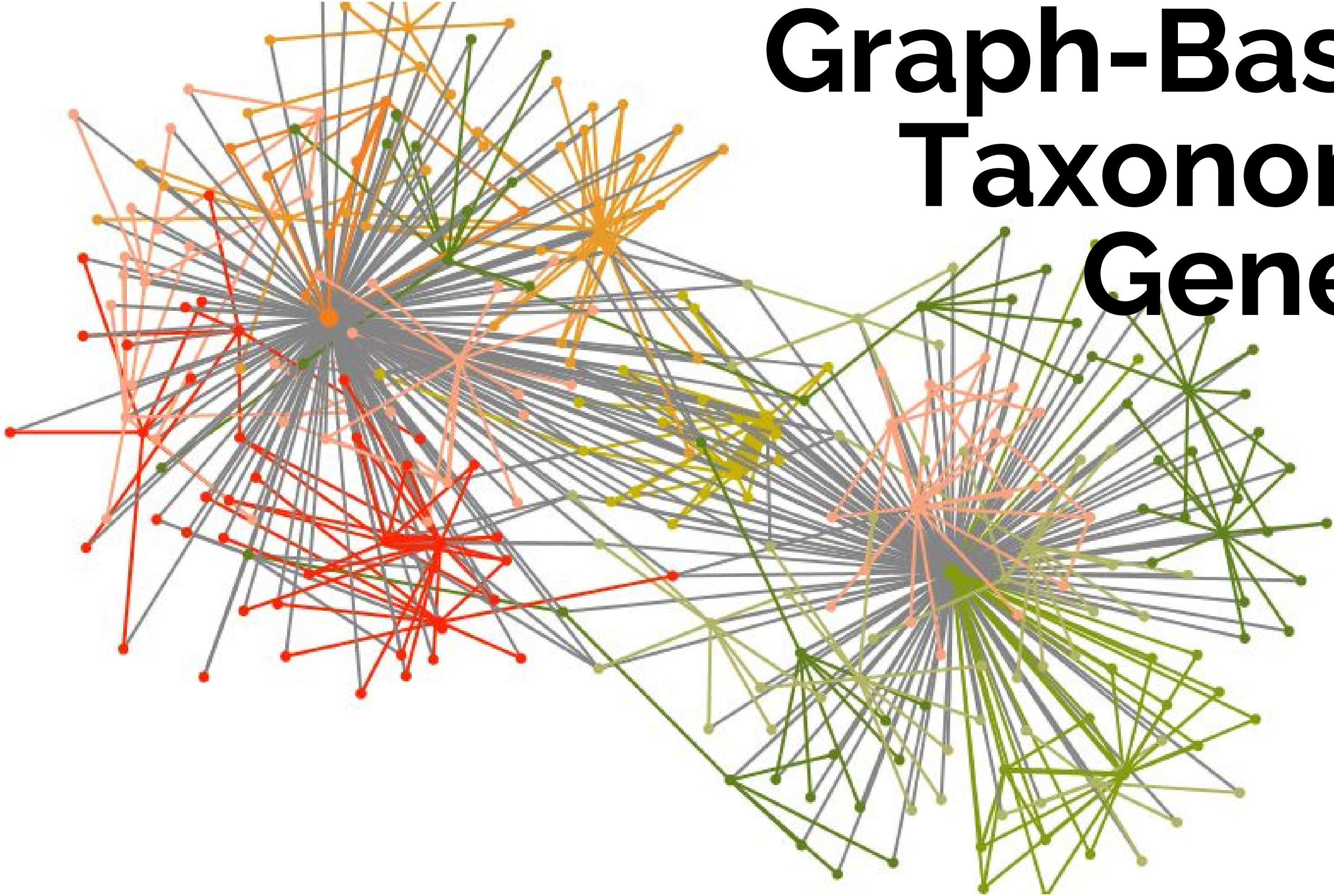
Graph-Based Taxonomy Generation



Sources

The bulk of the taxonomy generation algorithm itself, including the creation of the weighted graph, is owed to [Treeratpituk et al.2013] P Treeratpituk, M Khabsa, and CL Giles. 2013 Graph-based Approach to Automatic Taxonomy Generation (GraBTax) *arXiv:1307.1718v1 [cs.IR]*](https://arxiv.org/abs/1307.1718v1)

Graph partitioning is done via METIS, under the APL 2.0 [Karypis and Kumar1999] G Karypis and V Kumar. 1999 A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, pp. 359—392, 1999.(http://glaros.dtc.umn.edu/gkhome/fetch/papers/mlSIAMSC99.pdf)

Python-Metis interop is thanks to https://github.com/kw/metis-python, under MIT License

And contributors.

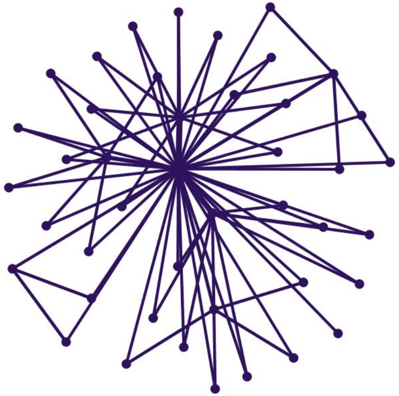
1. Overview

Generating taxonomies from a corpus can be a large task:

➔ **Expensive**
Human annotators cost time and money

➔ **Scale issues**
The human cost makes it time consuming to update the catalog -- forget about real-time

➔ **Fidelity**
It's anyone's guess if the taxonomy in an annotator's head matches the data you actually have.



How many **taxonomies** are there for a given corpus?

Many! It depends on perspective.

Camera Equipment > Accessories > Batteries ?

Electronics > Accessories > Batteries ?

Both?

But the trick is getting it right.

