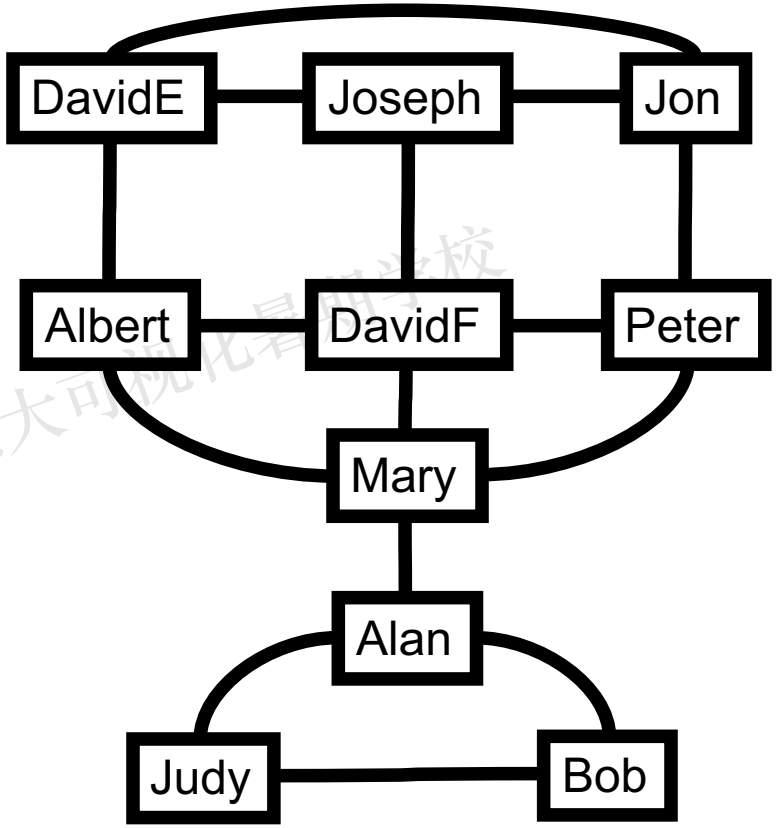
Graph Visualization



Yesterday

- Two classical approaches to graph visualization:
    1. Topology-shape-metrics
    2. Energy-based methods, slow

Today

- Faster energy-based methods
- Some recent research directions
    1. Measuring faithfulness, shape-based metrics
    2. RAC graphs
- Some final remarks

Yesterday's conclusion about metro maps:

- Energy based methods for metro maps "do not scale"
  - Runtime is too large
  - Some tangles in some maps

Elegance	Yes!
Effectiveness	Maybe
Efficiency	No

Poor performance when data size is large

There are two facets of the scale problem:

1. Computational complexity

- *Efficiency*
- Runtime
  - We need more efficient algorithms

2. Visual complexity

- *Effectiveness*
- Readability and faithfulness
  - We better quality drawings

FADE

Problem: Simple spring methods are too slow for huge graphs

Computing the forces takes quadratic time

1. p_u = some initial position for each node u;
2. Repeat
   - 2.1 $F_u := 0$ for each node u;
   - 2.2 For each pair u,v of nodes
     - 2.2.1 calculate the force f_{uv} between u and v;
     - 2.2.2 F_u += f_{uv};
     - 2.2.3 F_v += f_{uv};
   - 2.3 For each node u, p_u += εF_u;

   Until p_u converges for all u;

