Grandalf : A Python module for Graph Drawings

https://github.com/bdcht/grandalf

Axel Tillequin
Bibliography on Graph Drawings - 2008-2010

June 2011

Outline

1. Graphs generalities
   - Motivations
   - Definitions
   - Graph Drawing Principles
2. Sugiyama Hierarchical Layout
   - 1. Layering
   - 2. Ordering
   - 3. x-coordinate assignment
   - 4. y-coordinate assignment
3. Force-driven Hierarchical Layout
   - Energy minimization
   - Hierarchical layer constraints
   - Quadratic Programming with Orthogonal Constraints

1. Graphs generalities
   - Motivations
   - Definitions
   - Graph Drawing Principles

2. Sugiyama Hierarchical Layout
   - 1. Layering
   - 2. Ordering
   - 3. x-coordinate assignment
   - 4. y-coordinate assignment

3. Force-driven Hierarchical Layout
   - Energy minimization
   - Hierarchical layer constraints
   - Quadratic Programming with Orthogonal Constraints

Motivations

Interactive drawing of an evolutionary *hierarchical graph* ?

Application: browsing the flow graph of a malware

- identifying/viewing/folding procedures by semantic analysis
- visualizing properties at some points in the program

Needs

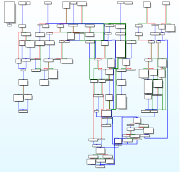
Interactive/adaptative drawings of small 2D directed graphs ($|V| \approx 100$): allowing node contraction (folding), and layout updating without entire recomputation !

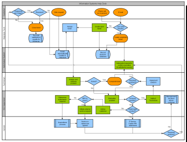
$\rightsquigarrow$ **Grandalf : small Python module for experimental Graph layout testings**

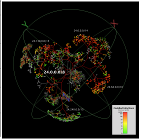
Existing Tools

see [1]

- program flow browsers:
  - IDA, BinNavi: good interfaces but fails at semantic-driven analysis,
- graph drawings:
  - general-purpose, 2D:
    - graphviz (open source, C),
    - OGDF (GPL, C++), PIGALE (GPL, C++), GUESS, ...
    - GDToolkit (commercial, C++), yFiles (commercial, Java)
  - Huge graphs, 2D: Tulip (GPL, C++),
  - Huge graphs, 3D: OGDF, Walrus (GPL, Java).



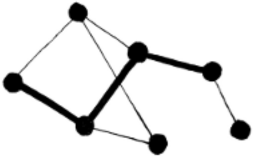




Graph theory basics

Definition

A Graph is a pair $G = (V, E)$ of sets such that $E \subset V^2$.

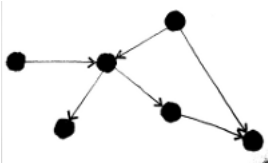


- V are vertices (nodes, points), and E are edges (lines, segments)
- $v \in V$ has neighbours (adjacent nodes)
- a path P is a subgraph of distinct nodes $v_0, ..., v_k$ s.t $(v_i, v_{i+1}) \in E$.
- G has k-connected components: forall $v_i, v_j \in V$, $\exists k$ paths.

Graph theory basics

Properties:

- A graph G is **directed** if one can distinguish initial/terminal nodes for an edge. G is **hierarchical** if it has also a rooted tree T.
- A graph G is **acyclic** if no path are closed. Then G is a forest, its components are trees.
- A graph is planar if it can be drawn on a plane with no edge crossing.



$\rightsquigarrow$ minimize edge-crossing for non-planar graphs !

Graph Drawing Principles

for *all* graphs :

Drawing Rules

Many static/dynamic, semantic/structural rules:

- avoiding node overlapping,
- minimizing edge crossing, minimize total edge length
- favor straight line placement, avoid edge bends,
- use hierarchical information,
- balance width/height,
- show symmetries,

$\Longrightarrow$ link with many NP-complete problems

Layouts and Methods

Hierarchical vs. undirected layouts:

Undirected graphs : 2D or 3D, can be of huge sizes, focusing mainly on connectivity so that force-driven (energy minimization) methods give good drawings.

Hierarchical graphs : mostly 2D (or 2.5D) graphs for which edge directions provide a natural global orientation of the graph from a top root node down to leaves.

Methods and Solvers

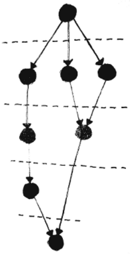
- heuristics algorithms
  $\Longrightarrow$ efficient but not suited to user-defined constraints
- Constraint based solvers
  $\Longrightarrow$ inefficient but depend on constraints expressions only

1. Layering

$L = (L_1, ..., L_h)$, ordered partition of V into layers L_i.

- directed acyclic graph: $\Longrightarrow$ cycle removal algorithm
- simple "natural" layering: put top nodes (no "in" edges) in queue,

```
for v in queue:
  v.rank=max([x.rank for x in v.N(-1)]+[-1])+1
  for e in v.e_out(): e.scan=True
  for x in v.N(+1):
    if not (False in [e.scan for e in x.e_in()]):
      queue.add(x)
```



- add "dummy" nodes for edges that span over several layers

minimum total edge length $\Longrightarrow$ simplex solver

2. Ordering

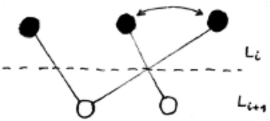
see [3]

Global ordering decomposed into iterated 2-layers ordering:

2-layers ordering

optimal edge crossing is NP ! but many heuristics for approx. solutions by fixing L_i and ordering $L_{i\pm1}$:

- **position** of v_j depend on upper/lower neighbours (median, barycenter)
- count all crossings and exchange accordingly.