Two approaches

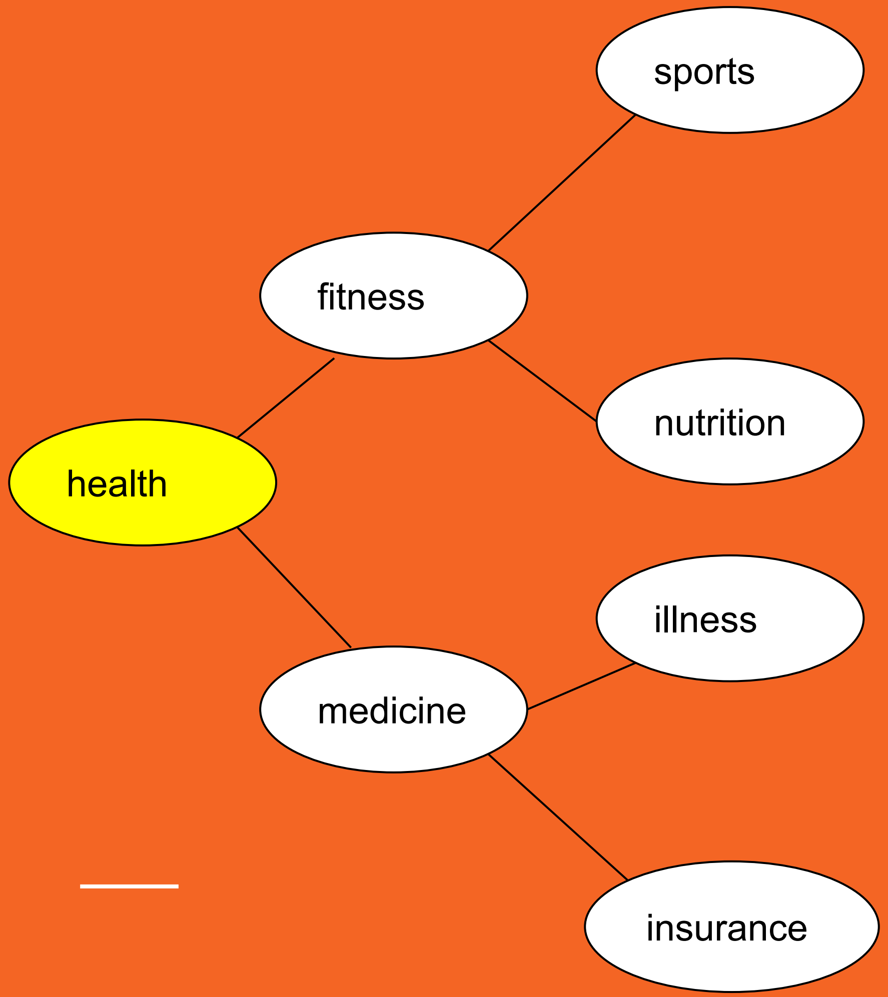
1. **Query-Independent**

Pros: consistent view of the corpus
Cons: Global behavior -- assumes there actually is a single taxonomy

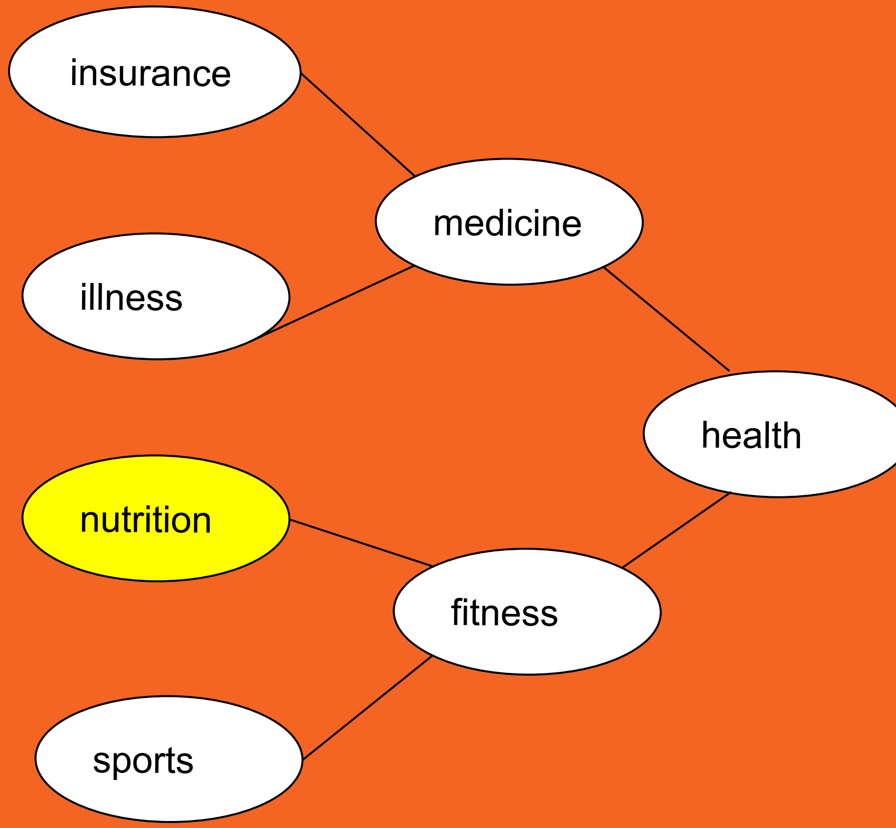
Two approaches

2. Query-Dependent
Pros: Local behavior -- relationships change depending on what you're looking for.
Cons: no consistent, static taxonomy. No global view.

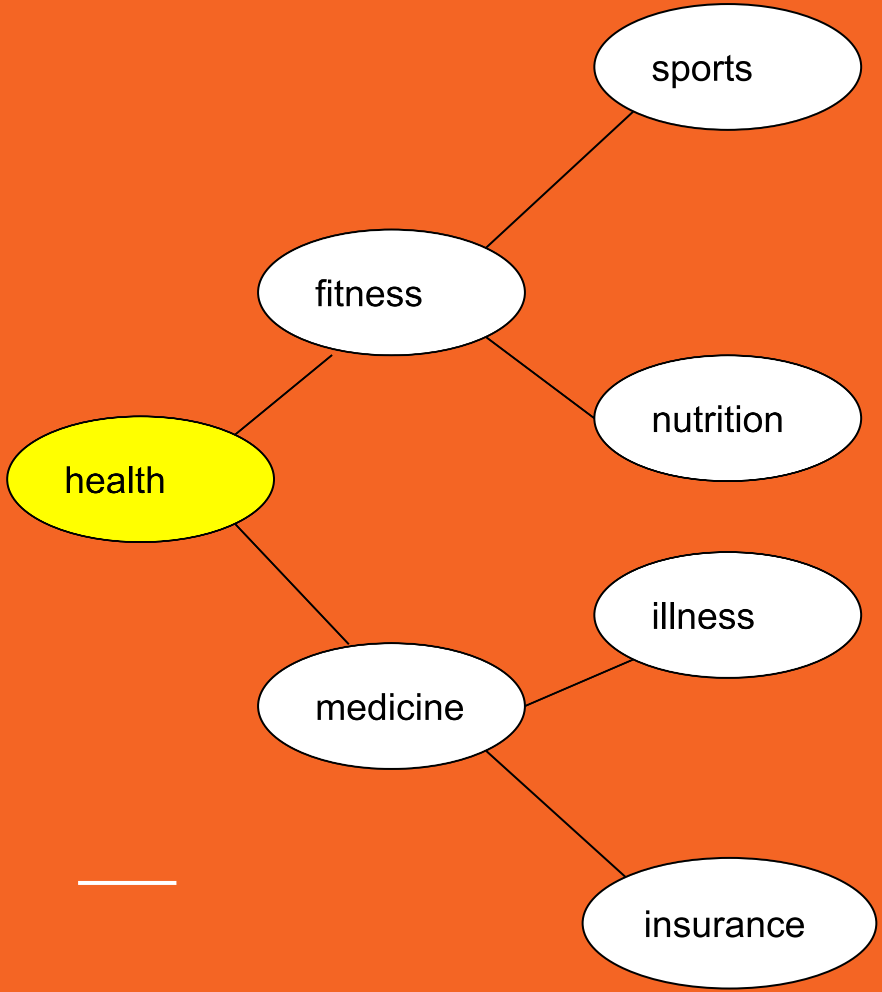
A query-independent taxonomy means context never changes.



