Graph Drawing: From algorithms to tools to specific use case

Zhiying Jiang • 08.04.2021

Overview

Introduction

Graph and Graph Drawing

Graph Layout Algorithms

- Trees
- DAG
- General Graph

DAG Visualization Tools

- Software Based
- Web App Based

Use Case - Concept Map

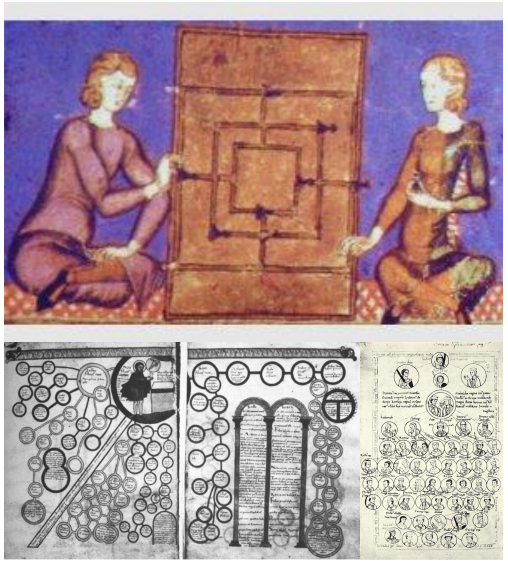
- Requirement Analysis
- Implementation and Evaluation

Introduction

Introduction

History

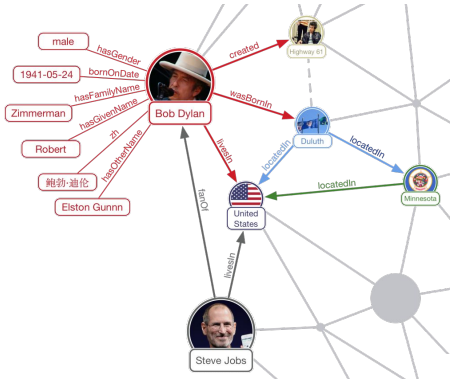
- Noli turbare circulos meos! (Do not disturb my circles!)
- Mill or Morris games
- The Middle Ages Family Tree

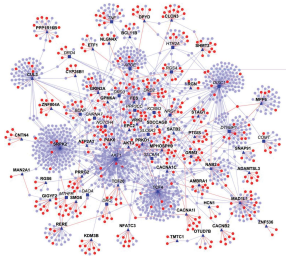


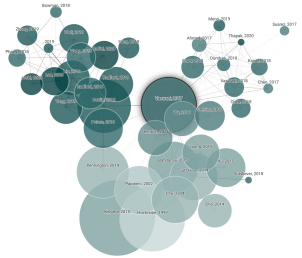
Introduction

Nowadays

- Social Networks
- Knowledge Graph
- Protein Protein Interaction
- Citation Network







Graph and Graph Drawing

Graph and Graph Drawing

Graph

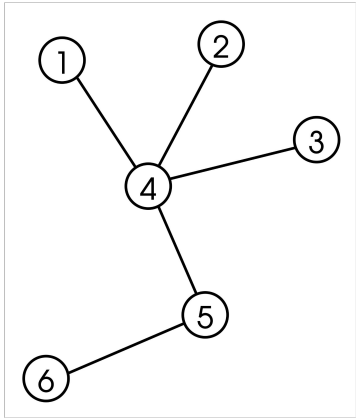
- graph G=(V,E) is defined as the pair of vertices V and edges E, and any edge in E connects either two vertices in V

Graph Drawing

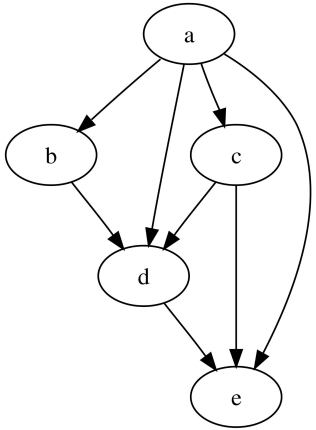
- it's defined as mapping d that satisfies d: G-> d(G), where d(G) in R^2 (or even R^3). Specifically, this mapping d assigns coordinates to the nodes and the bends of edges.

