NetworkX Graph Visualization

Stanisław Bitner, Aleksander Wojsz

October 14, 2024

Table of contents

Intro

- Math Representation
- Problem statement
- Hand Drawing
- Tablet Drawing
- Tikz
- NetworkX

Math Representation

$$
\begin{aligned}
G &= \langle V, E \rangle \\
V &= \{1 \ldots 9\} \\
E &= \{ \\
&\quad \{1,2\}, \{1,3\}, \{2,4\}, \{2,5\}, \{2,6\}, \\
&\quad \{3,5\}, \{3,6\}, \{4,7\}, \{4,8\}, \{5,7\}, \\
&\quad \{5,8\}, \{6,7\}, \{6,8\}, \{7,9\}, \{8,9\} \\
\}
\end{aligned}
$$

Math Representation

$$
\begin{aligned}
G &= \langle V, E \rangle \\
V &= \{1 \ldots 9\} \\
E &= \{ \\
&\{1,2\},\{1,3\},\{2,4\},\{2,5\},\{2,6\}, \\
&\{3,5\},\{3,6\},\{4,7\},\{4,8\},\{5,7\}, \\
&\{5,8\},\{6,7\},\{6,8\},\{7,9\},\{8,9\} \\
\}
\end{aligned}
$$

TERRIBLE!!!

Problem statement

Input

Graph $G = \langle V, E \rangle$

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

- adjacent vertices close

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

- adjacent vertices close
- non-adjacent vertices distant

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

- adjacent vertices close
- non-adjacent vertices distant
- short and similar in length edges

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

- adjacent vertices close
- non-adjacent vertices distant
- short and similar in length edges
- as few crossings as possible

Problem statement

Input

Graph $G = \langle V, E \rangle$

Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

- adjacent vertices close
- non-adjacent vertices distant
- short and similar in length edges
- as few crossings as possible
- nodes distributed evenly

Problem statement

Input

Graph $G = \langle V, E \rangle$

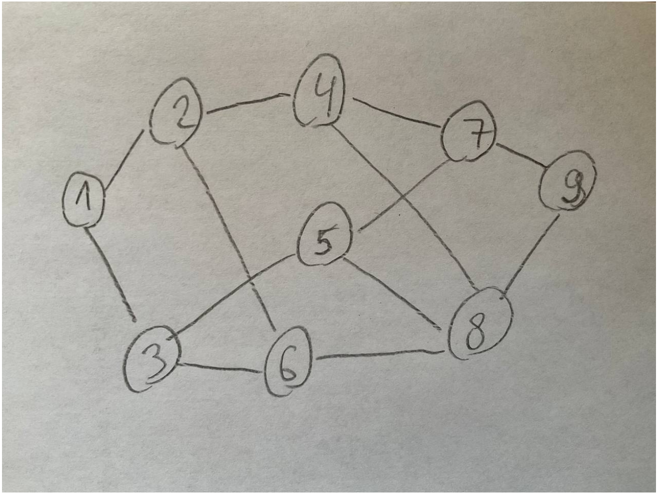
Output

Clear and readable drawing of G (we focus on straight-line edges).

Criteria

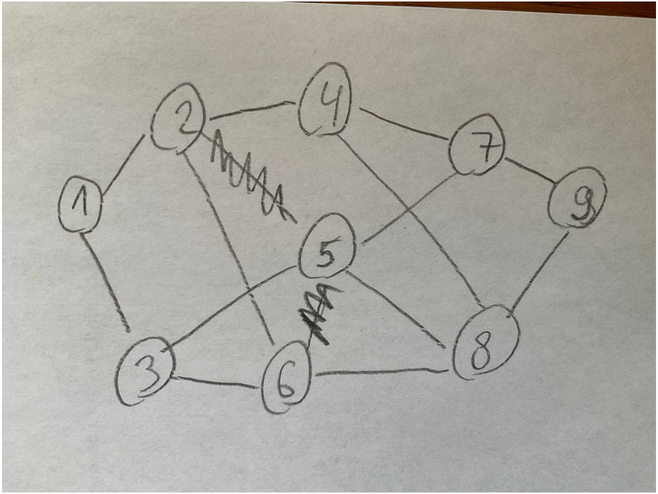
- adjacent vertices close
- non-adjacent vertices distant
- short and similar in length edges
- as few crossings as possible
- nodes distributed evenly
- clusters together