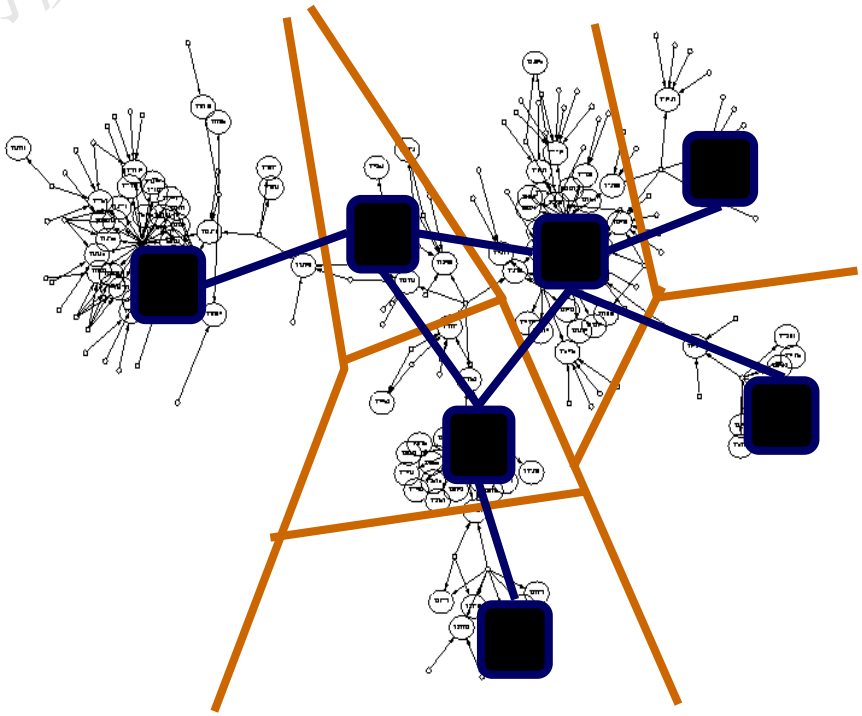
A general *multilevel* approach

We can use

1. A spring method, and
2. A clustering method to give a cluster hierarchy for the graph
   - The clustering could be geometric or combinatorial

Use divide&conquer on the cluster hierarchy:

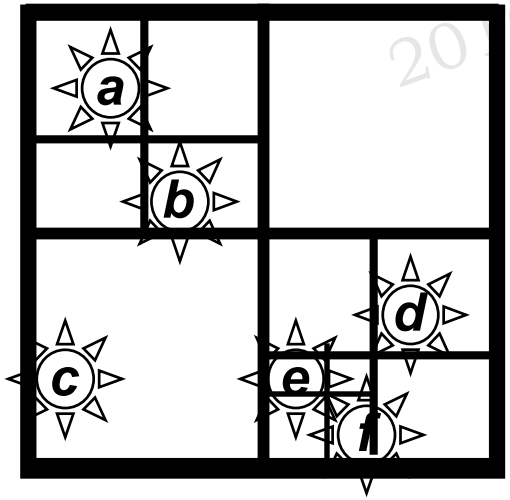
– First apply the spring method at a higher level in the hierarchy, then at a lower level