Graph and Graph Drawing

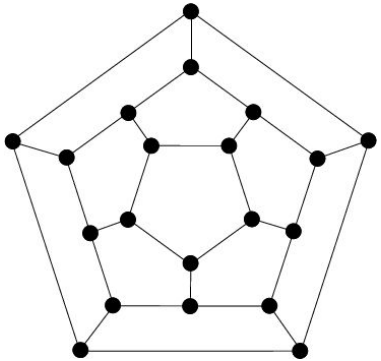
Types



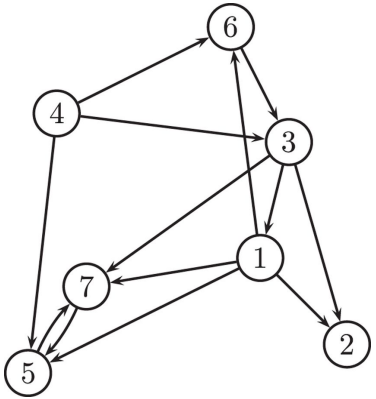
Tree



DAG



Planar Graph



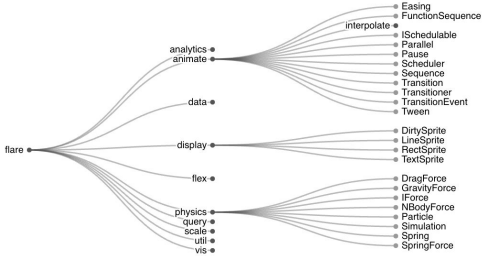
General Graph

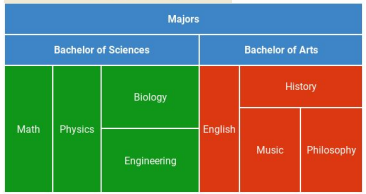
Graph Layout Algorithms

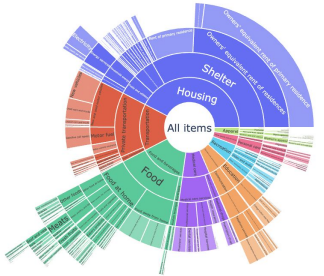
Graph Layout Algorithms

Tree (Diagram Types)

- Indentation
- Node-Link diagrams
- Enclosure diagrams
- Layered Diagrams







Graph Layout Algorithms

Tree (Knuth's)

- Do a top-down inorder traversal of a tree, with the depth as its y value and the global counter as its x value.

```
i = 0
def knuth_layout(tree, depth):
    if tree.left_child:
        knuth_layout(tree.left_child, depth+1)
    tree.x = i
    tree.y = depth
    i += 1
    if tree.right_child:
        knuth_layout(tree.right_child, depth+1)
```

Principles:

1. No crossed edges.
2. Nodes at the same level/depth should be placed on the same horizontal line (same y value).

Graph Layout Algorithms

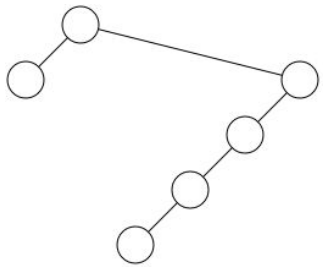
Tree (Knuth's)

- Do a top-down inorder traversal of a tree, with the depth as its y value and the global counter as its x value.

```
i = 0
def knuth_layout(tree, depth):
    if tree.left_child:
        knuth_layout(tree.left_child, depth+1)
    tree.x = i
    tree.y = depth
    i += 1
    if tree.right_child:
        knuth_layout(tree.right_child, depth+1)
```

What's missing?

- Might be:



Graph Layout Algorithms

Tree (Wetherell's)

- Do a bottom-up algorithm that keeps track of next slot on each row and then traverse the tree in postorder

Principles:

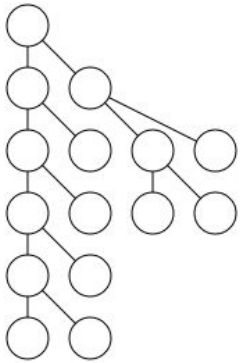
1. No crossed edges.
2. Nodes at the same level/depth should be placed on the same horizontal line (same y value).
3. Place the whole graph into minimum width

Graph Layout Algorithms

Tree (Wetherell's)

- Do a bottom-up algorithm that keeps track of next slot on each row and then traverse the tree in postorder

What's missing?



Graph Layout Algorithms

Tree (Reingold-Tilford's)

- Start with bottom-up pass of the tree. The initialization of x coordinate is arbitrary and y coordinate is by depth; then merge left and right subtrees by shifting right and finally do a top-down pass for assignment of final positions. global counter as its x value.

Additional Principles:

4. Parent should be centered above its children

5. Isomorphic subtrees should be drawn identically

Graph Layout Algorithms

Tree (Reingold-Tilford's)

Algorithm 1: Reingold-Tilford Algorithms

Do post-order traversal of the tree
if *node v is a leaf* **then**
  $v.x = 0$;
else
  Place the right subtree of v as close to v as possible;
  Record how many Δx to move
end
Place the node halfway between its children.
A second top-down pass to accumulate Δx for the final assignment.

Additional Principles:

4. Parent should be centered above its children

5. Isomorphic subtrees should be drawn identically

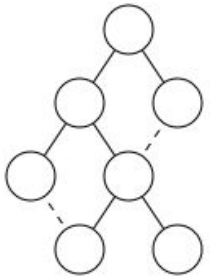
Graph Layout Algorithms

Tree (Reingold-Tilford's)

Algorithm 1: Reingold-Tilford Algorithms

Do post-order traversal of the tree
if *node v is a leaf* **then**
  $v.x = 0$;
else
  Place the right subtree of v as close to v as possible;
  Record how many Δx to move
end
Place the node halfway between its children.
A second top-down pass to accumulate Δx for the final assignment.