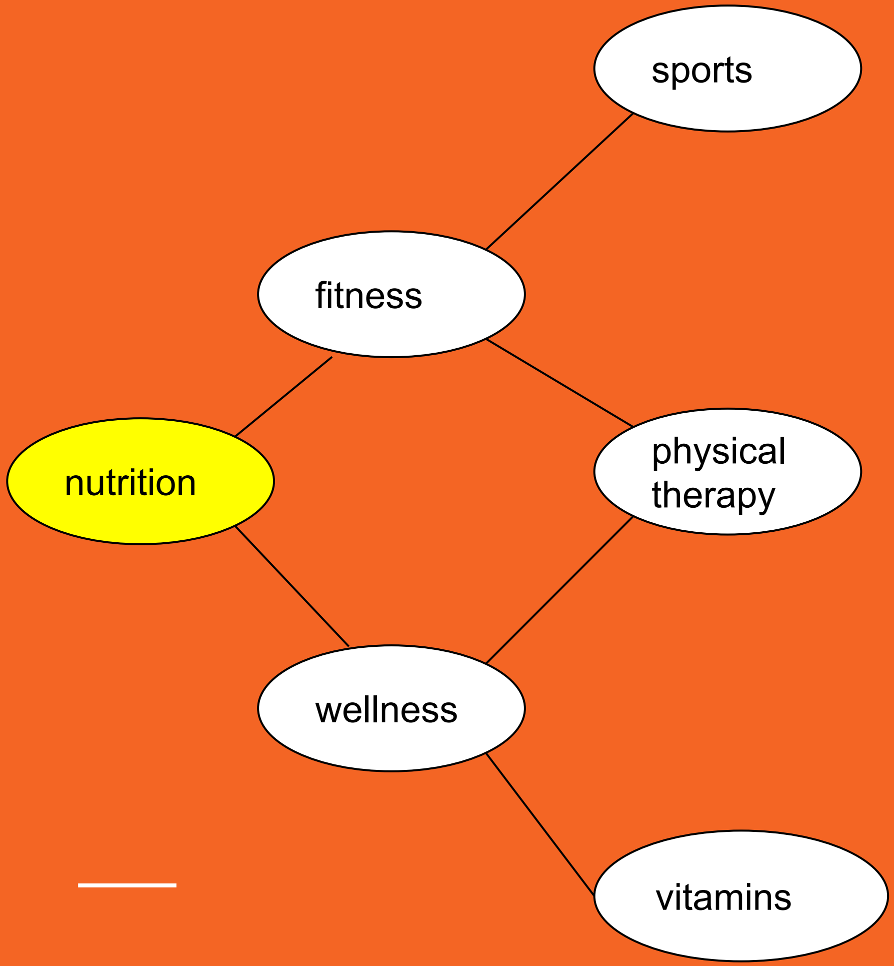
The view of the graph is the same from every vertex.



A query-dependent taxonomy has a different perspective from other vertices.



The perspective of the graph changes with context. Local behavior affects the perspective.



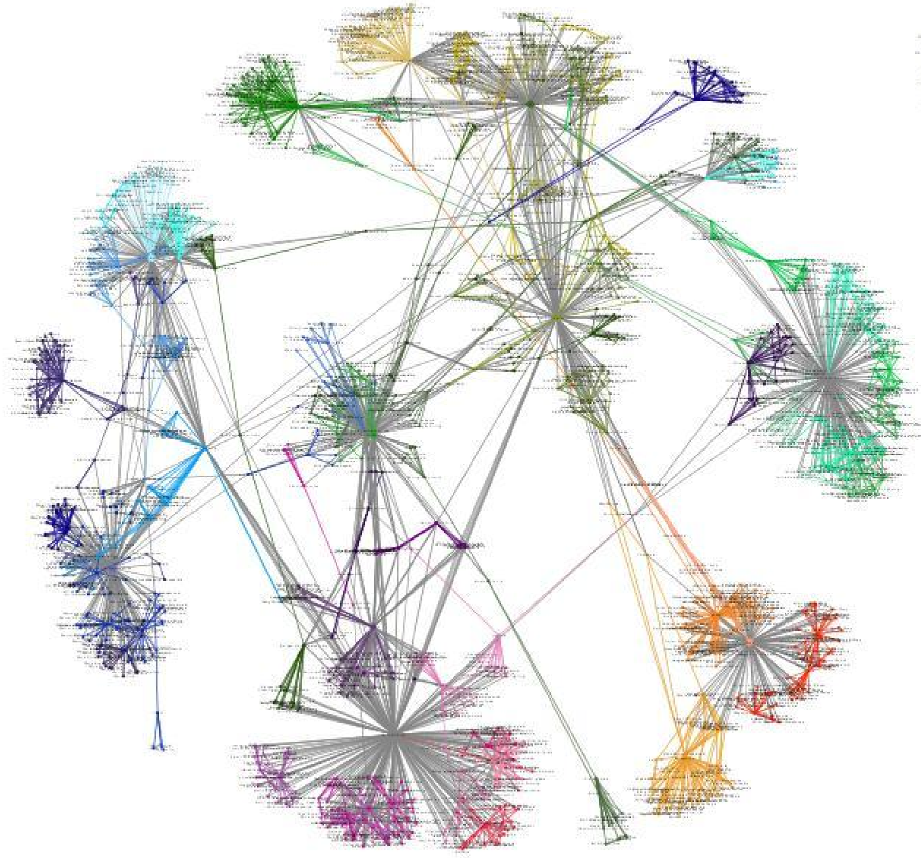
It is possible to generate a taxonomy from various semantic models (such as topic models), with **no external taxonomy or knowledge base**

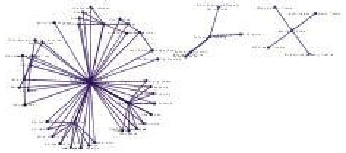
Statistical co-occurrence

Semantic similarity

Flexibility

(Treeratpituk, Khabsa and Giles, 2013)





Graph Partitioning

The crux of the generation is the utilization of multistage graph partitioning, as published by **(Karpis and Kumar, 1999).**

