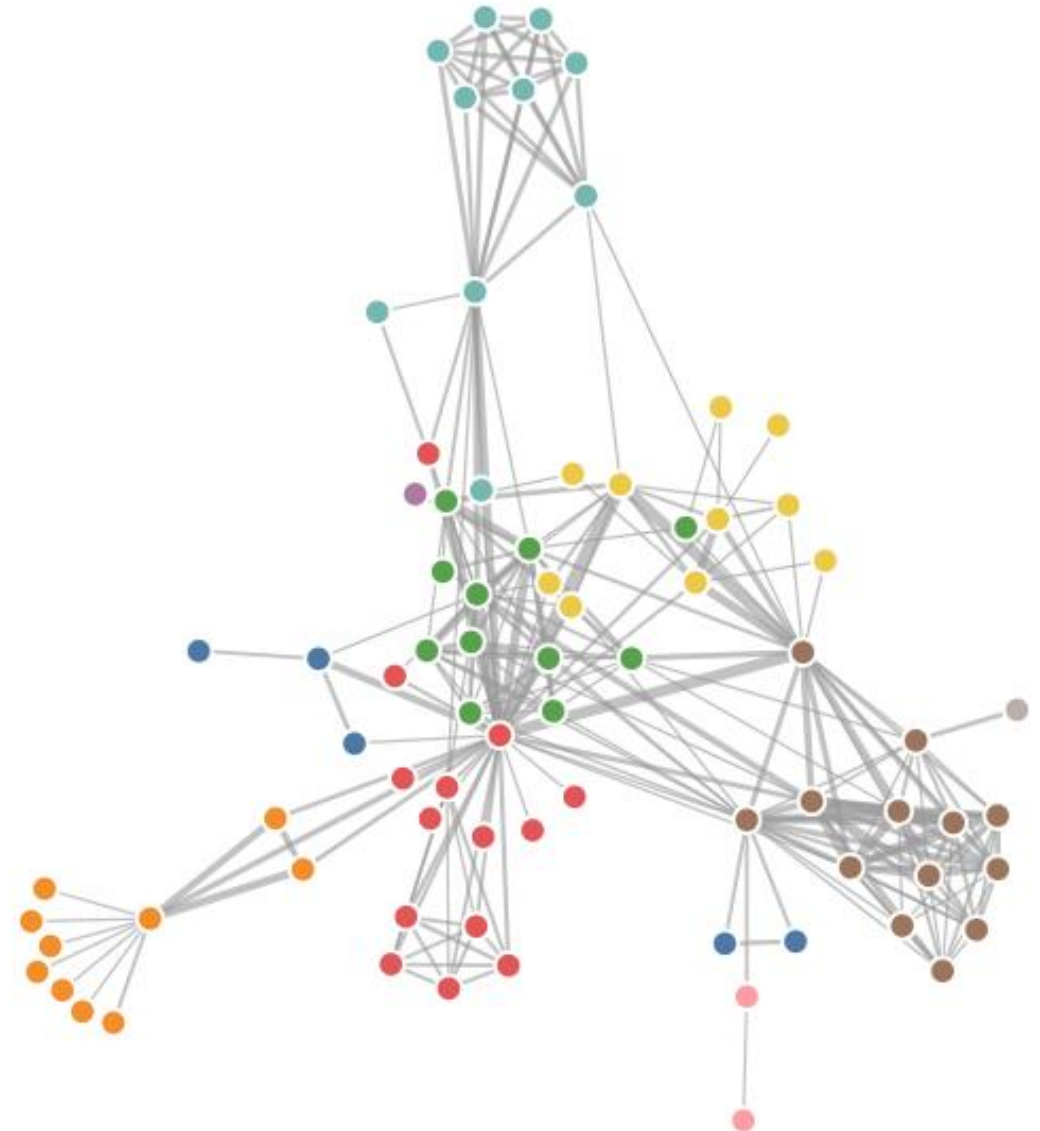


D3: Force Directed

Tommaso Piselli

Force Directed Layout

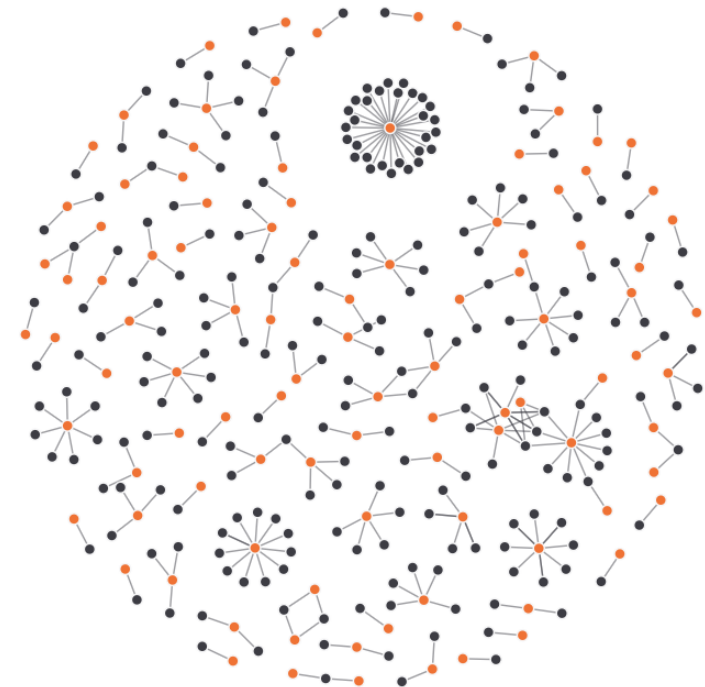
- The **force layout** is a physical inspired layout to determine a network's most optimal graphical representation.
- **Idea:**
 - All the edges have almost equal length and there are as few crossings as possible.
 - The forces are based on the relative positions of the nodes and edges.
 - Then, we use these forces either to simulate the motion of the edges and nodes or to minimize their energy.
- In D3, unlike other layouts, the **force layout** operates in real-time rather than as a preprocessing step before rendering.