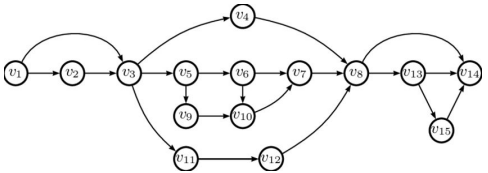
Finally: 👏

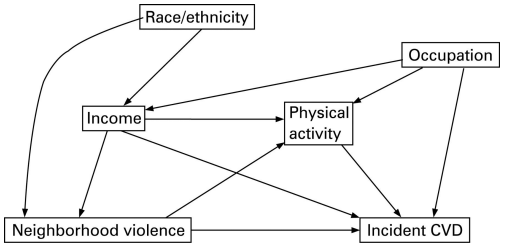


Graph Layout Algorithms

Directed Acyclic Graph (DAG)

- Directed graphs with no cycles
- If and only if can be ordered topologically, by arranging the vertices as a linear ordering that is consistent with all edge directions
- Comparing to trees, allow multiple parents

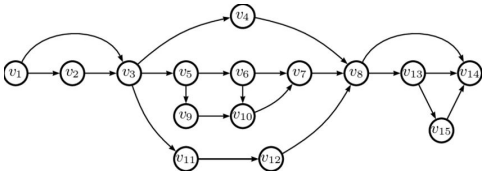


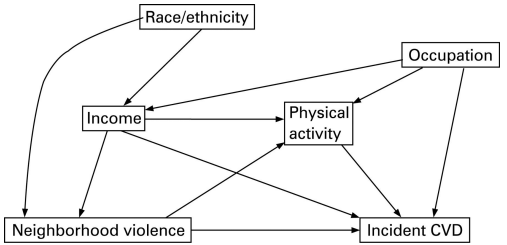


Graph Layout Algorithms

Directed Acyclic Graph (DAG)

- Directed graphs with no cycles
- If and only if can be ordered topologically, by arranging the vertices as a linear ordering that is consistent with all edge directions
- Comparing to trees, allow multiple parents





Graph Layout Algorithms

DAG (Sugiyama's method) - Principles

- Edges should point in a uniform direction
- Short edges are more readable
- Nodes should be distributed uniformly to avoid clutter
- Edge crossings should be minimized
- Straight edges are more readable

Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

Algorithm 2: Sugiyama methods in a high level

Cycles Removal;
Calculate layering;
Crossing Reduction;
Routing of the edges;

Graph Layout Algorithms

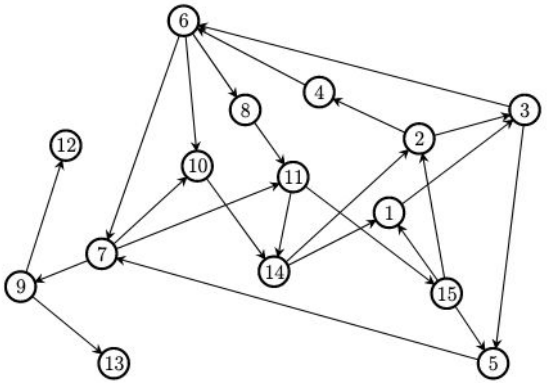
DAG (Sugiyama's method) - High Level Algorithm

- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;

Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

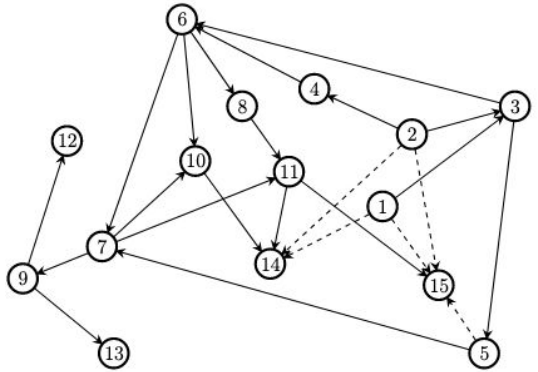
- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;



Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

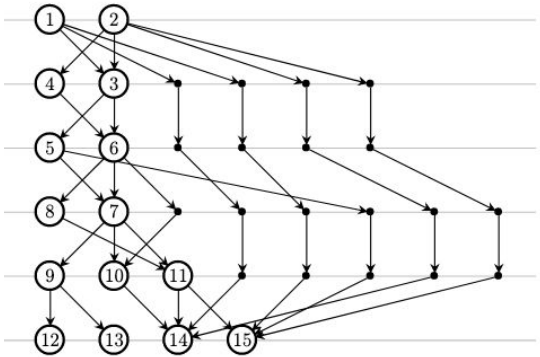
- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;



Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

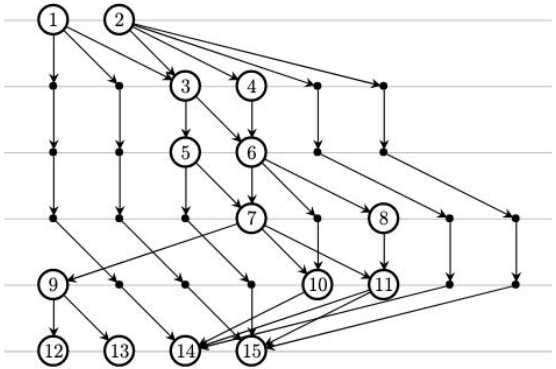
- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;



Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

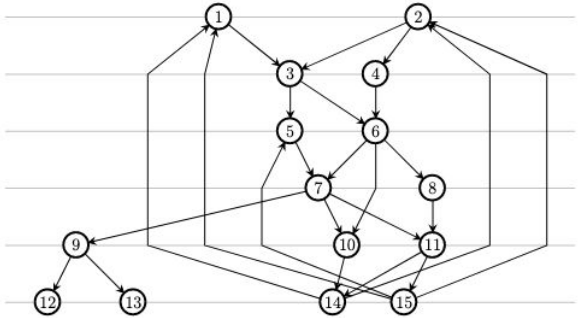
- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;



Graph Layout Algorithms

DAG (Sugiyama's method) - High Level Algorithm

- Cycles Removal;
- Calculate layering;
- Crossing Reduction;
- Routing of the edges;



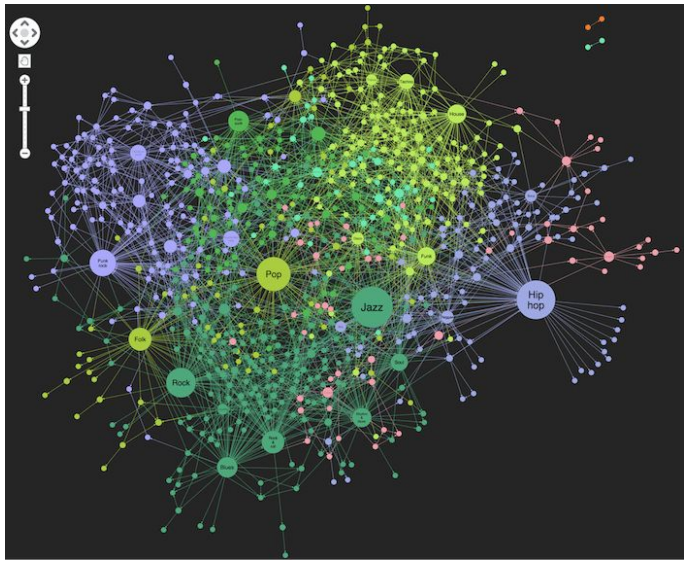
Graph Layout Algorithms

General Graph

- No explicit hierarchy
- Often large scale

The Principle

- Minimizing crossing edges



Graph Layout Algorithms

General Graph (Force Directed Method)

- Also called “Spring Embedder”
- Assign “force” to pair of nodes
- Treat graph layout optimization as a “physical system” simulation problem
- General framework

Graph Layout Algorithms

General Graph (Force Directed Method)

Basic Version:

Algorithm 3: Force Directed Method

```
foreach node v do
    foreach pair of nodes (u, v) do
        compute repulsive force f_r(u, v)
    foreach edge e = (u, v) do
        compute attractive force f_a(u, v)
    sum over all force vectors on v → F(v) ;
    move node v according to F(v)
```

|Attractive force|:
c1*log(d/c2)

|Repulsive force|:
c3/d^2

Graph Layout Algorithms

Aesthetics

- Visual complexity --- how easy it is to get an overview
- Regularity --- repetitions like if isomorphic graphs look the same
- Symmetry --- geometric symmetry by rotation, reflection and translation
- Consistence --- if showing similar patterns indicates similar meaning
- Form, size and proportionality --- size and ``node density and sparsity"
- Algorithms' time complexity --- how long does algorithm take to run

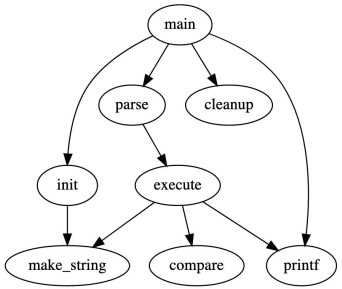
DAG Visualization Tools

DAG Visualization Tools

GraphViz 

- DOT language
  - Simple & Intuitive
  - Adopted by many other applications
  - Support any output format like GIF, PNG, SVG, PDF or PostScript
- Sugiyama's method for directed graph

```
digraph G {
    main -> parse -> execute;
    main -> init;
    main -> cleanup;
    execute -> make_string;
    execute -> printf
    init -> make_string;
    main -> printf;
    execute -> compare
}
```



DAG Visualization Tools

GraphViz 

- CLI

```
dot -Tsvg input.dot
```

  - dot for directed graph
  - neato for undirected graph

- Web

```
<img src=/cgi-bin/webdot/input1.dot.neato.png>
```

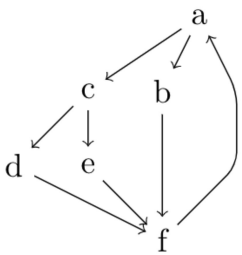
  - webdot HTTP server

DAG Visualization Tools

TikZ/PGF

- For TeX
- Basically can draw anything
- All graph layout algorithms require LuaTeX compiler
    - You can also implement your own algorithm in Lua
- Intuitive syntax

```
\tikz [rounded corners]
  \graph [layered output, sibling distance=8mm, level distance=8mm]
  {
    a -> {
        b,
        c -> {d, e}
    } ->
    f ->
    a
  }
```



DAG Visualization Tools

d3dagre 

- d3 + dagre
- dagre is a specific “flavor” of Sugiyama’s method
- Take full advantage of d3’s flexibility without writing layout algorithm
- Steep learning curve
- SVG