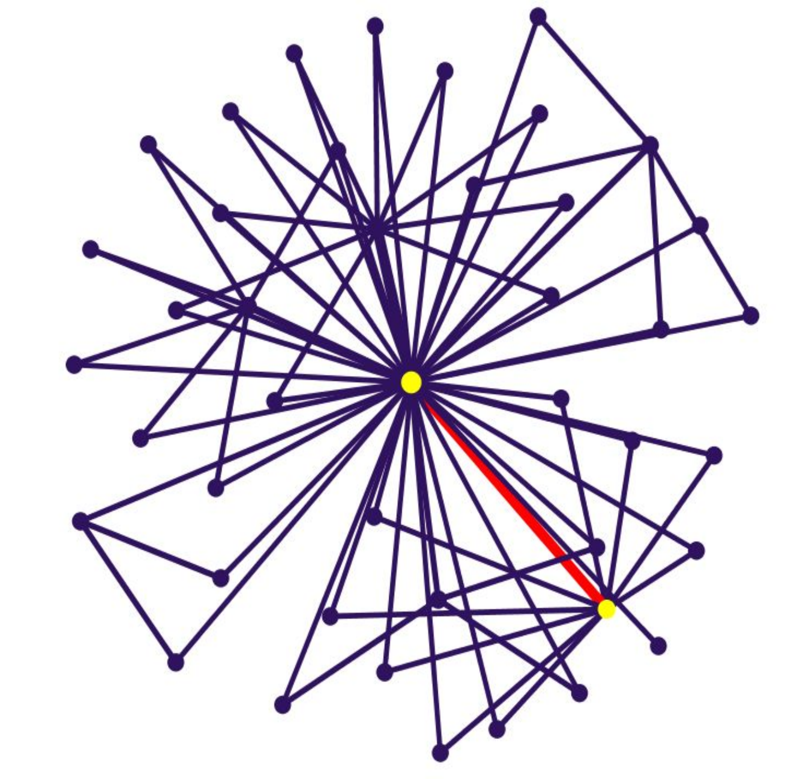
Multistage Graph Partitioning

Edge and vertex weighted graph

Recursive Partition

1. Coarsen. Match vertices and collapse.
2. Bisect
3. Uncoarsen

(Karpis and Kumar, 1999)



Heavy-Edge Matching

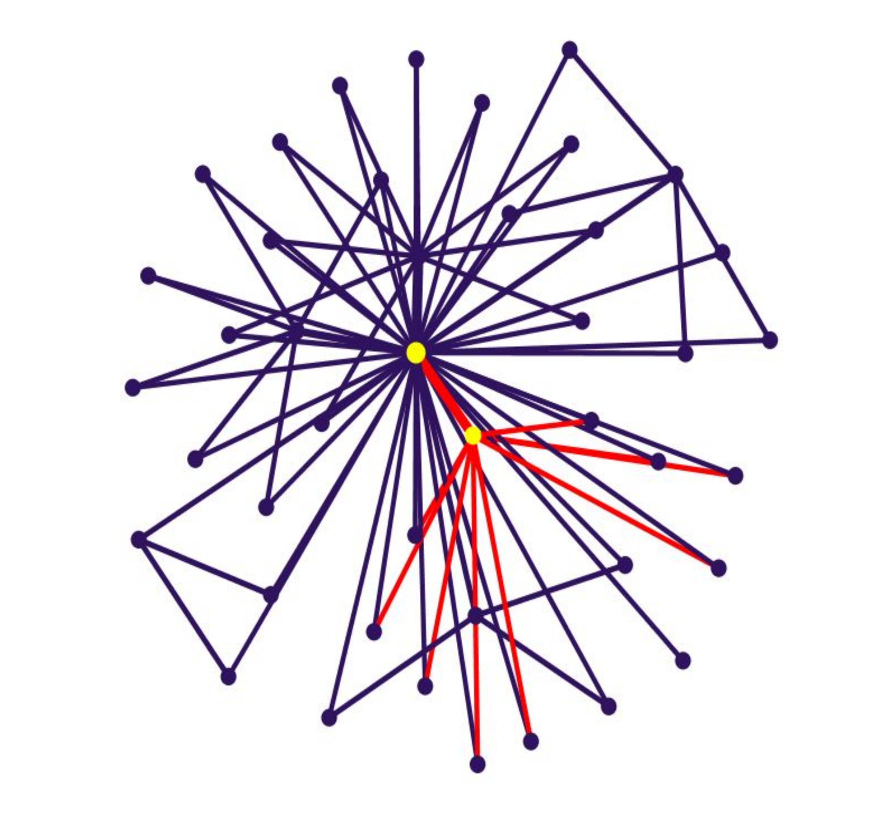
- ➔ **Visit vertex**
  Randomly visit vertices
- ➔ **Match**
  If the vertex is unmatched, select the unmatched neighbor with the highest edge weight.

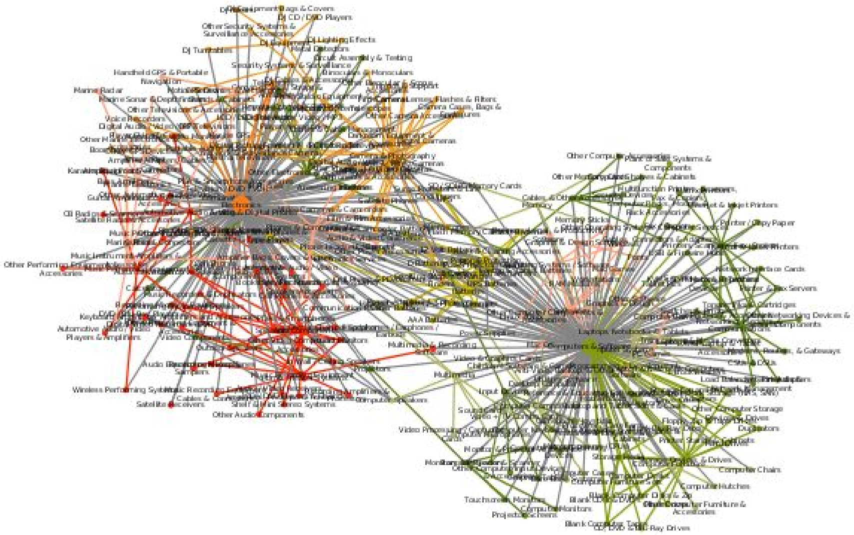
Collapse

➔ **Collapse the matched vertices**
Creation of the multi-node -- weight equals sum of the matching

➔ **Maximize edge weight**
Any collapsed edges are reweighted -- weight equals sum of the matching

