Algorithms for Graph Visualization

Summer Semester 2016
Lecture #6

Hierarchical Drawings

(based on slides from Marcus Krug, KIT)
Example

E-Mail-Graph between groups in Computer Science, KIT
Hierarchical Drawing

Problem statement:

- **Input:** directed graph $D = (V, A)$
- **Output:** Drawing of $D$ which *closely* reproduces the hierarchical properties of $D$.

Desireable Properties

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

Criteria can be contradictory!
Classical Approach

[Sugiyama, Tagawa, Toda ’81]

Input → Cycle breaking → Leveling

Crossing minimization → Vertex positioning → Edge drawing
Step 1: Cycle Breaking

Approach

- Find minimum set $A^\star$ of edges which are not upwards.
- Remove $A^\star$ and insert reversed edges.

Problem **Minimum Feedback Arc Set (FAS):**

- Input: directed graph $D = (V, A)$
- Output: min. set $A^\star \subseteq A$, so that $D - A^\star$ acyclic

NP-hard :-(
Step 1: Cycle Breaking

Approach

▷ Find minimum set $A^*$ of edges which are not upwards.
▷ Remove $A^*$ and insert reversed edges.

Problem **Minimum Feedback Arc Set (FAS):**

▷ Input: directed graph $D = (V, A)$
▷ Output: min. set $A^* \subseteq A$, so that $D - A^* + A_r^*$ acyclic

$\text{NP-hard :-(}$
Greedy-Heuristic for FAS

GreedyMakeAcyclic(Digraph $D = (V, A)$)

$$A' \leftarrow \emptyset \text{ (these will be the edges we keep)}$$

foreach $v \in V$ do

if $\text{outdeg}(v) > \text{indeg}(v)$ then

$A' \leftarrow A' \cup \text{out}(v)$

else

$A' \leftarrow A' \cup \text{in}(v)$

$A \leftarrow A \setminus (\text{out}(v) \cup \text{in}(v))$

return $(V, A')$

$\gg$ Timing: $O(V + A)$

$\gg$ Quality guarantee: $|A'| \geq |A|/2$
Improved Greedy-Heuristic for FAS

- Each iteration of foreach, first look for sources and sinks. If there are none, pick $v$ such that $|\text{outdeg}(v) - \text{indeg}(v)|$ is maximized.

- Timing: $O(V + A)$

- Quality guarantee: $|A'| \geq |A|/2 + |V|/6$
Step 2: Leveling

Problem

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

Objective: minimize \ldots

- **Number of layers**, i.e. \(|y(V)|\)
- **Length of the longest edge**, i.e. \( \max_{uv \in A} y(v) - y(u) \)
- **Total edge length**, i.e. number of dummy vertices
Algorithm to Minimize the Number of Layers

\[ \text{for each source } q \]
set \( y(q) := 1 \)

\[ \text{for each non-source } v \]
set \( y(v) := \max \{ y(u) \mid uv \in A \} + 1 \)

**Obs.** \( y(v) \) is...
Length of the longest path from a source to \( v \) plus 1.
...also optimal with respect to the number of layers!

**Question:** Can we do this in linear time?
Linear time implementation

ComputeLayering(AcyclicDigraph \( D = (V, A) \))

\[
y = \text{new} \ \text{int}[1..|V|] \quad // \ \text{all} \ = \ 0
\]

foreach source \( q \in V \) do
\[
\quad y(q) \leftarrow 1
\]

foreach non-source \( v \in V \) do
\[
\quad \text{ComputeYRec}(D, v, y)
\]

return \( y \)

\[
\text{ComputeYRec}(\text{AcyclicDigraph} \ D = (V, A), \ \text{Vertex} \ v, \ \text{int[]} \ y)
\]

if \( y(v) \ == \ 0 \) then
\[
\quad y(v) \leftarrow \max \{ \text{ComputeYRec}(D, u, y) \mid uv \in A \} + 1
\]

return \( y(v) \)
Our Example

Looks good .... right?

The drawing can be very wide :-(
Goal: Narrower layer assignment.

Problem: Leveling with a given width.

- **Input:** acyclic, digraph $D = (V, A)$, 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.