Force Directed Layout and D3

- To run a D3 simulation, we call the method `d3.forceSimulation()` from the module [d3-force](#)
- Based on the forces applied to the simulation, D3 appends two types of information to each node:
 - their next position (`x` and `y`)
 - their next velocity (`vx` and `vy`)

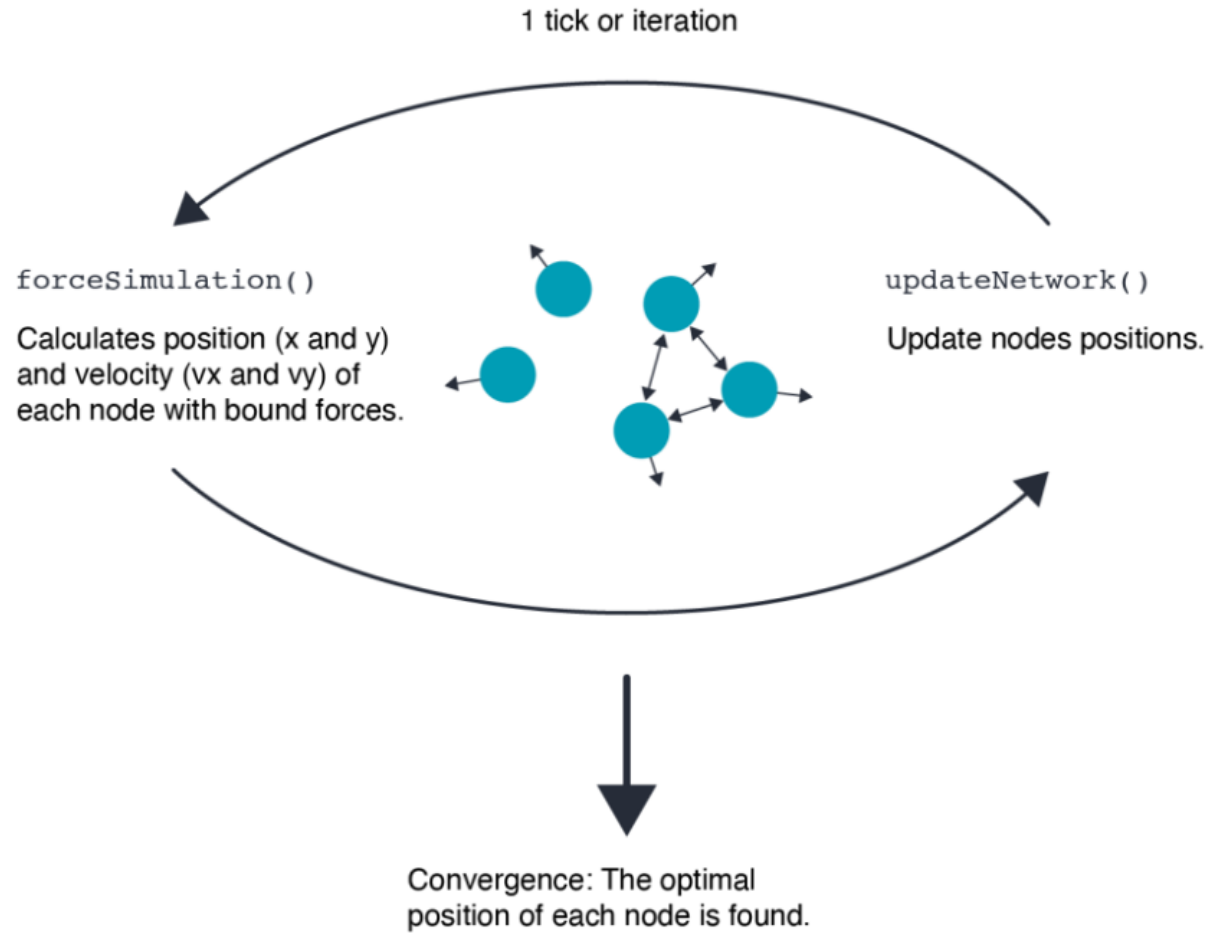
d3-force



Force Directed and D3

- Every time D3 completes an iteration of the simulation, named **tick**, we call a custom function.
- In this function, we change the position of the **svg circle** (a.k.a. the **node**) according to the data appended by D3.
 - With a spark of fantasy, this function is usually called **ticked()** as a convention.
- We perform this cycle until the **simulation converges**, meaning that the nodes found an optimal position according to the bound forces.