➔ **Minimize edge-cut**
By maximizing edge-weight, coarser graph is lighter





Minimize edge cut

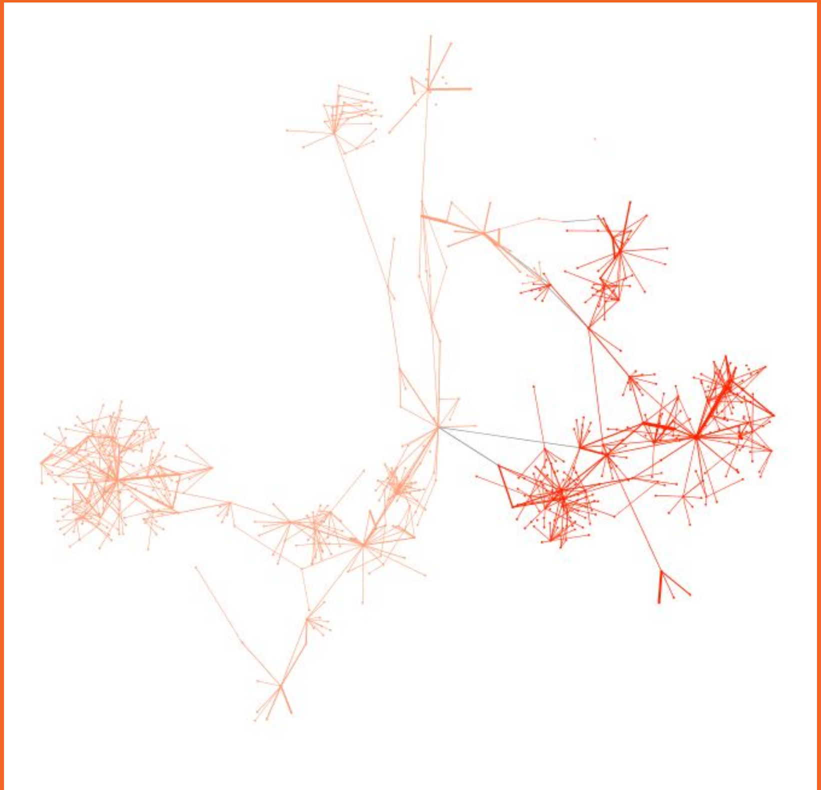
Edge cut: the number of incident edges which belong to different partitions

Maximal matching: stop collapsing when any edge not in the matching has at least one of its endpoints matched

Graph Bisection

Bisect coarsened graph, minimizing edge-cut.

Each part should contain roughly half the vertex-weight.



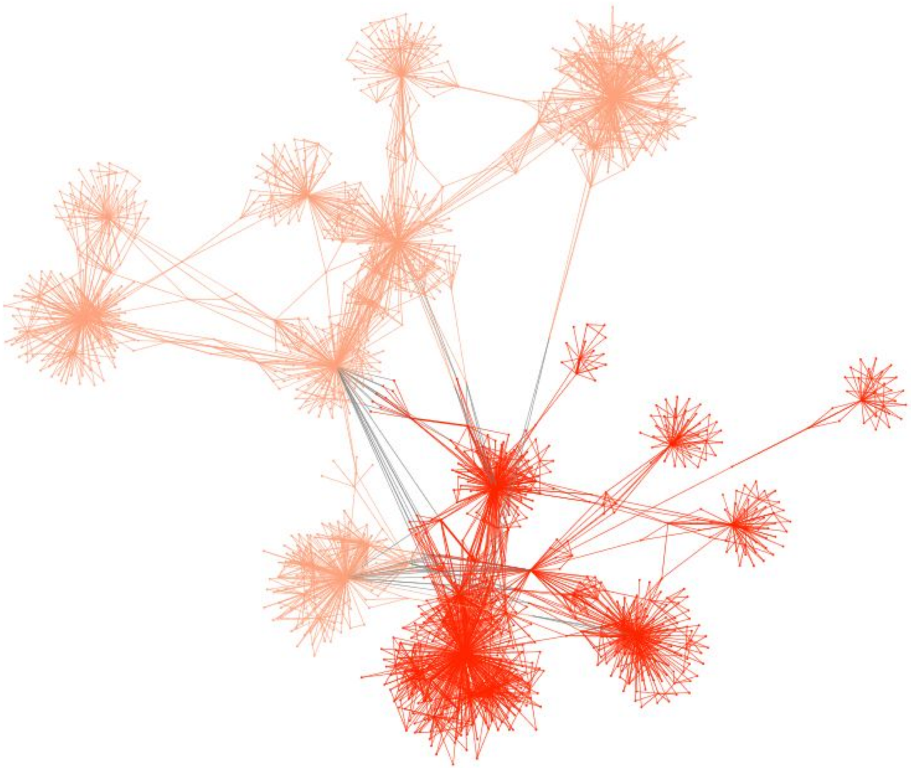
Kernighan-Lin

Iterative process

Initial bipartition is optimized by swapping vertices between them that minimize edge-cut

Terminates when no such subset can be located -- local minimum found

Repeat with other, random initial bipartations and choose the one with lowest edge-cut. Stop when derivative is zero for X iterations (modified KL).



Uncoarsening

Un-collapse the multinodes and edges

Project partitions back onto original graph

Refinement

Each partition represents a local minimum of the coarser graph

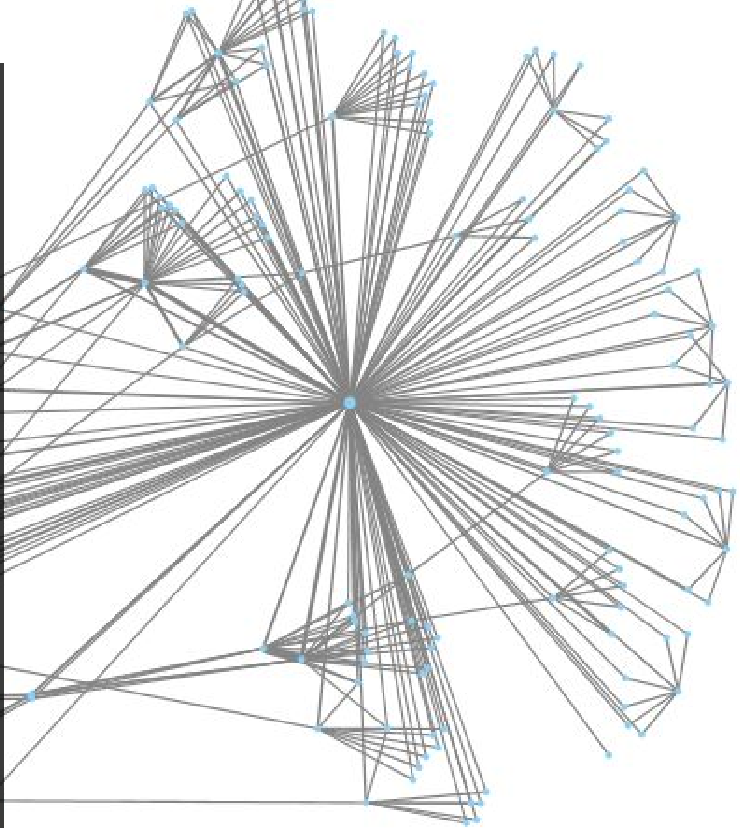
May no longer be a local minimum after refinement -- more information now exists

➔ **Refinement algorithm**
Karypis and Kumar use a refinement algorithm based on KL bisection

➔ **Project and compare**
After projecting the partition back onto the uncoarsened graph, re-run KL partition on the projected partition until convergence.

