

Working Group: Navigation in Large Graphs

Hoi Ming Wong (hoi-ming.wong@tu-dortmund.de)

Ulrich Lauther (ulrich.lauther@t-online.de)

Nikola Nikolov (nikola.nikolov@ul.ie)

Lev Nachmanson (lev@microsoft.com)

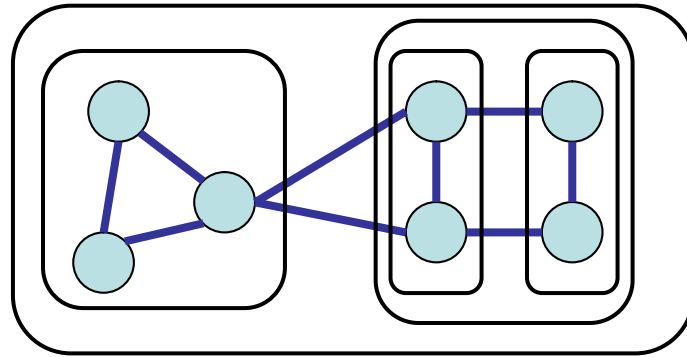
Georg Sander (sander@ilog.fr)

- Motivation

- Social Networks, in particular Web 2.0 graphs are usually very large
- Biochemical networks can grow very large
- Traditional layout algorithms (spring embedder, Sugiyama) are too slow
 - Spring embedder due to expensive force calculation and iteration
 - Sugiyama due to explosion of number of bends and crossing reduction
- Fast algorithms such as MDS do not produce satisfying results for all kinds of graphs

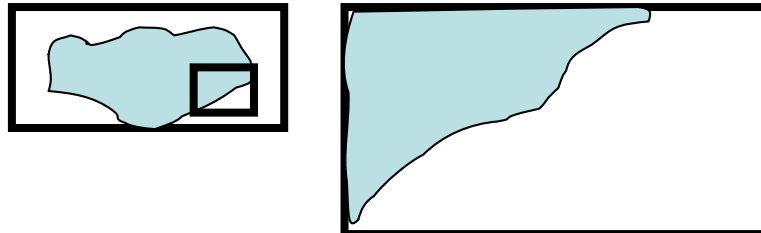
- Initial goals: How to explore large graphs
 - By clustering
 - By interactions/navigation
 - By transformation
- Collection of old and new ideas
 - Topological fisheyes
 - Filtering with thresholds of nodes and edges
 - Clustering
 - Based on info in data
 - Random clustering
 - Based on topology: clique, biconnected comp., ...
 - Based on cut-set (edges cutting clusters)
 - Based on backbone spanning tree

- Clustering:
 - Already covered by other groups
 - Lets assume the other groups invent meaningful multilevel clustering



- Navigation
 - Our main focus now

- Navigation with Overview:
 - Overview window shows overview
 - Main window shows detail view
 - Region in overview can be freely moved
 - Layout for main window is calculated on fly
 - Nodes that once got coordinates will keep these coordinates forever
 - Keep the mental map when showing region again
 - Principle of minimal surprise



- Don't show all
 - Overview shows only highest clustering level
 - Fast layout
 - Detail view shows sub-clustering level depending on zoom level
 - Coordinates in overview must (somehow) match coordinates in detail view
- Lazy layout on the fly
 - When shown first, do layout

- Framework Algorithm
 - Calculate coordinates for highest clustering level (if clusters are given by data, then use it)
 - Calculate Voronoi-Diagram for highest clustering level => area for the nodes of the corresponding cluster
 - Determine which clusters are inside the region of detailed window
 - Intersection with Voronoi regions

- while detail level not reached
 - Go down one cluster level
 - Calculate induced cluster graph at that level
 - Calculate coordinates so that nodes of cluster fit inside cluster (Voronoi) region
 - Calculate Voronoi diagram of inner cluster level
 - Determine which clusters are inside the region of detailed window
 - Intersection with Voronoi regions

Open Problems

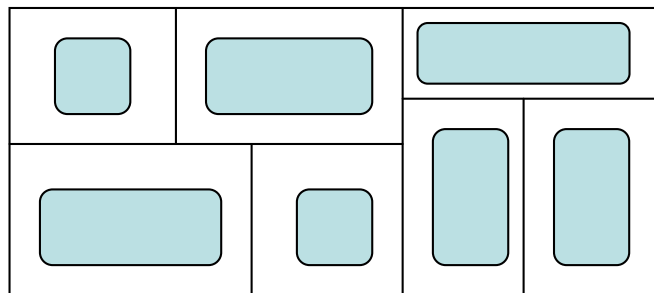
- Show/ hide edges, which nodes are not shown in the visible window
- Quality of the drawing? Global vs. on the fly
 - Edge crossings
 - Edge-node overlaps
 - Total edge length
- Voronoi diagram problems:
 - Voronoi diagram areas must be proportional to the size of the clusters
or
 - Scaling the partition of the drawing area after calculate the voronoi diagram
- Runtime analysis
- Suitable clustering algorithms
- Research in partial GD, interactive GD, navigation, exploration

Voronoi diagram problem:

- Area of Voronoi region proportional to cluster size
- Several definitions of weighted Voronoi diagrams exist
 - Additive weighted ...
 - Multiplicative weighted ...
 - Component weighted ...
- Does any of this has anything to do with area of Voronoi region? => need literature analysis

Instead of Spring layout & Voronoi Diagram

- Rectangular partition approach instead of layout
 - Recursive Bipartition based on cut set until partition contains only 1 node
 - Assign rectangular region proportional to partition size. Region \approx Voronoi Region



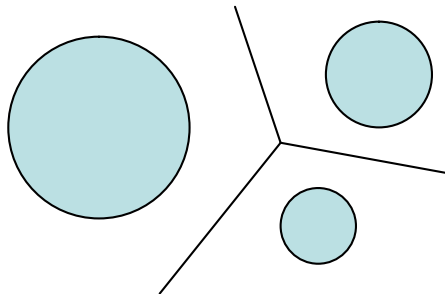
- Open question:
 - How to incorporate edges?
 - Does the layout looks nice?

Probably better: Combine Spring layout and Voronoi region adjustment

- Initial spring layout
- Calculate Voronoi region
- While Voronoi region size is not as desired
 - Calculate forces on nodes according to size mismatch of Voronoi region size
 - Adjust node positions
 - Recalculate Voronoi region (incrementally?)

Probably fastest: Use Spring Layout with large nodes & calculate Voronoi wrt. node border

- Idea:
 - percentage of “waste area” between nodes is small
 - Voronoi partitions only the waste area
 - $\text{Size}(\text{Voronoi region}) = \text{size}(\text{node}) + \text{part}(\text{waste area})$
 - Error of size of Voronoi region is neglectable
- Only an approximation, but needs no recursive or iterative recalculation of Voronoi diagram



Further plans:

- Elaborate details
- Implementation?, Paper?

Comments from the audience:

- Look at the Voronoi online game
 - Might handle the area size problem