3. x-coordinate assignment

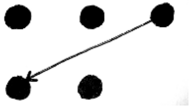
see [3, 1]

ordering $\neq$ horizontal assigment:
$\Longrightarrow$ need to guarantee vertical inner segments, fair balance etc.

Horizontal alignment

minimize $\sum_{e=(u,v)\in E} w(e)|u.x - v.x|$ subject to minimum separation

- select edges that influence alignment
- perform 4 vertical alignments with median heuristic:
  - upper/left, upper/right alignement
  - lower/left, lower/right alignement



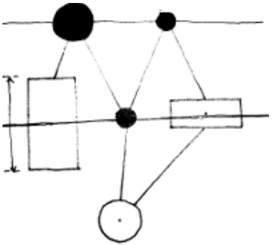
- sort 4 coords and set $v.x$ to barycenter of medians

4. y-coordinate assignment

What about Node size ?

Previous heuristics assume all nodes have same size, avoiding edge routing problems...

- height of layer L_i set to max height of its nodes...simple but not optimal !
- other heuristics exists[2]...more heuristics on heuristics...



Energy minimization

see [4, 5]

introduced in 1989 by Kamada&Kawai for undirected graph $G = (V, E)$. The idea is to minimize:

$$\sigma(X) = \sum_{i<j} w_{ij}(\|X_i - X_j\| - d_{ij})^2$$

where $X_i = (v_i.x, v_i.y)$, d_{ij} is the *ideal* distance between node v_i and v_j (ie. graph distance), and $w_{ij} = 1/d_{ij}^2$ is a normalization coefficent.
Note that, $\sum_{i<j} w_{ij}\|X_i - X_j\|^2 = Tr(X^t \cdot L^w \cdot X)$ with L^w the weighted Laplacian matrix of G: $L^w_{ij} = \begin{cases} \sum_{i \neq k} w_{ik}, & i = j \\ -w_{ij}, & i \neq j \end{cases}$ s.t we have:

$$\sigma(X) \leq Cst + Tr(X^t \cdot L^w \cdot X) - 2Tr(X^t \cdot L^{wd/Z} \cdot Z)$$

$$\partial F^Z(X) = 0 \Longrightarrow L^w \cdot X = L^{wd/Z} \cdot Z$$

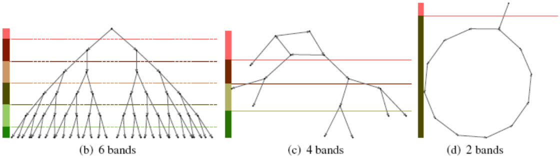
Hierarchical contrained Energy

see [6]

hierarchical energy: take $w_{ij} = 1$ and $\delta_{ij} = v_i.y - v_j.y$ (1 if $(v_i, v_j) \in E$), a partition of V is given by minimizing $E(Y) = \frac{1}{2}\sum_{i,j} w_{ij}(y_i - y_j - \delta_{ij})^2$

$$\implies L^w Y = b, \quad (Y \cdot 1 = 0)$$

with $b_i = \sum_j w_{ij}\delta_{ij}$.



(illustration from [6])

L^w is semi-definite positive $\implies$ Conjugated gradient $O(n)$ iterations.

Quadratic Programming with Orthogonal Constraints

see [6, 5]

minimize $\sigma(X)$, subject to contraints :

$$\forall v_j \in level(i): \quad v_j.y \geq l_i, \quad i = 1, ..., k$$

$$\forall v_j \in level(i+1): \quad v_j.y + \Delta l \leq l_i, \quad i = 1, ..., k$$

$\Longrightarrow$

- gradient descent ($\tilde{X}_{k+1} = \min_X F^{X_k}(X)$)
- projection to levels $\Pi(\tilde{X}_{k+1}) = \hat{X}_{k+1}$
- set $X_{k+1} = X_k + \alpha(\hat{X}_{k+1} - X_k)$

References (1/2)

[1] Graphviz:
http://www.graphviz.org/

[2] Ellson et al.
Graphviz and Dynagraph - Static and Dynamic Graph Drawing Tools.

[3] Brandes, Kopf
Fast and Simple Horizontal Coordinate Assigment
GD 2001, LNCS 2265, pp. 31-44, 2002.

[4] Gansner, North
Improved force-directed layouts
AT&T Labs, 1998.

[5] Gansner, Koren, North
Graph Drawing by Stress Majorization
AT&T Labs, 2004.

[6] Dwyer, Koren
DIG-COLA: Directed Graph Layout through Constrained Energy Minimization, 2005.

References (2/2)

- [7] M. Forster
  A Fast and Simple Heuristic for Constrained Two-Level Crossing Reduction
- [8] North, Woodhull
  On-line Hierarchical Graph Drawing
  AT&T Labs
- [9] Eiglsperger et al.
  An Efficient Implementation of Sugiyama's Algorithm for Layered Graph Drawing
  Journal of Graph Algorithms and Applications, http://jgaa.info/, vol. 9 no.3, 2005.
- [10] Gajer, Kobourov
  GRIP: Graph Drawing with Intelligent Placement
  Journal of Graph Algorithms and Applications, vol. 6 no.3, 2002.
- [11] Dwyer, Koren, Marriott
  Stress Majorization with Orthogonal Ordering Constraints
  GD 2005.