Hand Drawing

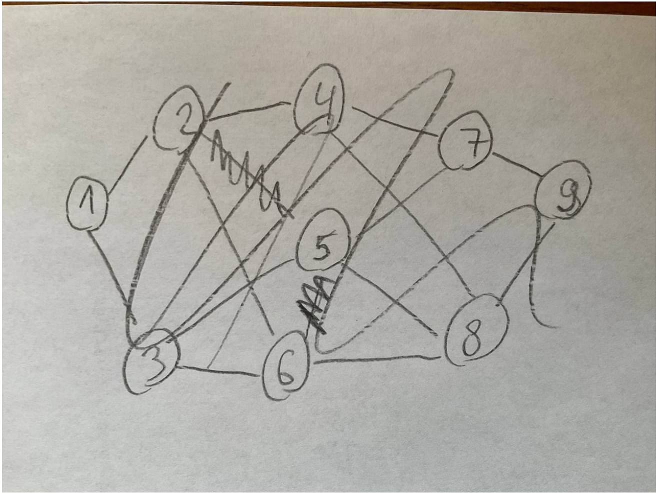


Hand Drawing

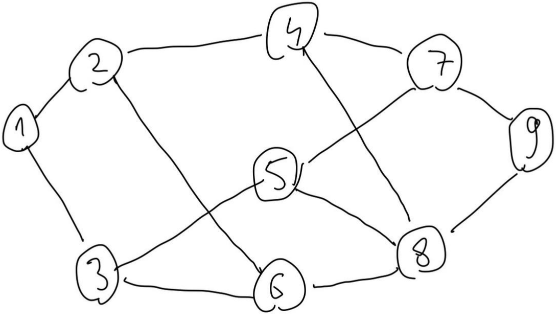
Mistake



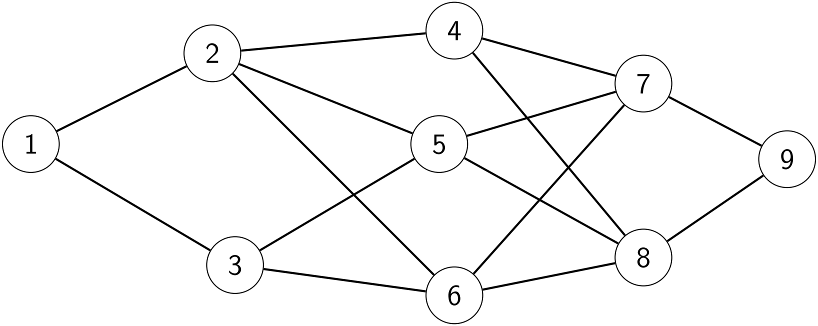
Hand Drawing
Mistake



Tablet Drawing

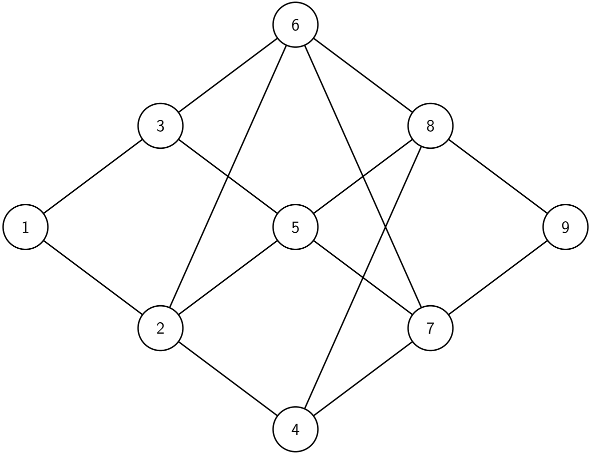


Tikz



```
\node[e4c node] (1) at (0.00, 0.89) {1};
\node[e4c node] (2) at (0.24, 1.01) {2};
\node[e4c node] (3) at (0.27, 0.73) {3};
\node[e4c node] (4) at (0.56, 1.04) {4};
\node[e4c node] (5) at (0.54, 0.89) {5};
\node[e4c node] (8) at (0.81, 0.74) {8};
\node[e4c node] (7) at (0.81, 0.97) {7};
\node[e4c node] (9) at (1.00, 0.87) {9};
\node[e4c node] (6) at (0.56, 0.69) {6};
\path[draw,thick]
    (1) edge[e4c edge]  (2)
    (1) edge[e4c edge]  (3)
    (2) edge[e4c edge]  (4)
    ...
```

NetworkX



```
G = nx.Graph()
G.add_edges_from([
    (1, 2), (1, 3), (2, 4), (2, 5), (2, 6),
    (3, 5), (3, 6), (4, 7), (4, 8), (5, 7),
    (5, 8), (6, 7), (6, 8), (7, 9), (8, 9)
])
pos = nx.bfs_layout(G, 1)
```

```
G = nx.Graph()
G.add_edges_from([
    (1, 2), (1, 3), (2, 4), (2, 5), (2, 6),
    (3, 5), (3, 6), (4, 7), (4, 8), (5, 7),
    (5, 8), (6, 7), (6, 8), (7, 9), (8, 9)
])
pos = nx.bfs_layout(G, 1)
```

Materials for presentation

GitHub - `https://github.com/Rentib/graph-visualization`