$\triangleright$ NP-hard, $(2 - \frac{2}{W})$-Approx., no $(\frac{4}{3} - \varepsilon)$-Approx. ($W \geq 3$).
Approximating PCMPS

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

Output: Schedule

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

Question: Good approximation factor?
Algorithm

- 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.
Analysis for $W = 2$

**Precedence graph** $G_<$

```
1 → 2 → 5
3 → 6 → 9 → 8
4 → 7 → A → D
      E
```

**Schedule**

```
<table>
<thead>
<tr>
<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>-</td>
<td>3</td>
<td>-</td>
<td>7</td>
<td>9</td>
<td>B</td>
<td>D</td>
<td>F</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$t$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
```

"The art of the lower bound"

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

**Goal:** measure the quality of our algorithm using the lower bound(s).

\[
\text{Gen.} \leq (2 - 1/W) \cdot \text{OPT}
\]

**Bound**

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

Insertion of pauses (−) in the schedule (except the last) maps to layers of $G_<$.
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
- hardly any approaches optimize over multiple layers :-(
Iterative crossing reduction – idea

- 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.

**Obs.** The number of crossings only depends on permutations of adjacent layers.
Iterative crossing reduction – Algorithm

(1) choose a random permutation of $L_1$.

(2) iteratively consider adjacent layers $L_i$ und $L_{i+1}$.

(3) minimize crossings by permuting $L_{i+1}$ and keeping ($L_i$ fixed).

(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
One-sided Crossing Minimization

Problem

Input: bipartite graph $G = (L_1 \cup L_2, E)$, permutation $\pi_1$ on $L_1$

Output: permutation $\pi_2$ on $L_2$, the number of edge crossings is minimized.

One-sided crossing minimization is NP-hard.

Algorithms

- barycenter heuristic
- median heuristic
- Greedy-Switch
- ILP
Barycentre Heuristic

- 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 \text{ exercise!}
- $O(\sqrt{n})$-approximation factor
Median heuristic

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

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

\[ \text{move vertices } u \text{ und } v \text{ by small } \delta, \text{ when } x_2(u) = x_2(v) \]

\[ \text{linear runtime} \]
\[ \text{relatively good results} \]
\[ \text{optimal, if no crossings are required} \]
\[ 3\text{-Approximation factor} \]

Proof in [DETT]
Median heuristic [Eades & Wormald ’94]

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

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

move vertices \(u\) and \(v\) 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 [DETT]

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 $O(L_2)$ per iteration; at most $|L_2|$ iterations
- suitable as post-processing for other heuristics

Worst case?

\[ L_1 \approx k^2 / 4 \quad \approx 2k \]
Integer Linear Program

\[ c_{ij} := \text{num. of crossings between edges incident to } v_i \text{ or } v_j \text{ when } \pi_2(v_i) < \pi_2(v_j) \]

\[ x_{ij} \text{ for each } 1 \leq i < j \leq n_2 := |L_2| \]

\[ x_{ij} = \begin{cases} 1 & \text{when } \pi_2(v_i) < \pi_2(v_j) \\ 0 & \text{otherwise} \end{cases} \]

\[ \text{The number of crossings of a permutations } \pi_2 \]

\[ \text{cross}(\pi_2) = \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} (c_{ij} - c_{ji})x_{ij} + \sum_{i=1}^{n_2-1} \sum_{j=i+1}^{n_2} c_{ji} \]

\[ \text{constant} \]
Minimize the number of crossings:

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

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 \( x_{ik} = 1 \)

Solution can be found via Branch-and-Bound on small degree graphs relatively quickly

\[\text{(Transitivity)}\]
Our Example – iterations
Step 4: Vertex positioning

Goal: paths should be close to straight.

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 $v_1v_k$ (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

$$\text{dev}(p_e) := \sum_{i=2}^{k-1} \left(x(v_i) - x(v_i)\right)^2$$
QP (cont.)

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 $v$

$$x(w) - x(v) \geq \rho(w, v)$$

$\rho(w, v)$ is min. horizontal distance between $w$ and $v$

Problem: QP and potentially exponential width
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.
Our Example
Step 5: drawing the edges
Step 5 – drawing the edges

All figs. from [Kaufmann und Wagner: Drawing Graphs]
Our Example