Building the initial graph

(Treeratpituk et al., 2013) devised a novel way to convert co-occurrence and similarity (across topics, etc.) into an edge and vertex weighted graph



GraBTax Process

Construct Association Graph

From LDA, etc.

Subgraph

Partition

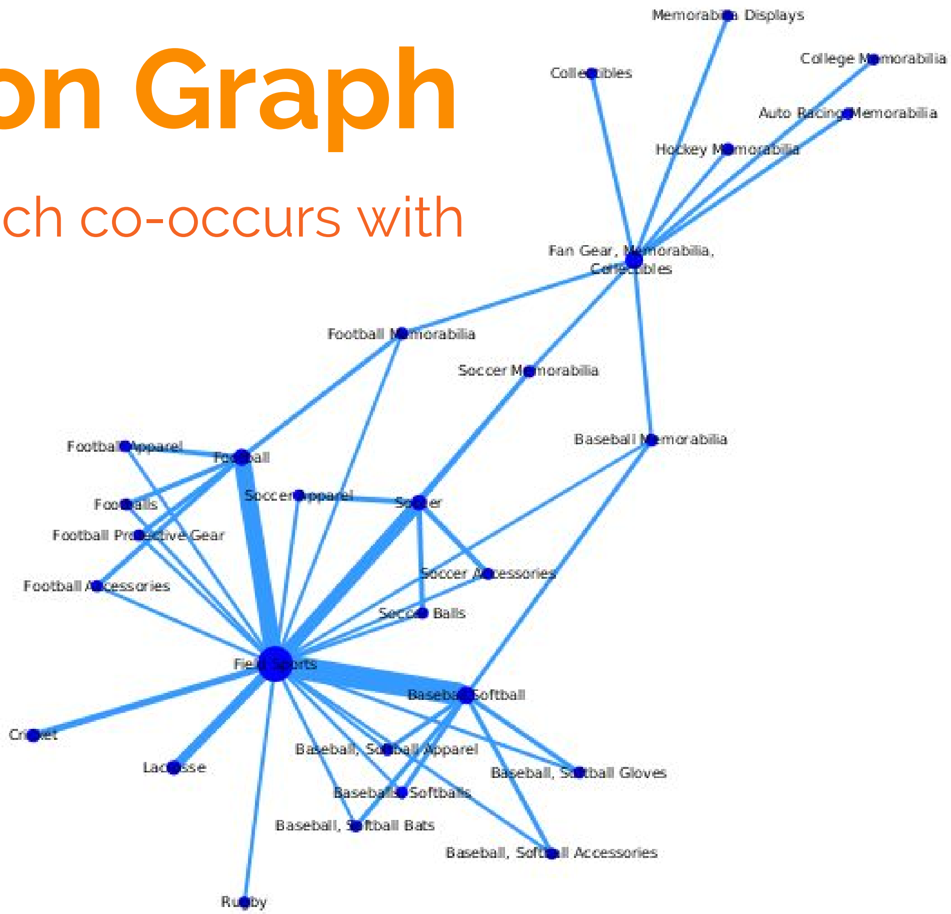
Topic Association Graph

Vertices exist for every topic which co-occurs with another topic at least once.

Topic weight = count of documents where t_i & t_j co-occur

$$s_i = \sum_{e_{ij} \in E} count(t_i, t_j)$$

(Treeratpituk, et al., 2013)



Topic Association Graph

Edge-weight is a function of the co-occurrence between t_i and t_j as well as their similarity.

$$w_{ij} = [1 + \lambda_1 \, {}^{1}(\mathit{rank}(t_i|t_j)=1 \text{ OR } \mathit{rank}(t_j|t_i)=1) + \lambda_2 \, \mathit{jac}(t_i, t_j)]$$

$$\times \, \mathit{count}(t_i, t_j)$$

where $1_{cond} = 1$ if *cond* is true, and 0 otherwise

$\mathit{rank}(ti\,|tj) = |\{ t_h \mid s_j < s_h \text{ and } P(t_h|t_j) > P(t_i|t_j) \}| + 1$

$\mathit{jac}(t_i, t_j)$ = Jaccard similarity between t_i and t_j

(Treeratpituk, et al., 2013)

It's no surprise Marcos uses Google Translate in his shop regularly.

There are 23 officially recognized languages in the EU.

