg, et al. Distributed Estimation of Graph 4-profiles, *WWW* 2016.
  - https://arxiv.org/abs/1510.02215
- I. Mitliagkas, et al. Frogwild! Fast PageRank Approximations on Graph Engines, *VLDB* 2015.
  - https://arxiv.org/abs/1502.04281

Pointers and links

- Code
- https://github.com/eelenberg/3-profiles
- https://github.com/eelenberg/4-profiles
- https://github.com/michaelbor/frogwild

- References

- C. E. Tsourakakis, et al. Triangle Sparsifiers. *Journal of Graph Algorithms and Applications* 15(6), 2011.
- P. Berkhin. A survey on pagerank computing. *Internet Mathematics* 2(1), 2005.
- Satish, et al. Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets, *SIGMOD* 2014.

Acknowledgements

- This research was supported by:
- NSF Division of Computing and Communication Foundations (CCF)
- CCF 1344364, 1407278, 1422549, 1618689 and
- ARO YIP W911NF-14-1-0258 in the Network Sciences program
- Research support by the Hartwig Fellowship, Google and Amazon

Many thanks!

fin

http://users.ece.utexas.edu/~dimakis/

Twitter @alexb80

Experimental Results