Visualisation of graphs

Hierarchical layouts

Sugiyama framework

Jonathan Klawitter · Summer semester 2020
Hierarchical drawings – motivation
Hierarchical drawing

Problem statement.

- Input: digraph $G = (V, E)$
- Output: drawing of $G$ that “closely” reproduces the hierarchical properties of $G$

Desireable properties.

- vertices occur on (few) horizontal lines
- edges directed upwards
- edge crossings minimized
- edges upward, straight, and short as possible
- vertices evenly spaced

Criteria can be contradictory!
Hierarchical drawing – applications

yEd Gallery: Java profiler JProfiler using yFiles

Source: Visualization that won the Graph Drawing contest 2016. Klawitter & Mchedlidze

Source: "Design Considerations for Optimizing Storyline Visualizations" Tanahashi et al.
Classical approach – Sugiyama framework

[Sugiyama, Tagawa, Toda '81]
Step 1: Cycle breaking

Approach.
- Find minimum set $E^*$ of edges which are not upwards.
- Remove $E^*$ and insert reversed edges.

**Problem** Minimum Feedback Arc Set (FAS).
- Input: directed graph $G = (V, E)$
- Output: min. set $E^* \subseteq E$, so that $G - E^* + E_r^*$ acyclic
Step 1: Cycle breaking

Approach.
- Find minimum set $E^*$ of edges which are not upwards.
- Remove $E^*$ and insert reversed edges.

Problem **Minimum Feedback Arc Set** (FAS).
- Input: directed graph $G = (V, E)$
- Output: min. set $E^* \subseteq E$, so that $G - E^* + E^*_r$ acyclic

...NP-hard :-(
Heuristic 1
[Berger, Shor '90]

GreedyMakeAcyclic(Digraph $G = (V, E)$)

$E' \leftarrow \emptyset$

foreach $v \in V$ do

if $|N\rightarrow(v)| \geq |N\leftarrow(v)|$ then

$E' \leftarrow E' \cup N\rightarrow(v)$

else

$E' \leftarrow E' \cup N\leftarrow(v)$

remove $v$ and $N(v)$ from $G$.

return $(V, E')$

- $G' = (V, E')$ is a DAG
  - we create an order on $V$
  - $E \setminus E'$ is a feedback arc set

- Time: $O(|V| + |E|)$

- Quality guarantee: $|E'| \geq |E|/2$
Heuristic 2

[Eades, Lin, Smyth '93]

\( E' \leftarrow \emptyset \)

while \( V \neq \emptyset \) do

\( \text{while in } V \text{ exists a sink } v \text{ do} \)

\( E' \leftarrow E' \cup N^{\leftarrow}(v) \)

remove \( v \) and \( N^{\leftarrow}(v) \)

Remove all isolated vertices from \( V \)

\( \text{while in } V \text{ exists a source } v \text{ do} \)

\( E' \leftarrow E' \cup N^{\rightarrow}(v) \)

remove \( v \) and \( N^{\rightarrow}(v) \)

if \( V \neq \emptyset \) then

let \( v \in V \) such that \( |N^{\rightarrow}(v)| - |N^{\leftarrow}(v)| \) maximal;

\( E' \leftarrow E' \cup N^{\rightarrow}(v) \)

remove \( v \) and \( N(v) \)

\begin{itemize}
  \item Time: \( O(|V| + |E|) \)
  \item Quality guarantee: \( |E'| \geq |E|/2 + |V|/6 \)
\end{itemize}
Step 2: Leveling

Problem.

- **Input:** acyclic, digraph $G = (V, E)$
- **Output:** Mapping $y: V \rightarrow \{1, \ldots, |V|\}$, so that for every $uv \in A$, $y(u) < y(v)$.

Objective is to minimize ...

- number of layers, i.e. $|y(V)|$
- length of the longest edge, i.e. $\max_{uv \in A} y(v) - y(u)$
- width, i.e. $\max\{|L_i| \mid 1 \leq i \leq h\}$
- total edge length, i.e. number of dummy vertices
Min number of layers

Algorithm.

- for each source $q$
  set $y(q) := 1$

- for each non-source $v$
  set $y(v) := \max \{y(u) \mid uv \in E\} + 1$

Observation.

- $y(v)$ is length of the longest path from a source to $v$ plus 1.
  ...which is optimal!

- Can be implemented in linear time with recursive algorithm.
Example
Total edge length – ILP

Can be formulated as an integer linear program:

\[
\begin{align*}
\text{min} & \quad \sum_{(u,v) \in E} (y(v) - y(u)) \\
\text{subject to} & \quad y(v) - y(u) \geq 1 \quad \forall (u,v) \in E \\
& \quad y(v) \geq 1 \quad \forall v \in V \\
& \quad y(v) \in \mathbb{Z} \quad \forall v \in V
\end{align*}
\]

One can show that:
- Constraint-matrix is **totally unimodular**
  \(\Rightarrow\) Solution of the relaxed linear program is integer
- The total edge length can be minimized in polynomial time
Drawings can be very wide.
Narrower layer assignment

Problem: Leveling with a given width.