Subgraph Selection

Select the vertices from which we will generate our final taxonomy

Lots of dials to turn

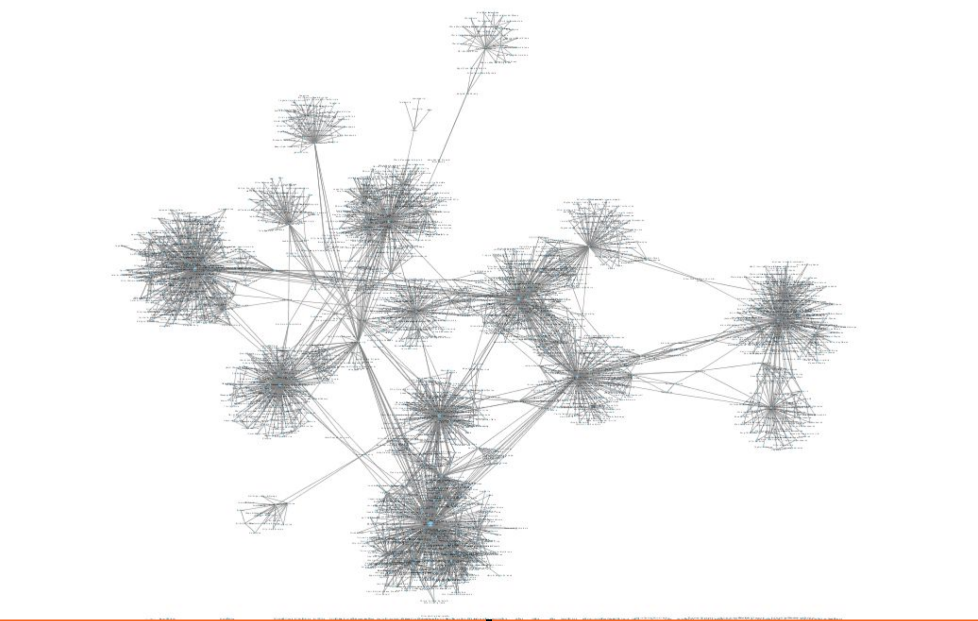
Begin with a query vertex t_0

From the query vertex, calculate a subgraph

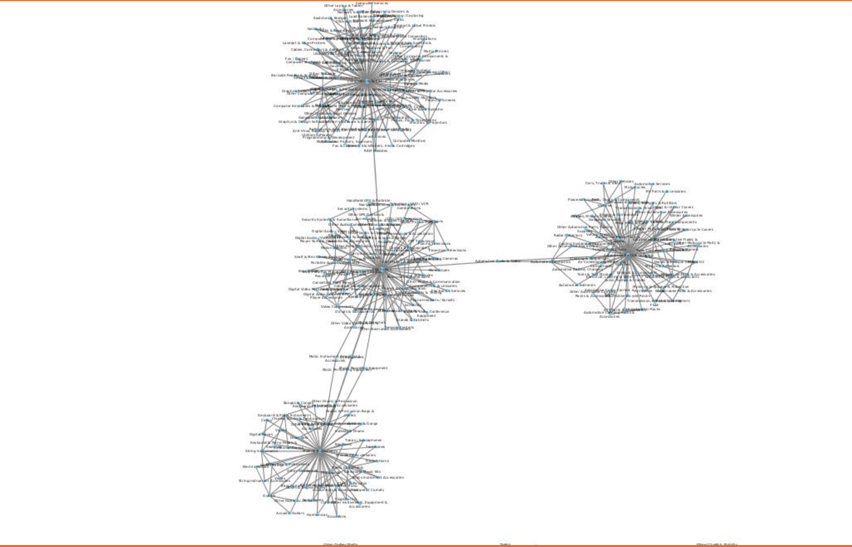
Subgraph vertices must be:

$\mathit{rank}(t_0,t_i) \leq r_{max}$

$k_i \geq k_{min}$ and $s_i \geq s_{min}$



Subgraph Example

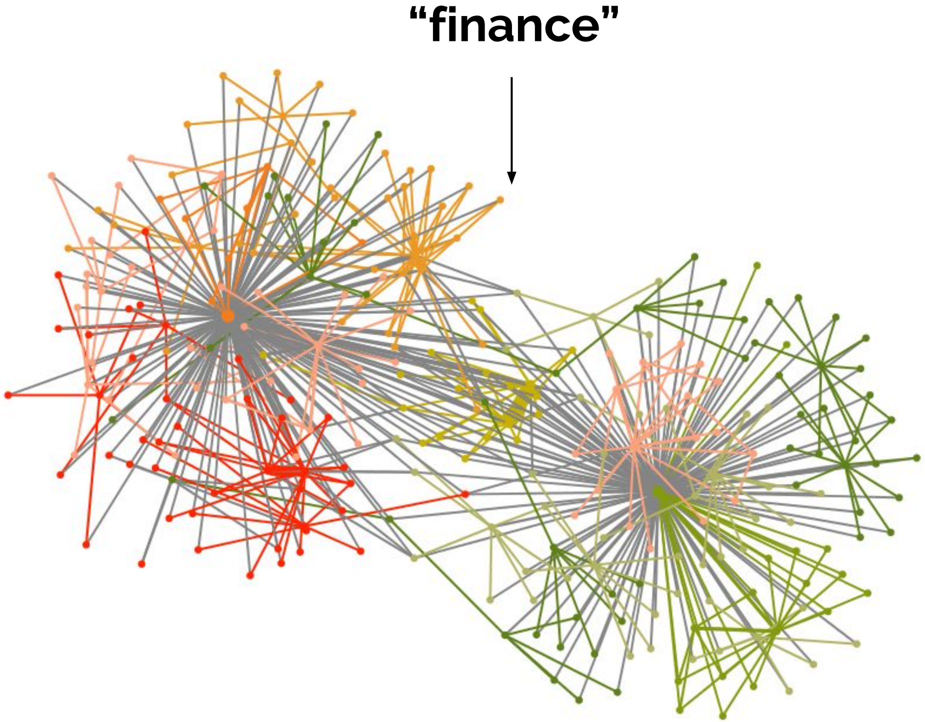


Partition and Select Labels

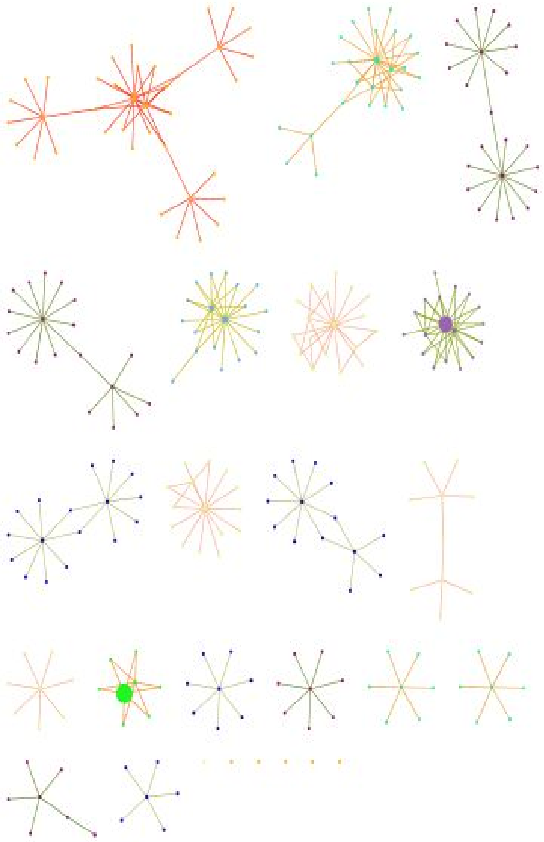
K-way partition of subgraph

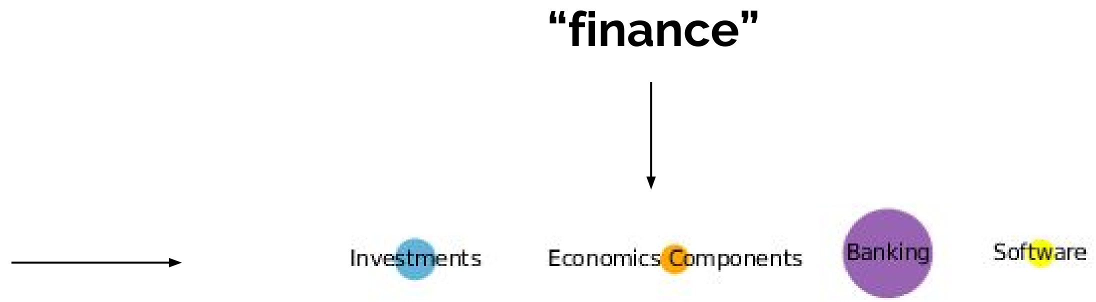
Within each partition, select the node with the highest degree of connectedness

These becomes the labels for the root level of the taxonomy.