Force Directed and D3



Positioning Forces

- The positioning forces are called with the methods:
 - `forceX()`: makes the nodes move toward a horizontal position
 - `forceY()`: makes the nodes move toward a vertical position
- So, if we set their relative position values to `0`, the nodes will position themselves into the other direction.
- If we set both their positioning forces to `0`, the nodes will stack one on top of the other.

This is not the intensity of the force, but the value of the coordinate

```
forceSimulation()  
  .force("x", d3.forceX(0) )
```



```
forceSimulation()  
  .force("y", d3.forceY(0) )
```



```
forceSimulation()  
  .force("x", d3.forceX(0) )  
  .force("y", d3.forceY(0) )
```



Positioning Forces: Strength

- We can control the intensity of the forces with the `strength()` accessor function of the positioning forces.
 - By default, it is set to `1`.

Here, by setting the **strength** of the force in the **X** direction to `0.01`, we strengthen the attraction towards the coordinate `(0, 0)` from the vertical direction.

```
forceSimulation()  
  .force("x", d3.forceX(0) )  
  .force("y", d3.forceY(0) )
```



```
forceSimulation()  
  .force("x", d3.forceX(0).strength(0.01) )  
  .force("y", d3.forceY(0) )
```



```
forceSimulation()  
  .force("x", d3.forceX(0) )  
  .force("y", d3.forceY(0).strength(0.01) )
```



Collision Forces

- To prevent node overlapping, we can use the collision force `d3.forceCollide()`.
- As a parameter, this function takes a **radius** used to set the minimum distance between the nodes.
 - The radius is considered from the center of the nodes.
- If we apply the `forceCollide()` into specific directions, we obtain a **beeswarm layout**.

In this example, nodes have a radius of 10px and the `forceCollide` is set at a distance of 12px. This guarantees a 2px margin.

```
forceSimulation()  
  .force("collide", d3.forceCollide().radius(12) )
```



```
forceSimulation()  
  .force("x", d3.forceX(0) )  
  .force("collide", d3.forceCollide().radius(12) )
```



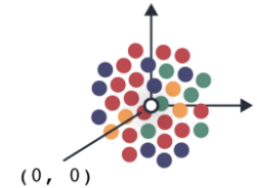
```
forceSimulation()  
  .force("y", d3.forceY(0) )  
  .force("collide", d3.forceCollide().radius(12) )
```



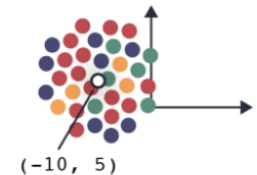
Centering Force

- The `d3.forceCenter()` moves a group of nodes toward a specific position.
- The positioning forces `forceX()` and `forceY()` move nodes individually and this can flatten the layout.
- Meanwhile, `forceCenter()` moves the whole system together, which keeps its original shape.

```
forceSimulation()  
  .force("collide", d3.forceCollide().radius(12) )
```



```
forceSimulation()  
  .force("center", forceCenter().x(-10).y(5) )  
  .force("collide", d3.forceCollide().radius(12) )
```



Many-body Force

- The `d3.forceManyBody()` mimics an attraction / repulsion force:

- A positive value means **attraction** 😊
- A negative value means **repulsion** 😞

```
forceSimulation()
```

```
forceSimulation()  
  .force("charge", forceManyBody().strength(-10))
```

```
forceSimulation()  
  .force("charge", forceManyBody().strength(10))
```



Links

- The `d3.forceLink()` applies a force between connected nodes.
- The more strongly two nodes are connected, the closer the link force will pull them together.
- The combination of all the forces we presented is usually used to create a `forceSimulation()`.

```
// add forces to the simulation
function initializeForces() {
  // add forces and associate each with a name
  simulation
    .force("link", d3.forceLink())
    .force("charge", d3.forceManyBody())
    .force("collide", d3.forceCollide())
    .force("center", d3.forceCenter())
    .force("forceX", d3.forceX())
    .force("forceY", d3.forceY());
  // apply properties to each of the forces
  updateForces();
}
```

Control the Simulation

- The force layout is designed to stop after the nodes are placed on their **optimal position**.
- However, we can use the following three functions to control a simulation:
 - `simulation.stop()`: used to stop a simulation. This is useful if there is an interaction between your network and another object on the page.
 - `simulation.restart()`: easy function to restart the simulation.
 - `simulation.tick()`: moves the layout of the simulation to the next step. We can also pre-compute the step of the simulation by setting the value of the parameter. So, the `simulation.tick(42)` will compute the 42nd iteration of the simulation.

Bibliography

- All the slides are based on “D3.js in Action, Third Edition” book;
- [MDN](#) for anything related the Web;
- D3 documentation.