Google Colab - `http://tiny.cc/networkx`

How is it done?

- Overview
- Bipartite
- BFS
- ForceAtlas2
- Force-directed drawings

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning
- Flexible styling

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning
- Flexible styling

Support

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning
- Flexible styling

Support

- Image, PDF, SVG, and other formats

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning
- Flexible styling

Support

- Image, PDF, SVG, and other formats
- Interactive visualization using matplotlib

Overview

Algorithms

NetworkX provides several algorithms for graph visualization, focusing on different layout strategies to represent nodes and edges effectively.

Eye candy

- Nodes, edges, labels, and other graph elements
- Custom positioning
- Flexible styling

Support

- Image, PDF, SVG, and other formats
- Interactive visualization using matplotlib
- **LaTeX** (Tikz, PGF)

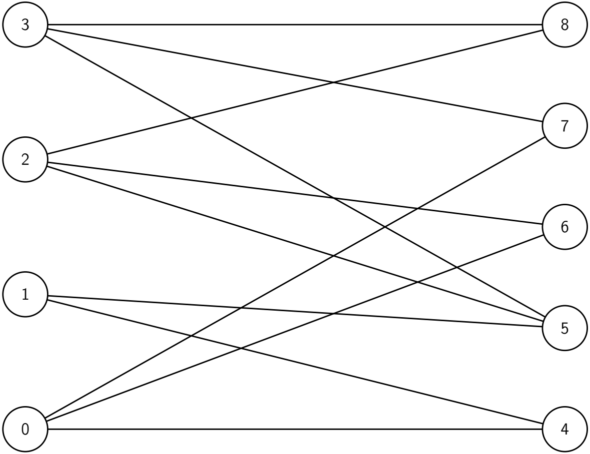
Bipartite

Algorithm

Do 2-coloring and group nodes by color.

Algorithm

Do 2-coloring and group nodes by color.



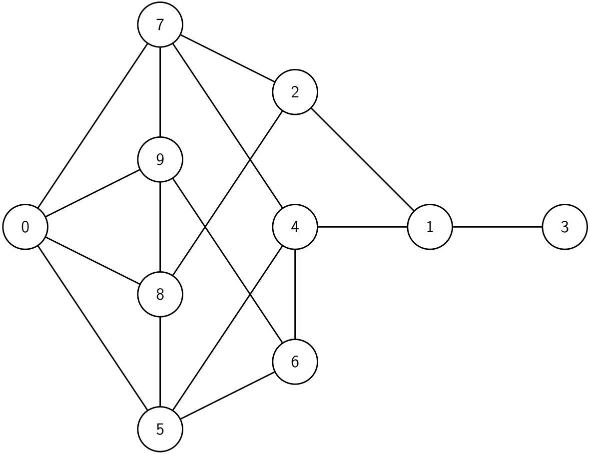
BFS

Algorithm

Run breadth first search and group nodes by depth.