DAG Visualization Tools

Cytoscape.js 

- Origin from software, aiming at analyzing large scale bio-medical data
- Specialized for graph, no other charts
- Number of graph analysis functions implemented, like minimum spanning tree, clustering, etc.
- Have the most graph layout algorithms
- Canvas

DAG Visualization Tools

GoJS 

- Model-View architecture to separate data and style
- Limited pre-built layout algorithm
- Mature commercialized data visualization library
- Easy-to-use API
- Canvas

My Use Case - Concept Map

Concept Map

Intro

- Node? - Concept
- Concept? - Subject concepts like “bayes’ rule”, “gradient descent”
- Edge? - Prerequisite relation
- Scale? - About 500 concepts, 1000 relations

Concept Map

Requirement Analysis

- Cannot fit in one screen - Zoom in/out function required
- Too many nodes to see prerequisite nodes clearly - Highlight prerequisite nodes and relations required
- Nodes need to display text directly - text wrapper or auto-fitting to the shape or other style manipulation required

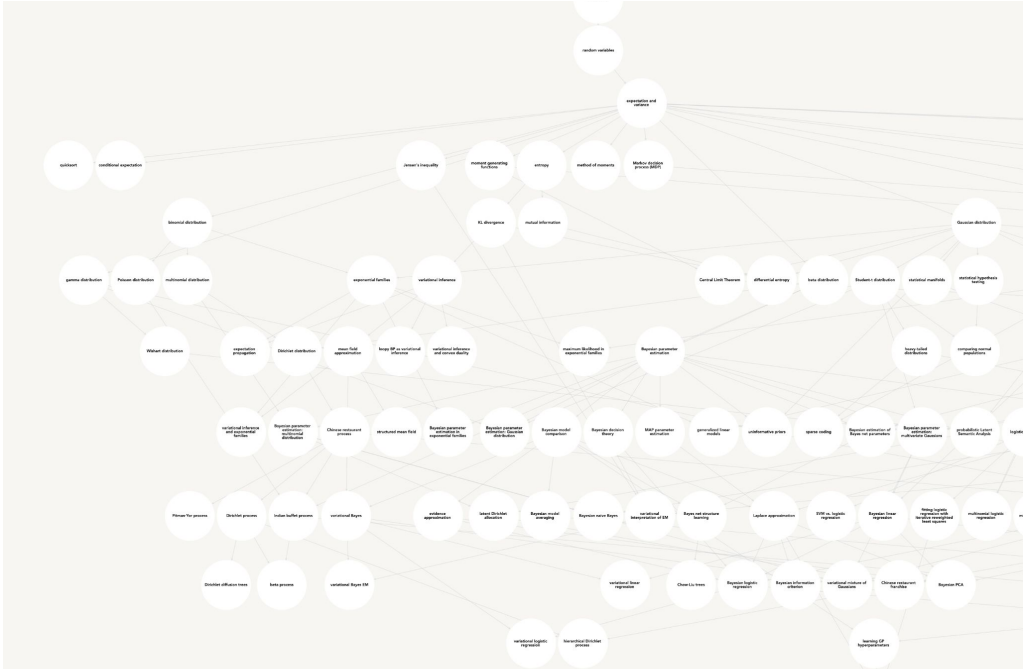
Concept Map

Implementation and Evaluation (Full Graph)

- Layout Complexity
- Styling Flexibility
- Function Flexibility
- How easy to implement
- Speed

Concept Map

Implementation and Evaluation (Full Graph)



Cytoscape

Concept Map

Implementation and Evaluation (Full Graph)

GoJS

Concept Map

Implementation and Evaluation (Full Graph)

d3dagre

Concept Map

Implementation and Evaluation (Full Graph)

- Layout Complexity
  - Cytoscape: Clear hierarchy, messy direct lines but easier to trace back
  - GoJS: Clean layout, zigzag lines, impossible to trace
  - d3dagre: Too many parallel lines, impossible to trace; too spacious

Concept Map

Implementation and Evaluation (Full Graph)

- Styling Flexibility
  - Cytoscape: text wrapper available, but rigorous on padding, which makes text too small
  - GoJS: text wrapper available, easy to configure, nice result
  - d3dagre: text wrapper not available, but shape will adopt to the text automatically; basically full control

Concept Map

Implementation and Evaluation (Full Graph)

- Function Flexibility (especially for highlighting prerequisites)
  - Cytoscape: Extremely easy to find parents or ancestors as they have functions available to call directly
  - GoJS: Can find parents but ancestors need to hack through their API
  - d3dagre: No handy tools to calculate parents or ancestors at front-end, but flexible to read and manipulate any back-end data

Concept Map

Implementation and Evaluation (Full Graph)

- How easy to implement
  - Cytoscape: Detailed documentation, many examples, lots of available functions
  - GoJS: Detailed documentation, many examples, easy to configure
  - d3dagre: Limited documentation, steep learning curve because of d3

Concept Map

Implementation and Evaluation (Full Graph)

- Speed
  - Cytoscape: 10.26s because of this layout, much faster for layout like BF layout
  - GoJS: 3.56s, thanks to Canvas
  - d3dagre: 11.88s because of the layout and also the fact that SVG is not as fast as Canvas