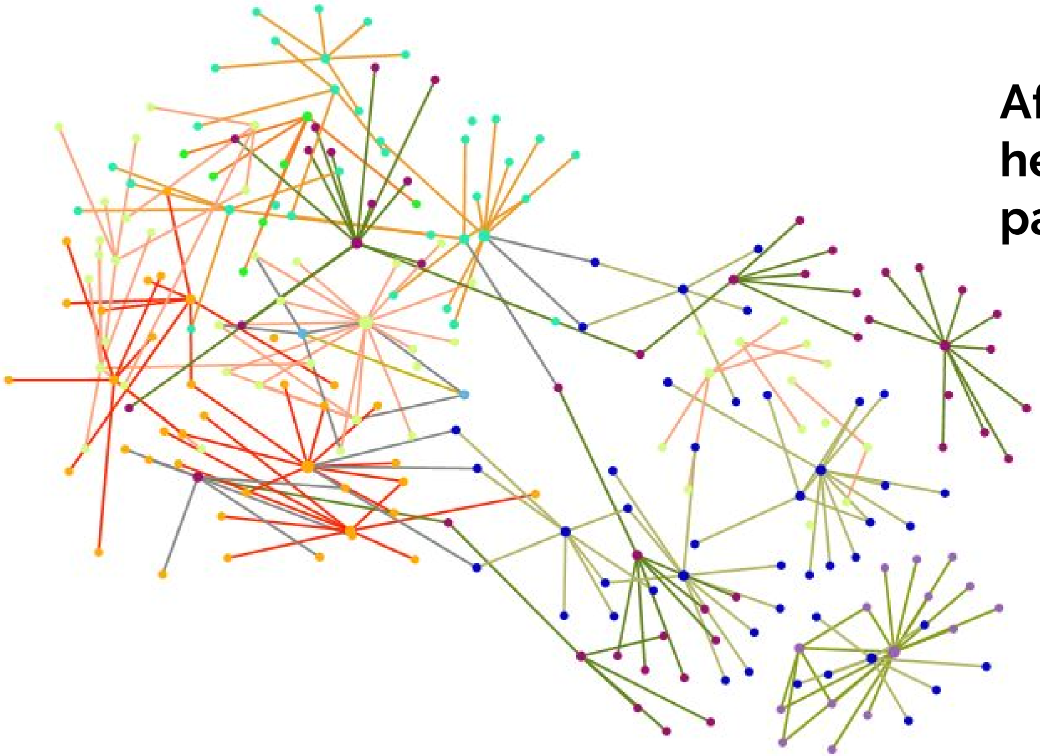
First partitioning



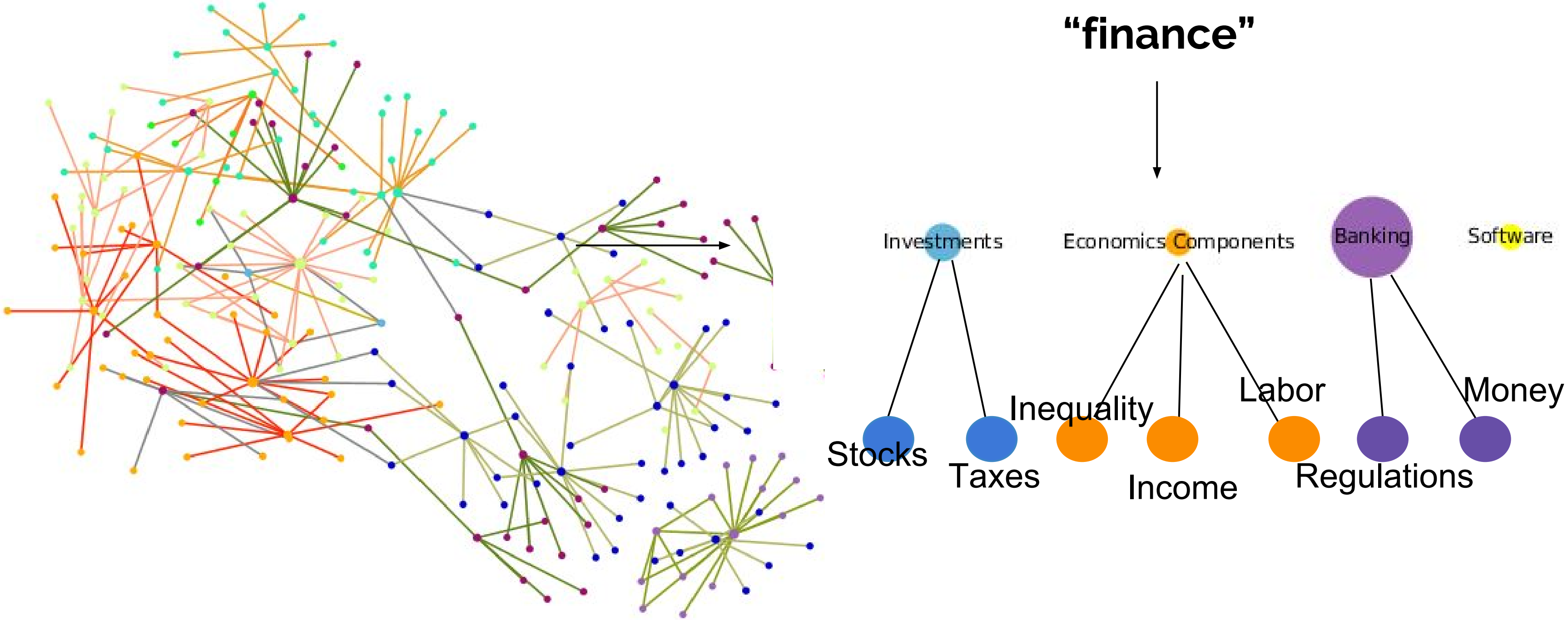


Select heaviest vertices from each partition

After removing the heaviest from each partition



second partitioning



Again, select heaviest vertices from each partition -- this forms the second level

Source Code

https://github.com/Lingistic/GraBTax

Source code for the GrabTax algorithm, including the recursive partition and selection code is up on github -- along with some examples.

This is still a work in progress, so give it a follow and check back later!

Contact

rob@lingistic.com
robmcdan@gmail.com
https://www.linkedin.com/in/robmcdan/