Algorithm

Run breadth first search and group nodes by depth.



ForceAtlas2

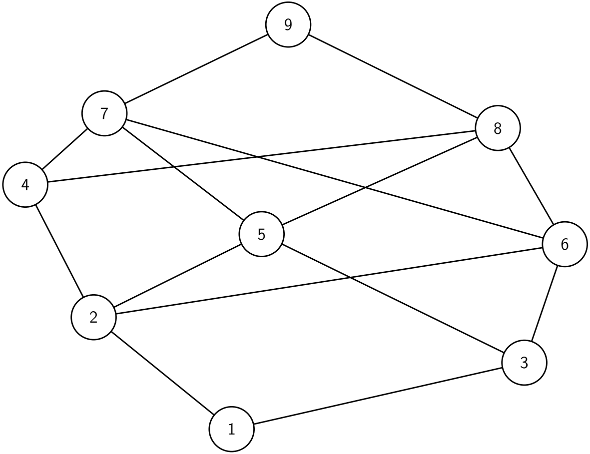
Algorithm

Out of scope for this presentation, but worth mentioning as the results are fantastic.

ForceAtlas2

Algorithm

Out of scope for this presentation, but worth mentioning as the results are fantastic.



Force-directed drawings

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E \rangle, p = (p_v)_{v \in V}, \varepsilon > 0, K \in \mathbb{N}$)

return p

end function

Force-directed drawings

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E \rangle, p = (p_v)_{v \in V}, \varepsilon > 0, K \in \mathbb{N}$)
  $l \leftarrow 1$
  while $l < K \wedge \max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**
    $l \leftarrow l + 1$
  end while
  return p
end function

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E\rangle, p = (p_v)_{v\in V}, \varepsilon > 0, K \in \mathbb{N}$)
    $l \leftarrow 1$
    while $l < K \wedge \max_{v\in V} \|F_v(t)\| > \varepsilon$ **do**
        for all $u \in V$ **do**

        end for

        $l \leftarrow l + 1$
    end while
    return p
end function

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E\rangle, p = (p_v)_{v \in V}, \varepsilon > 0, K \in \mathbb{N}$)
  $l \leftarrow 1$
  while $l < K \wedge \max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**
    for all $u \in V$ **do**
      $F_u(t) \leftarrow \sum_{v \in V} f_{\text{rep}}(u, v) + \sum_{uv \in E} f_{\text{attr}}(u, v)$
    end for
    $l \leftarrow l + 1$
  end while
  return p
end function

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E \rangle, p = (p_v)_{v \in V}, \varepsilon > 0, K \in \mathbb{N}$)
  $l \leftarrow 1$
  while $l < K \wedge \max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**
    for all $u \in V$ **do**
      $F_u(t) \leftarrow \sum_{v \in V} f_{\mathrm{rep}}(u, v) + \sum_{uv \in E} f_{\mathrm{attr}}(u, v)$
    end for
    for all $u \in V$ **do**

    end for
    $l \leftarrow l + 1$
  end while
  return p
end function

Analogy to physics

Edges are *springs*, vertices are *repelling objects*.

function ForceDirected($G = \langle V, E\rangle, p = (p_v)_{v \in V}, \varepsilon > 0, K \in \mathbb{N}$)
  $I \leftarrow 1$
  while $I < K \wedge \max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**
    for all $u \in V$ **do**
      $F_u(t) \leftarrow \sum_{v \in V} f_{\mathbf{rep}}(u, v) + \sum_{uv \in E} f_{\mathbf{attr}}(u, v)$
    end for
    for all $u \in V$ **do**
      $p_u \leftarrow p_u + \delta(t) \cdot F_u(t)$
    end for
    $I \leftarrow I + 1$
  end while
  return p
end function

Conclusion

NetworkX is GREAT!!!