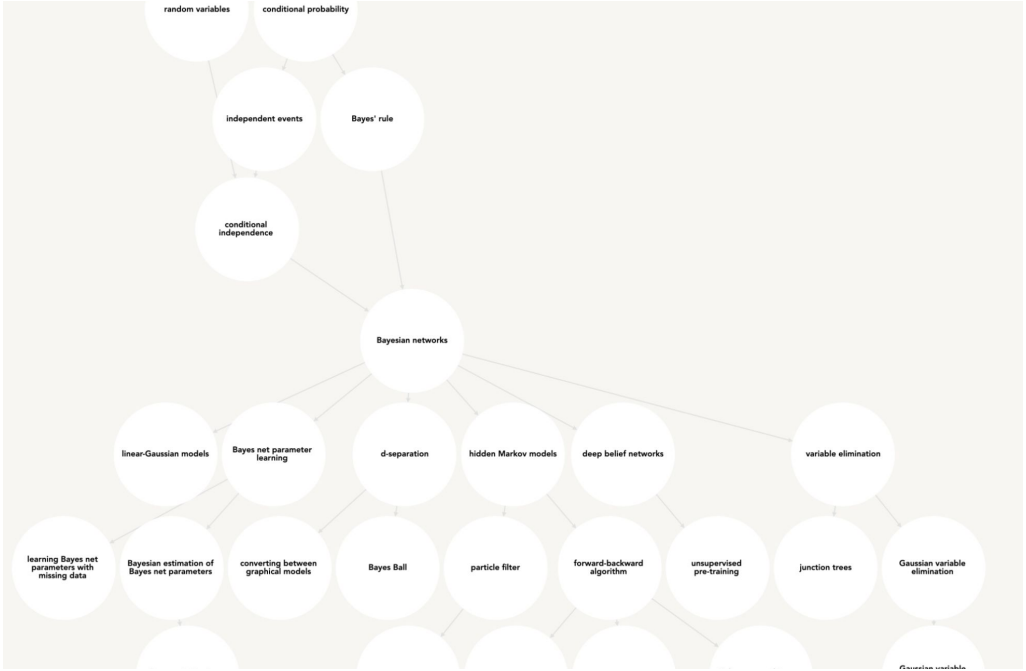
Concept Map

Implementation and Evaluation (Full Graph)

- More discussion
  - SVG, although slow, can manipulate every single DOM element inside, which makes us able to search text directly inside browser. But Canvas can't.
  - Full graph visualization still seem to be unrealistic
  - Maybe should try visualize a subgraph (clique) of the whole graph

Concept Map

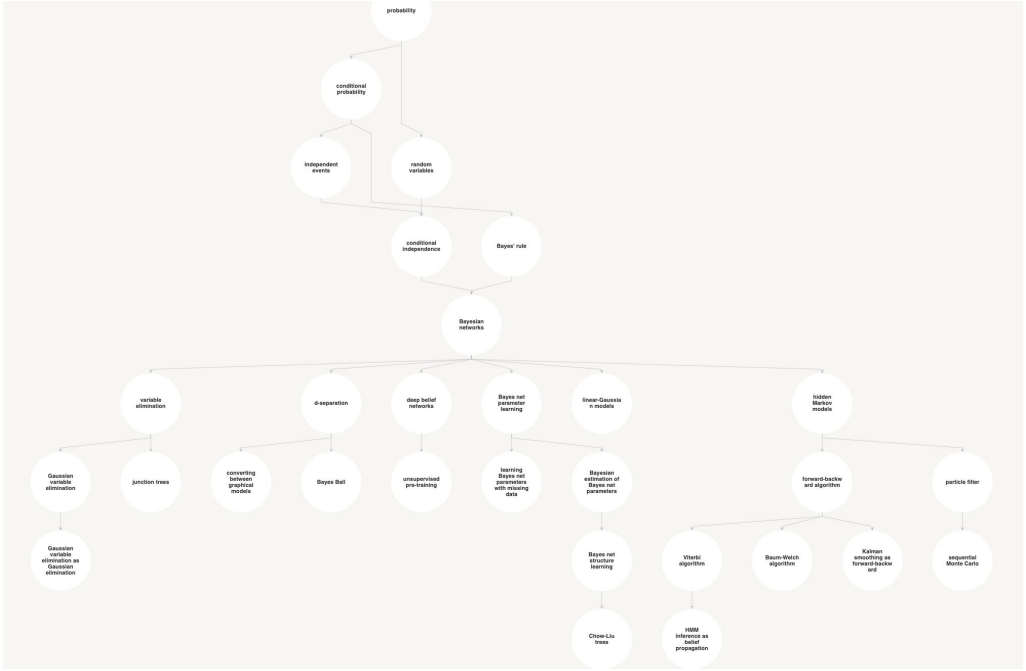
Implementation and Evaluation (Sub Graph)



Cytoscape

Concept Map

Implementation and Evaluation (Sub Graph)