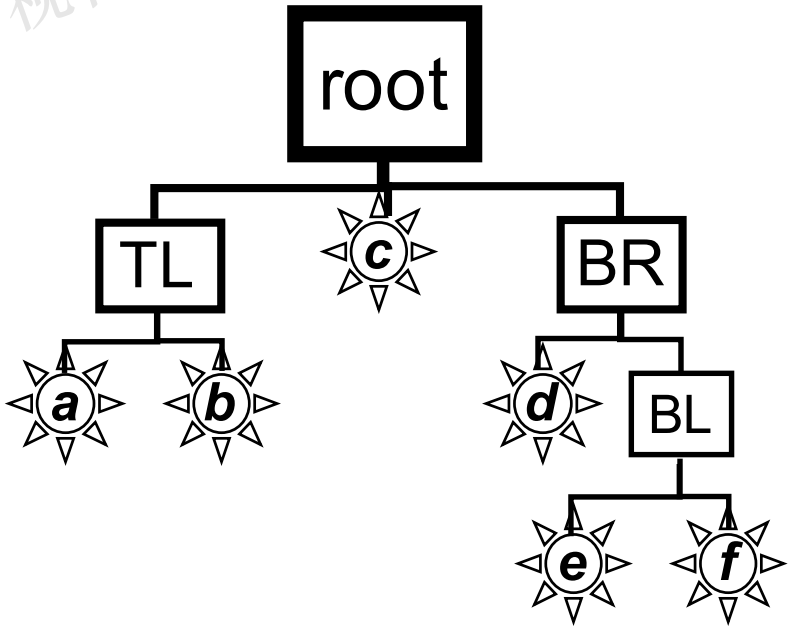


AT&T

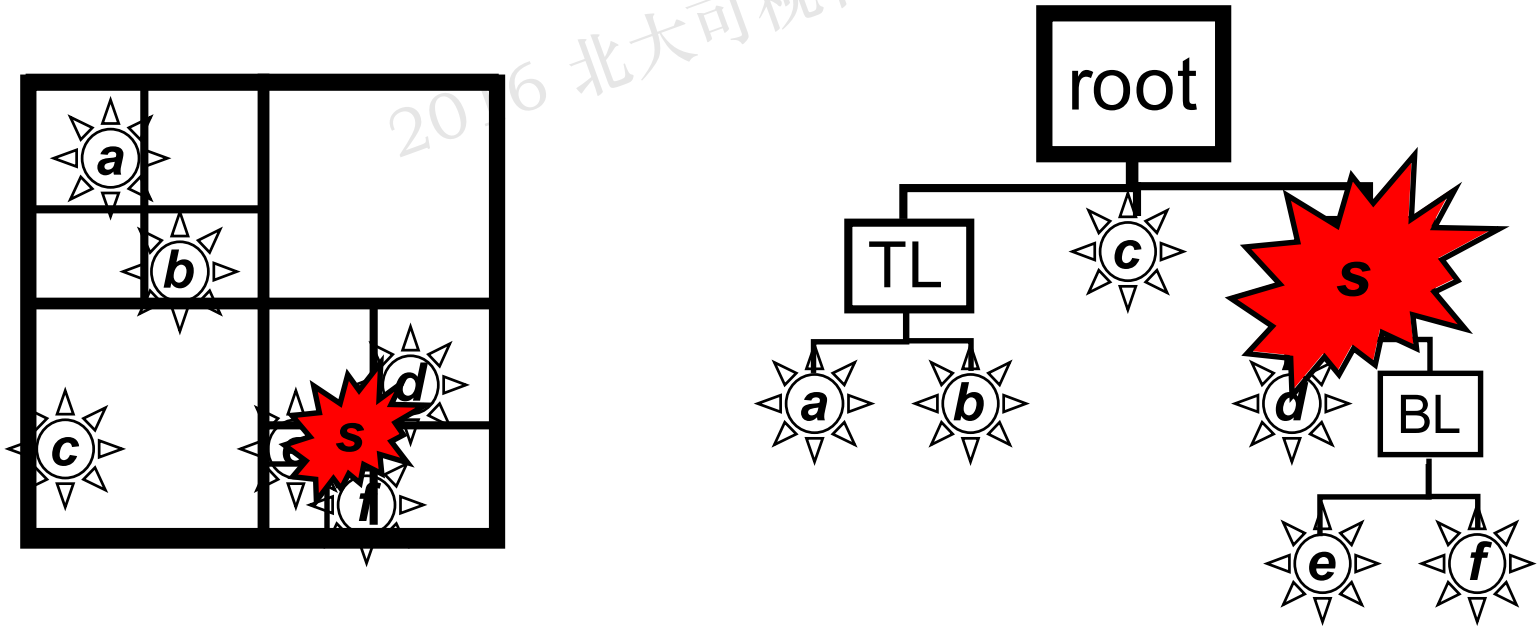
Barnes-Hutt method

- A method of computing configurations of stars.
  - Use a quadtree to cluster the stars
  - Use the forces between the clusters to approximate the forces between individual stars.