- **Input:** acyclic, digraph $G = (V, E)$, width $W > 0$
- **Output:** Partition the vertex set into a minimum number of layers such that each layer contains at most $W$ elements.

Problem: Precedence-Constrained Multi-Processor Scheduling

- **Input:** $n$ jobs with unit (1) processing time, $W$ identical machines, and a partial ordering $<$ on the jobs.
- **Output:** Schedule respecting $<$ and having minimum processing time.
- **NP-hard, $(2 - \frac{2}{W})$-Approx., no $(\frac{4}{3} - \varepsilon)$-Approx. ($W \geq 3$).
Approximating PCMPS

- jobs stored in a list $L$
  (in any order, e.g., topologically sorted)
- for each time $t = 1, 2, \ldots$ schedule $\leq W$ available jobs
- a job in $L$ is *available* when all its predecessors have been scheduled
- as long as there are free machines and available jobs, take the first available job and assign it to a free machine
Approximating PCMPS

**Input:** Precedence graph (divided into layers of arbitrary width)

![Precedence Graph](image)

Number of Machines is $W = 2$.

**Output:** Schedule

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>1 2 4 5 6 8 A C E G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>_ 3 _ _ 7 9 B D F _</td>
</tr>
<tr>
<td>$t$</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

**Question:** Good approximation factor?
Approximating PCMPS - analysis for $W = 2$

Precedence graph $G_<$

Schedule

<table>
<thead>
<tr>
<th>Schedule</th>
<th>$M_1$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_2$</td>
<td>-3</td>
<td></td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>1 2 3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"The art of the lower bound"

$\text{OPT} \geq \lceil n/2 \rceil$ and $\text{OPT} \geq \ell := \text{Number of layers of } G_<$

Goal: measure the quality of our algorithm using the lower bounds

Bound. $\text{ALG} \leq \left\lceil \frac{n + \ell}{2} \right\rceil \approx \frac{n}{2} + \ell/2 \leq 3/2 \cdot \text{OPT}$

insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$

$\leq (2 - 1/W) \cdot \text{OPT}$ in general case
Step 3: Crossing minimization

**Problem.**
- **Input:** Graph $G$, layering $y: V \rightarrow \{1, \ldots, |V|\}$
- **Output:** (Re-)ordering of vertices in each layer so that the number of crossings in minimized.

- NP-hard, even for 2 layers [Garey & Johnson ’83]
- hardly any approaches optimize over multiple layers :(
Iterative crossing reduction – idea

Observation.
The number of crossings only depends on permutations of adjacent layers.

- Add dummy-vertices for edges connecting “far” layers.
- Consider adjacent layers $(L_1, L_2), (L_2, L_3), \ldots$ bottom-to-top.
- Minimize crossings by permuting $L_{i+1}$ while keeping $L_i$ fixed.
Iterative crossing reduction – algorithm

1. choose a random permutation of $L_1$
2. iteratively consider adjacent layers $L_i$ and $L_{i+1}$
3. minimize crossings by permuting $L_{i+1}$ and keeping $L_i$ fixed
   \textit{one-sided crossing minimization}
4. repeat steps (2)–(3) in the reverse order (starting from $L_h$)
5. repeat steps (2)–(4) until no further improvement is achieved
6. repeat steps (1)–(5) with different starting permutations
One-sided crossing minimization

**Problem.**

- **Input:** bipartite graph $G = (L_1 \cup L_2, E)$, permutation $\pi_1$ on $L_1$
- **Output:** permutation $\pi_2$ of $L_2$ minimizing the number of edge crossings.

One-sided crossing minimization is NP-hard.

[Eades & Whitesides '94]

**Algorithms.**

- barycenter heuristic
- median heuristic
- Greedy-Switch
- ILP

...
Barycentre heuristic
[Sugiyama et al. '81]

- Intuition: few intersections occur when vertices are close to their neighbours

- The barycentre of $u$ is the average $x$-coordinate of the neighbours of $u$ in layer $L_1$ $[x_1 \equiv \pi_1]$

  $$x_2(u) := \text{bary}(u) := \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v)$$

- Vertices with the same barycentre of are offset by a small $\delta$.