GoJS

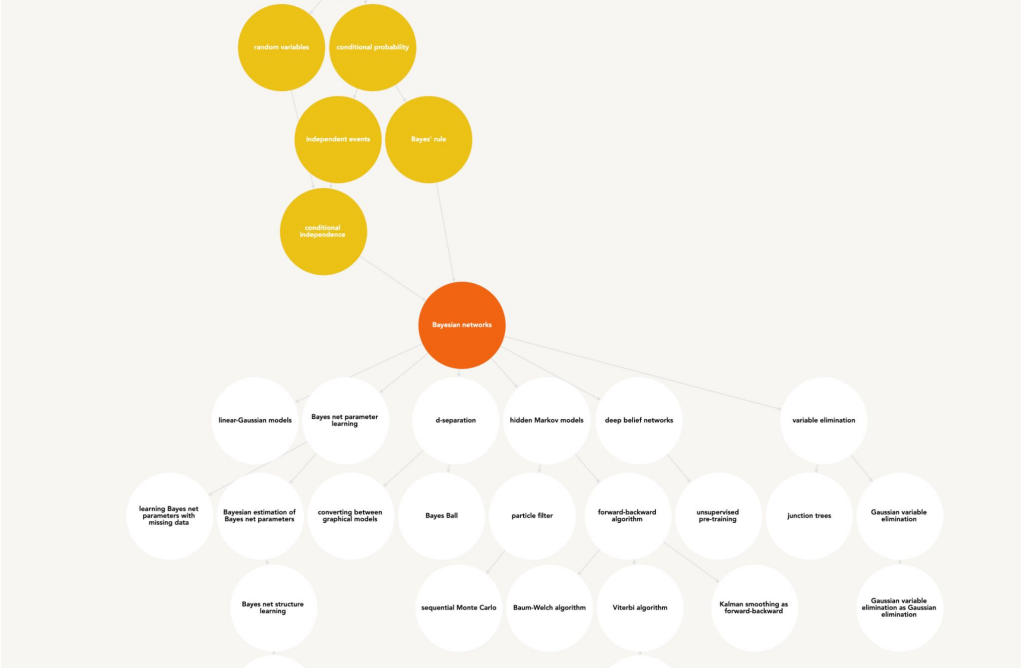
Concept Map

Implementation and Evaluation (Sub Graph)

d3dagre

Concept Map

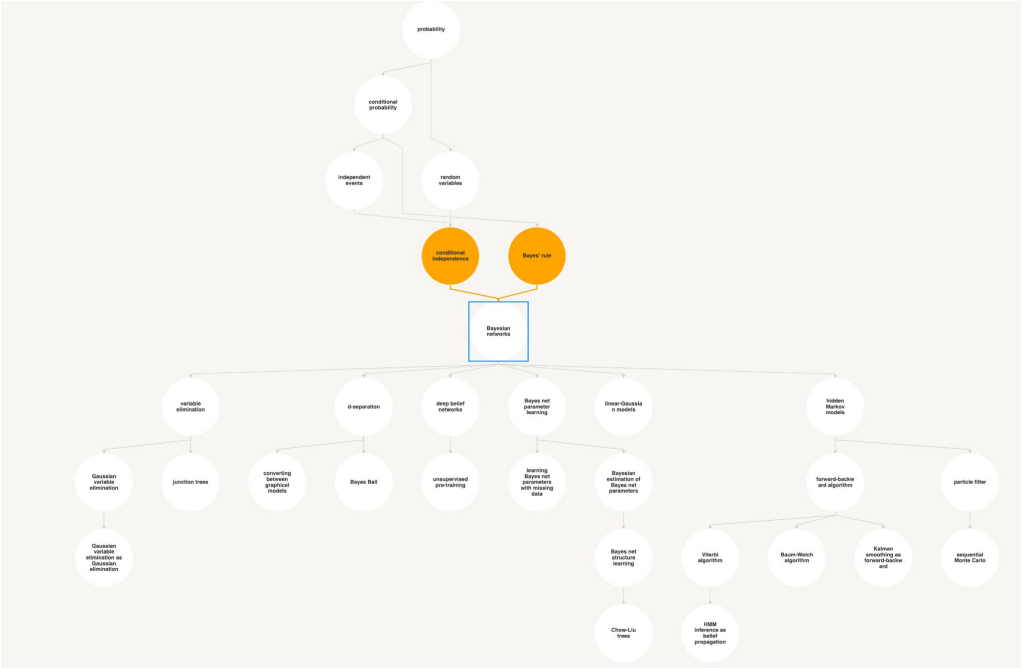
Implementation and Evaluation (Sub Graph)



Cytoscape

Concept Map

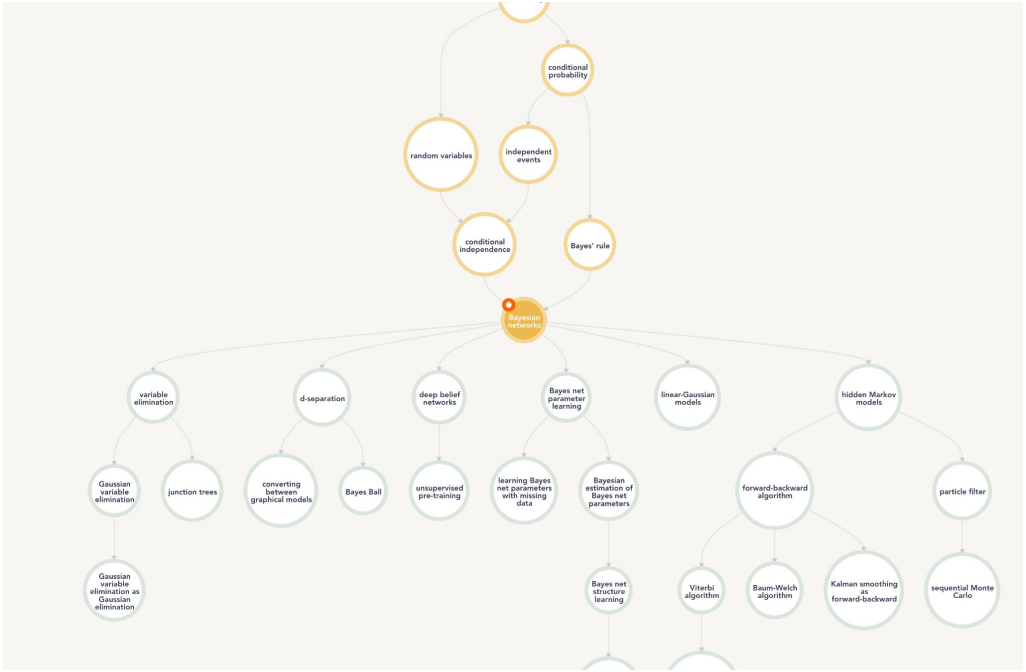
Implementation and Evaluation (Sub Graph)



GoJS

Concept Map

Implementation and Evaluation (Sub Graph)



d3dagre

Concept Map

Implementation and Evaluation (Sub Graph)

- More discussion
  - GoJS, although still has cleaner layout, becomes disadvantageous because of its zigzag edges.
  - They probably don’t do routing of the edges. Ironically, this might be desirable for really large graphs according to our previous experiments.
  - Sub graph seems to be a feasible solution for now

	Custom	Doc	Interactiveness	Open Source	Speed	Free	Easiness	Canv	SVG
GraphViz	☆☆☆☆	☆☆☆	☆	✔	☆☆☆☆☆	✔	☆☆☆☆☆	×	✔
TikZ	☆☆☆☆☆	☆☆☆☆☆	☆	-	☆☆☆☆☆	✔	☆☆☆☆☆	×	✔
Cytoscape	☆☆☆	☆☆☆☆☆	☆☆☆	✔	☆☆☆	✔	☆☆☆☆	✔	×
GoJS	☆☆☆	☆☆☆☆☆	☆☆☆	×	☆☆☆☆	×	☆☆☆☆	✔	×
d3dagre	☆☆☆☆☆	☆	☆☆☆☆☆	✔	☆☆	✔	☆	×	✔
sigmajs	☆☆☆☆☆	☆☆	☆☆☆☆☆	✔	☆☆☆☆☆	✔	☆☆☆☆	✔	✔
anychart	☆☆	☆☆☆☆	☆☆☆☆	×	☆☆☆	×	☆☆☆	×	✔
amchart	☆☆	☆☆☆	☆☆☆☆	×	☆☆☆☆☆	×	☆☆☆☆	×	✔

Thanks!

References

Alexander, T. Desmond. "From Adam to Judah: The Significance of the Family Tree in Genesis." *Evangelical Quarterly* 61.1 (1989): 5-19.

Reingold, Edward M., and John S. Tilford. "Tidier drawings of trees." *IEEE Transactions on software Engineering* 2 (1981): 223-228.

Stuckey, Peter J. "NP-completeness of minimal width unordered tree layout." *Graph Algorithms and Applications 5* 5 (2006): 295.

Knuth, Donald E. "Optimum binary search trees." *Acta informatica* 1.1 (1971): 14-25.

Wetherell, Charles, and Alfred Shannon. "Tidy drawings of trees." *IEEE Transactions on software Engineering* 5 (1979): 514-520.

Sugiyama, Kozo, Shojiro Tagawa, and Mitsuhiko Toda. "Methods for visual understanding of hierarchical system structures." *IEEE Transactions on Systems, Man, and Cybernetics* 11.2 (1981): 109-125.

Tamassia, Roberto, ed. *Handbook of graph drawing and visualization*. CRC press, 2013.

Ellson, John, et al. "Graphviz and dynagraph—static and dynamic graph drawing tools." *Graph drawing software*. Springer, Berlin, Heidelberg, 2004. 127-148.

Kruja, Eriola, et al. "A short note on the history of graph drawing." *International Symposium on Graph Drawing*. Springer, Berlin, Heidelberg, 2001.

Image References

https://www.connectedpapers.com/main/204e3073870fae3d05bcbc2f6a8e263d9b72e776/Attention-is-All-you-Need/graph
https://support.google.com/docs/answer/9146947?hl=en
https://miro.medium.com/max/1044/1*FBaB_aGUcWbfrLr1Ric5bQ.png
https://jech.bmj.com/content/jech/62/9/842/F2.large.jpg
https://community.atlassian.com/t5/image/serverpage/image-id/123488i68AC06362D325AF9?v=v2
https://miro.medium.com/max/1024/0*nnZdvIJqisZ5y7MB.png
https://observablehq.com/@d3/collapsible-tree