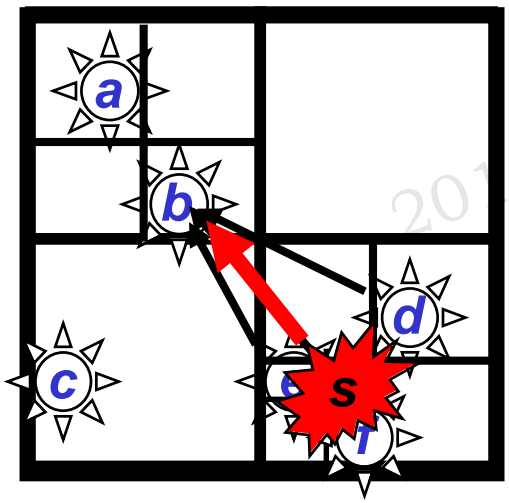


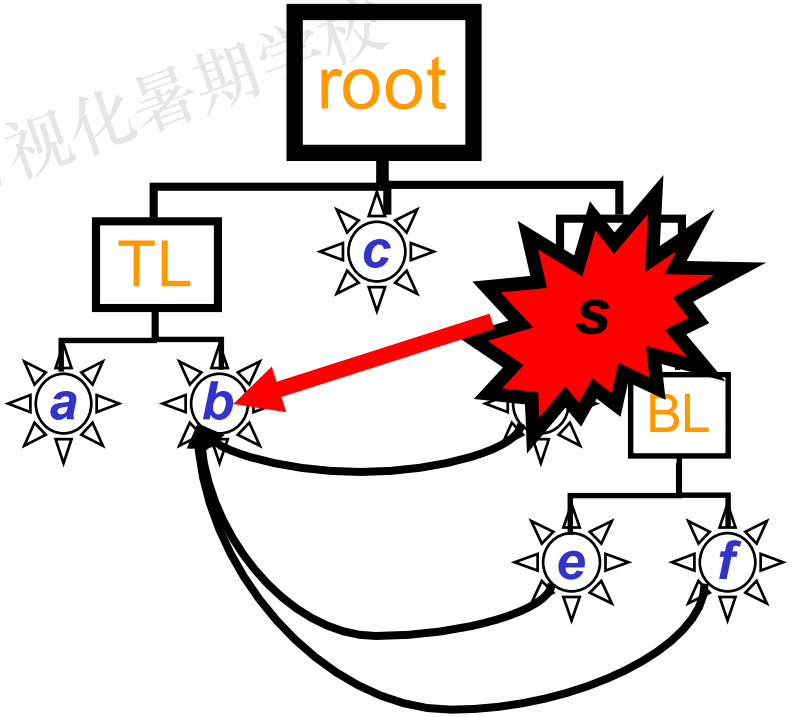


The contents of a subtree of can be approximated by a mass at the centroid.



The force that the subtree ***s*** exerts on the star ***b*** can approximate the sum of the forces that the nodes in ***s*** exert on ***b***.





To compute the force on star $\boldsymbol{x}$, we proceed from the root toward the leaves.

***ComputeForce*(star x; treenode t)**
If the approximation is good
then return the approximation;
else return $\sum_s$ ***ComputeForce***(x, s), where the sum is over all children s of t.

A simple method can be used to determine whether the approximation is good; it depends on the *mass of nodes* and the *distance* between $\boldsymbol{x}$ and $\boldsymbol{s}$.

$$\frac{w(t)}{d(x,t)} < c,$$

where $\boldsymbol{w(t)}$ is the width of $\boldsymbol{t}$; $\boldsymbol{d(x,t)}$ is the distance between $\boldsymbol{x}$ and $\boldsymbol{t}$, and $\boldsymbol{c}$ is a constant.

The Barnes-Hutt method is much faster than the usual spring algorithm.

1. $\boldsymbol{p_x}$ = some initial position for each star $\boldsymbol{x}$;
2. Repeat
   - 2.1 Build the quadtree;
   - 2.2 Foreach star $\boldsymbol{x}$
     - **ComputeForce**$(\boldsymbol{x}, \boldsymbol{root})$;
   - 2.3 Foreach star $\boldsymbol{x}$, $\boldsymbol{p_x}$ += $\boldsymbol{\varepsilon F_x}$;

   Until $\boldsymbol{p_x}$ converges for all $\boldsymbol{x}$;

In practice, computing all the forces takes $\boldsymbol{O(n\,log\,n)}$ *time*

More generally, we can **multi-level methods**

1. Cluster to nodes (geometrically or graph-theoretically)
2. Compute and apply forces between the clusters
3. Compute and apply forces between nodes.

For example:

Multi-level Layout (*Graph* G)
If G is small, then use a simple spring method to find a drawing of G;
else
Cluster (or "coarsen") G to create a graph G' smaller than G;
Recursively call *Multi-Level-Layout*(G') to get a rough layout of G;
Apply forces to all of G to get a good layout of G.

- FADE: clustering is done by a quadtree.
- Other methods: clustering by voronoi diagram, edge contraction, random walk, …

Future of faster force directed methods:

- Simple spring method: $O(|E| + |V|^2)$ force calculations at each iteration.
- Current good methods reduce the $O(|V|^2)$ term
  - For example FADE: $O(|E| + |V| \log |V|)$)
- Near future might reduce the $O(|E|)$ term:
  - for example, perhaps Speilman-Teng sparsification could give a good algorithm in time $O(|V| \log |V|)$
- Long term future: we need $o(|V|)$