- linear runtime
- relatively good results
- optimal if no crossings are required \textcolor{red}{exercise!}
- $O(\sqrt{n})$-approximation factor

Worst case?
Median heuristic

[Eades & Wormald '94]

- \( \{v_1, \ldots, v_k\} := N(u) \) with \( \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k) \)

- \( x_2(u) := \text{med}(u) := \begin{cases} 0 & \text{when } N(u) = \emptyset \\ \pi_1(v_{[k/2]}) & \text{otherwise} \end{cases} \)

- move vertices \( u \) und \( v \) by small \( \delta \), when \( x_2(u) = x_2(v) \)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-approximation factor

proof in [GD Ch 11]
Median heuristic
[Eades & Wormald '94]

- \( \{v_1, \ldots, v_k\} := N(u) \) with \( \pi_1(v_1) < \pi_1(v_2) < \cdots < \pi_1(v_k) \)

- \( x_2(u) := \text{med}(u) := \begin{cases} 
0 & \text{when } N(u) = \emptyset \\
\pi_1(v_{[k/2]}) & \text{otherwise}
\end{cases} \)

- move vertices \( u \) und \( v \) by small \( \delta \), when \( x_2(u) = x_2(v) \)

- linear runtime
- relatively good results
- optimal, if no crossings are required
- 3-approximation factor

proof in [GD Ch 11]

Worst case?

\[ 2k(k + 1) + k^2 \text{ vs. } (k + 1)^2 \]
Greedy-switch heuristic

- Iteratively swap each adjacent node as long as crossings decrease
- Runtime $O(L_2^2)$ per iteration; at most $|L_2|$ iterations
- Suitable as post-processing for other heuristics

Worst case?

\[ \approx k^2 / 4 \quad \approx 2k \]
Integer linear program

[Jünger & Mutzel, '97]

- Constant $c_{ij} := \#$ crossings between edges incident to $v_i$ or $v_j$ when $\pi_2(v_i) < \pi_2(v_j)$
- Variable $x_{ij}$ for each $1 \leq i < j \leq n_2 := |L_2|$
  \[
  x_{ij} = \begin{cases} 
  1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\
  0 & \text{otherwise}
  \end{cases}
  \]
- The number of crossings of a permutations $\pi_2$
  \[
  \text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij} + \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji}
  \]
  constant
Integer linear program

- Minimize the number of crossings:

\[
\text{minimize } \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij}
\]

- Transitivity constraints:

\[
0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1 \quad \text{for } 1 \leq i < j < k \leq n_2
\]

i.e., if \(x_{ij} = 1\) and \(x_{jk} = 1\), then \(x_{ik} = 1\)

Properties.

- branch-and-cut technique for DAGs of limited size
- useful for graphs of small to medium size
- finds optimal solution
- solution in polynomial time is not guaranteed
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Iterations on example
Step 4: Vertex positioning

Goal.
paths should be close to straight, vertices evenly spaced

- **Exact:** Quadratic Program (QP)
- **Heuristic:** iterative approach
Quadratic Program

- Consider the path \( p_e = (v_1, \ldots, v_k) \) of an edge \( e = v_1v_k \) with dummy vertices: \( v_2, \ldots, v_{k-1} \)
- \( x \)-coordinate of \( v_i \) according to the line \( \overline{v_1v_k} \) (with equal spacing):

  \[
  x(v_i) = x(v_1) + \frac{i - 1}{k - 1} (x(v_k) - x(v_1))
  \]
- define the deviation from the line

  \[
  \text{dev}(p_e) := \sum_{i=2}^{k-1} \left( x(v_i) - x(v_i) \right)^2
  \]
- Objective function: \( \min \sum_{e \in E} \text{dev}(p_e) \)
- Constraints for all vertices \( v, w \) in the same layer with \( w \) right of \( v \):

  \[
  x(w) - x(v) \geq \rho(w, v)
  \]

- QP is time-expensive
- width can be exponential
Iterative heuristic

- compute an initial layout
- apply the following steps as long as improvements can be made:
  1. vertex positioning,
  2. edge straightening,
  3. compactifying the layout width.
Example
Step 5: Drawing edges

Possibility.
Substitute polylines by Bézier curves
Example
Example
Example
Classical approach – Sugiyama framework

[Sugiyama, Tagawa, Toda '81]

- Input
- Cycle breaking
- Leveling
- Crossing minimization
- Vertex positioning
- Edge drawing

- Flexible framework to draw directed graphs
- Sequential optimization of various criteria
- Modelling gives NP-hard problems, which can still be solved quite well
Literature

Detailed explanations of steps and proofs in
■ [GD Ch. 11] and [DG Ch. 5]
based on
■ [Sugiyama, Tagawa, Toda ’81] Methods for visual understanding of hierarchical system structures
and refined with results from
■ [Berger, Shor ’90] Approximation algorithms for the maximum acyclic subgraph problem
■ [Eades, Lin, Smith ’93] A fast and effective heuristic for the feedback arc set problem
■ [Garey, Johnson ’83] Crossing number is NP-complete
■ [Eades, Whiteside ’94] Drawing graphs in two layers
■ [Eades, Wormland ’94] Edge crossings in drawings of bipartite graphs
■ [Jünger, Mutzel ’97] 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms