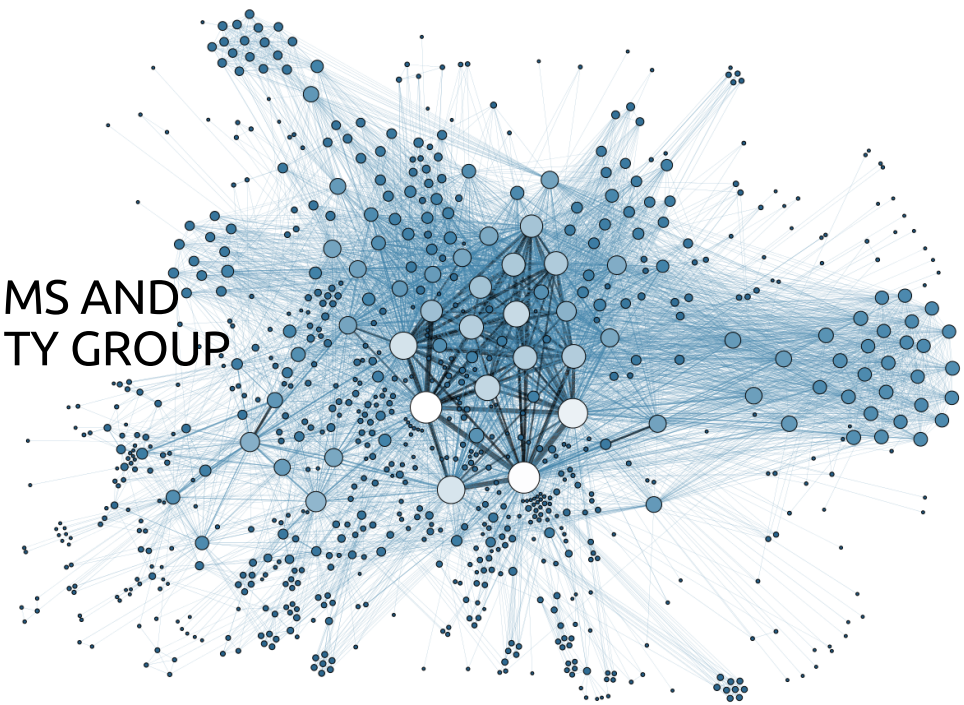
Force- and Stress-Based Graph Drawing

Lecture Graph Drawing Algorithms · 192.053

Martin Nöllenburg

08.05.2018



TU WIEN

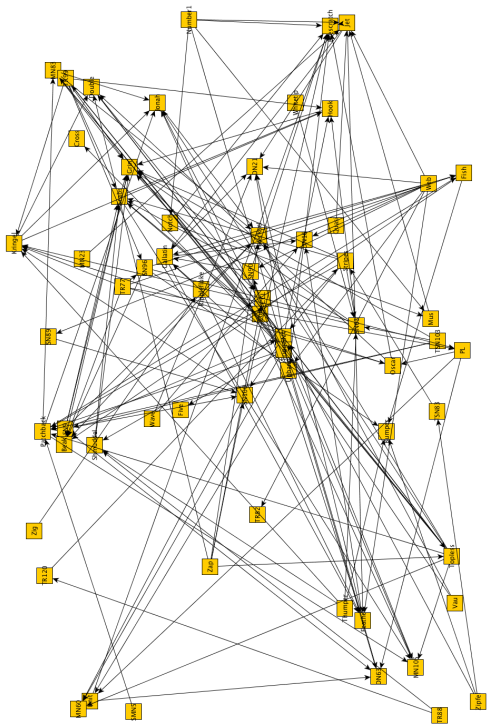
ALGORITHMS AND
COMPLEXITY GROUP

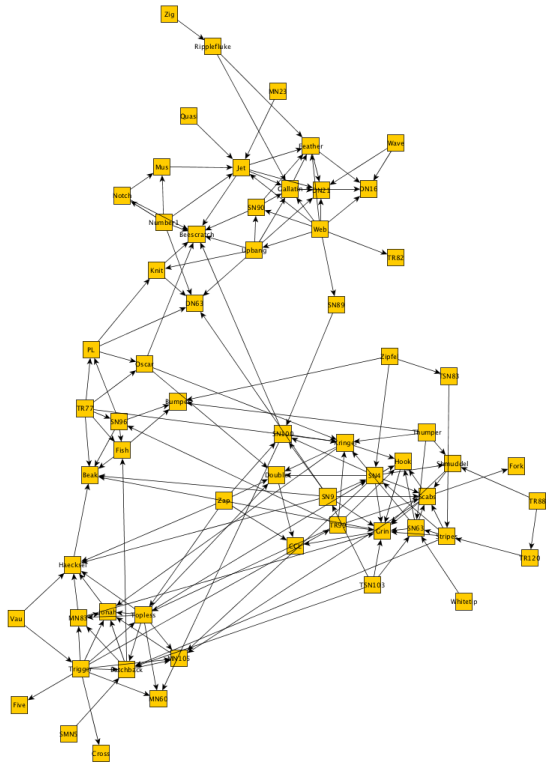
Drawing Arbitrary Graphs

In the previous lectures we studied graphs with structural properties (trees, series-parallel, planar).
What if we don't know the structure of our input graph?

What is a minimal set of constraints and criteria?

Assumptions: simple, undirected graph





- few edge crossings
- maximize minimum angle
- short, uniform edge lengths
- good use of screen space, distribute vertices
- show grouping structure
- adjacent vertices close, non-adj. farther

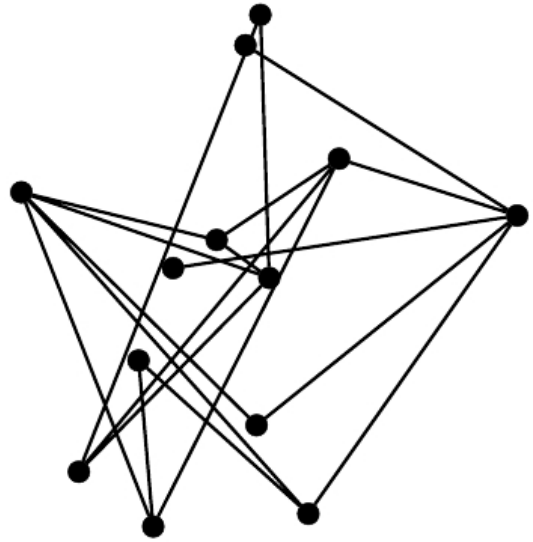
Overview

Force-based graph layout

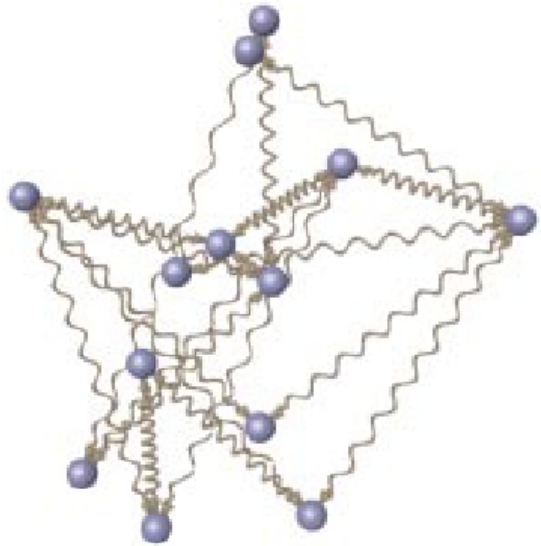
Scalability and faster force computation

Stress-based graph layout

Physical Model

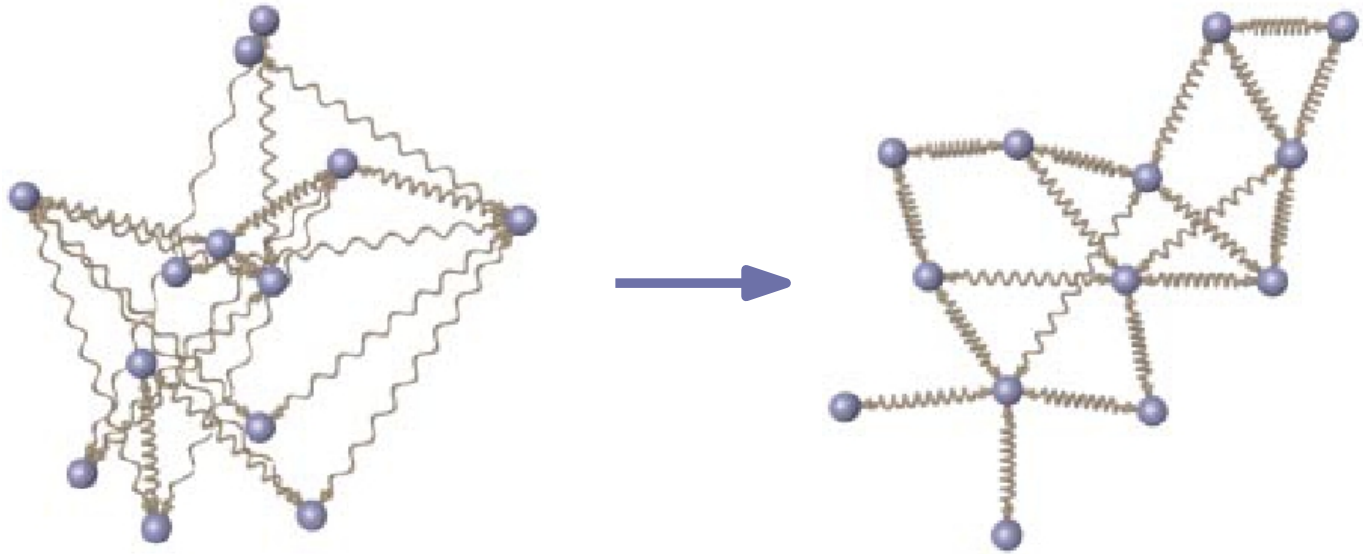


Physical Model



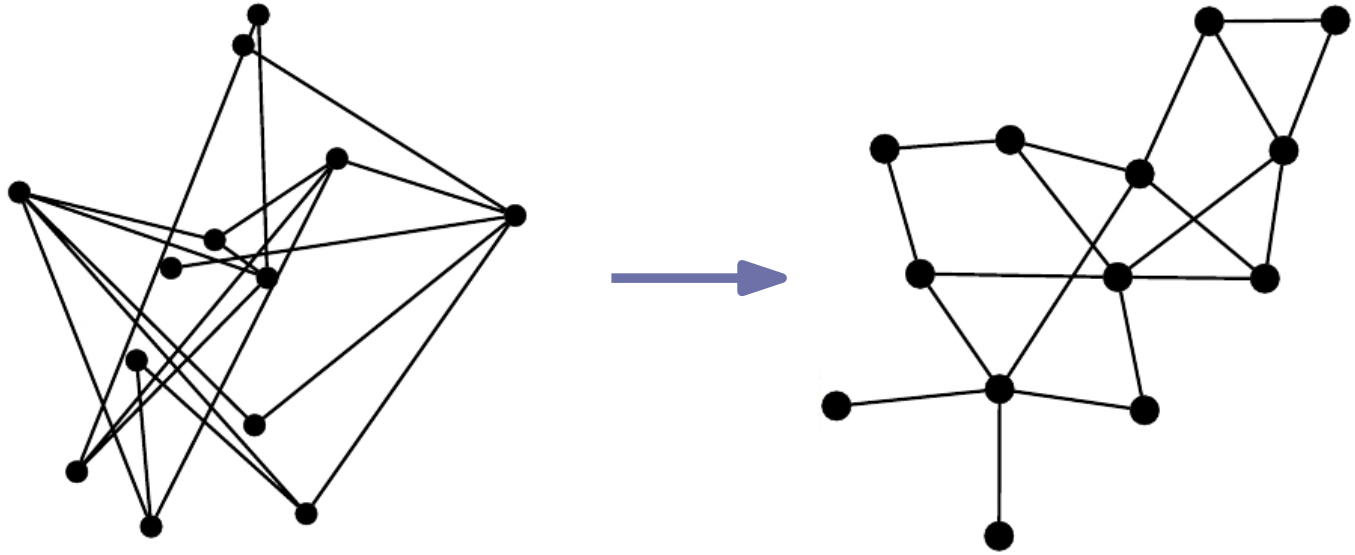
[Eades, 1984] “To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . .

Physical Model



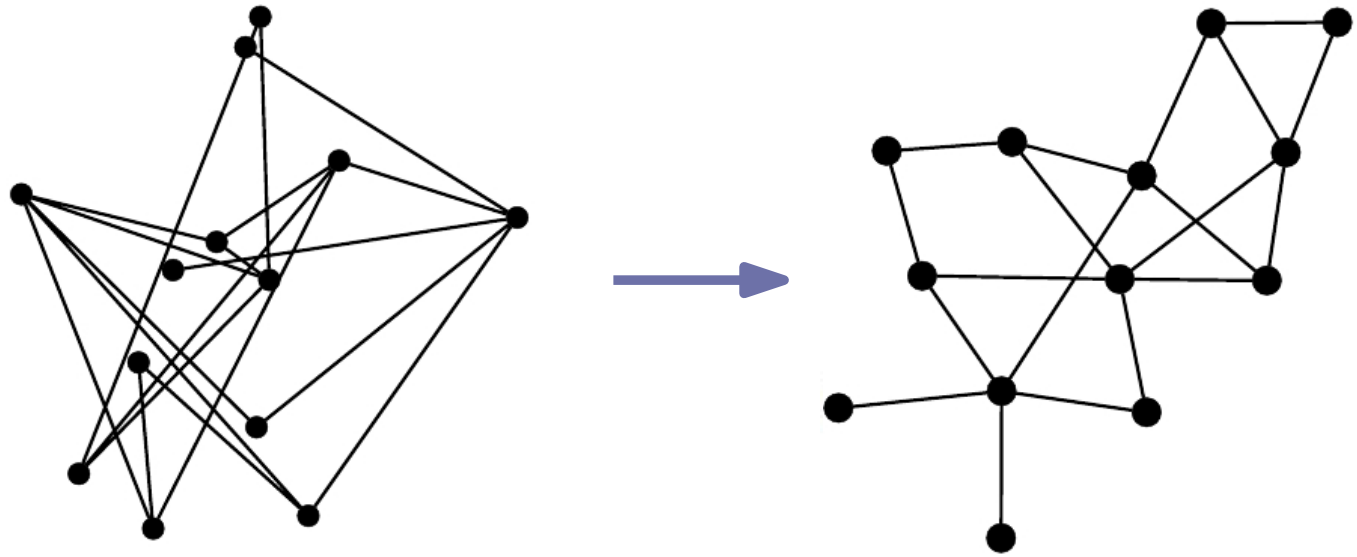
[Eades, 1984] "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . .
The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state."

Physical Model

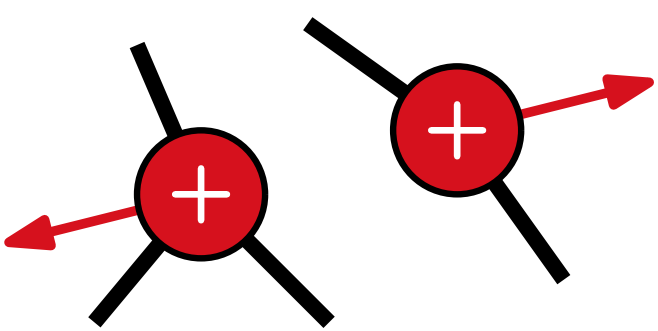


[Eades, 1984] "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . .
The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state."

Physical Model

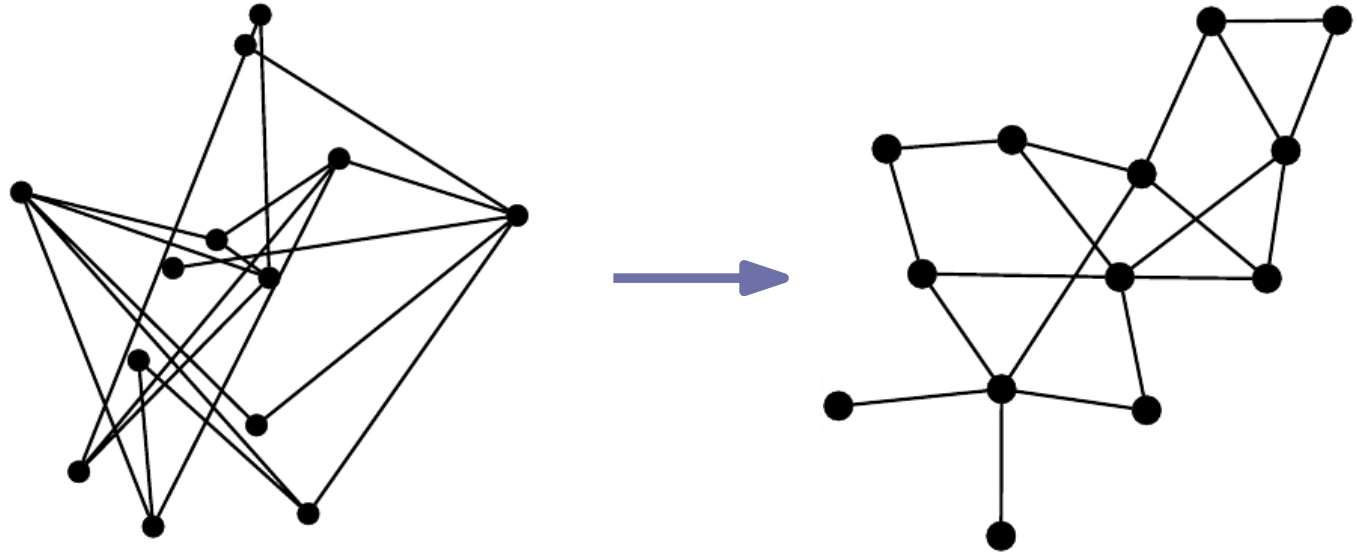


[Eades, 1984] "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . .
The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state."



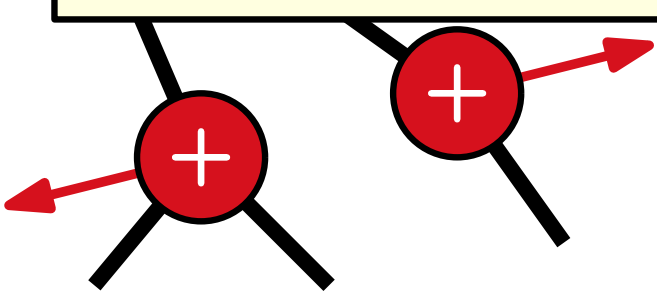
In addition consider (non-adjacent) vertices as electrically charged particles that repel each other.

Physical Model



[Eades, 1984] "To embed a graph we replace the vertices by steel rings and r
f

So-called **spring-embedder** or **spring-electrical** algorithms that work according to this or similar principles are among the most frequently used graph-drawing methods in practice.



In addition consider (non-adjacent) vertices as electrically charged particles that repel each other.

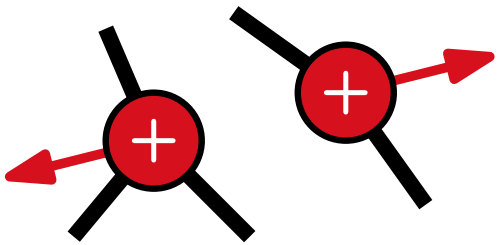
Notation

$\ell = \ell(e)$	ideal spring length for edge e
$p_v = (x_v, y_v)$	position of node v
$\|\|p_u - p_v\|\|$	Euclidean distance between u and v
$\overrightarrow{p_u p_v}$	unit vector pointing from u to v

Spring-Embedder (Eades, 1984)

Model:

- repulsive force between two non-adjacent vertices u and v

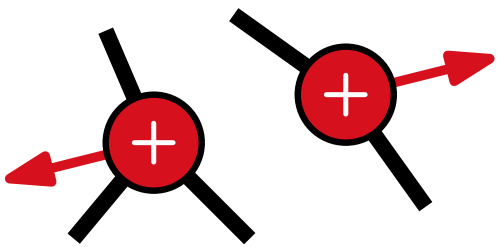


$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

Spring-Embedder (Eades, 1984)

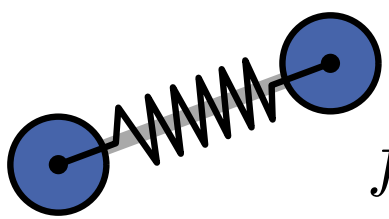
Model:

- repulsive force between two non-adjacent vertices u and v



$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

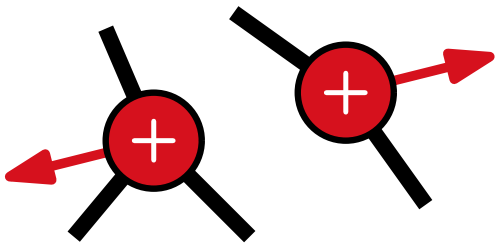


$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

Spring-Embedder (Eades, 1984)

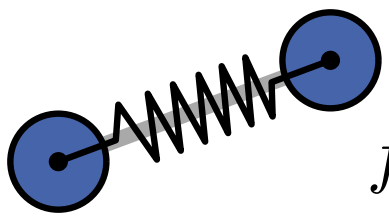
Model:

- repulsive force between two non-adjacent vertices u and v



$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

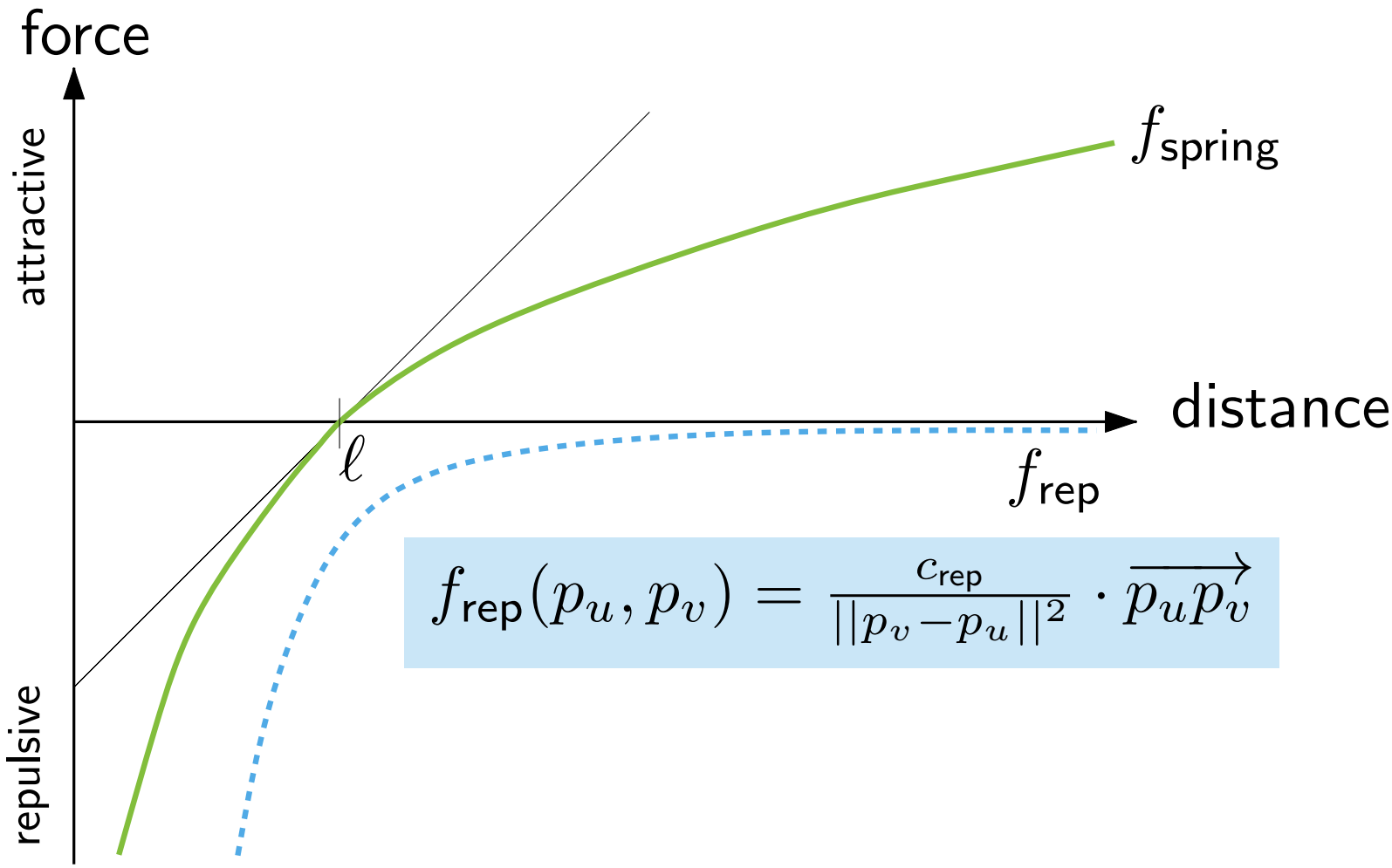


$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

- resulting displacement vector for vertex v

$$F_v = \sum_{uv \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{uv \in E} f_{\text{spring}}(p_u, p_v)$$

Diagram of Spring-Embedder Forces (Eades, 1984)



$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, threshold $\varepsilon > 0$, constant $\delta > 0$

Output: Layout p with low energy

$t \leftarrow 1$

while $t < K$ **and** $\max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**

  foreach $v \in V$ **do**

    $F_v(t) \leftarrow \sum_{uv \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{uv \in E} f_{\text{spring}}(p_u, p_v)$

  foreach $v \in V$ **do**

    $p_v \leftarrow p_v + \delta \cdot F_v(t)$

  $t \leftarrow t + 1$

Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, threshold $\varepsilon > 0$, constant $\delta > 0$

Output: Layout p with low energy

$t \leftarrow 1$

while $t < K$ **and** $\max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**

  foreach $v \in V$ **do**

    $F_v(t) \leftarrow \sum_{uv \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{uv \in E} f_{\text{spring}}(p_u, p_v)$

  foreach $v \in V$ **do**

    $p_v \leftarrow p_v + \delta \cdot F_v(t)$

  $t \leftarrow t + 1$

Demo

Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, threshold $\varepsilon > 0$, constant $\delta > 0$

Output: Layout p with low energy

$t \leftarrow 1$

while $t < K$ **and** $\max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**

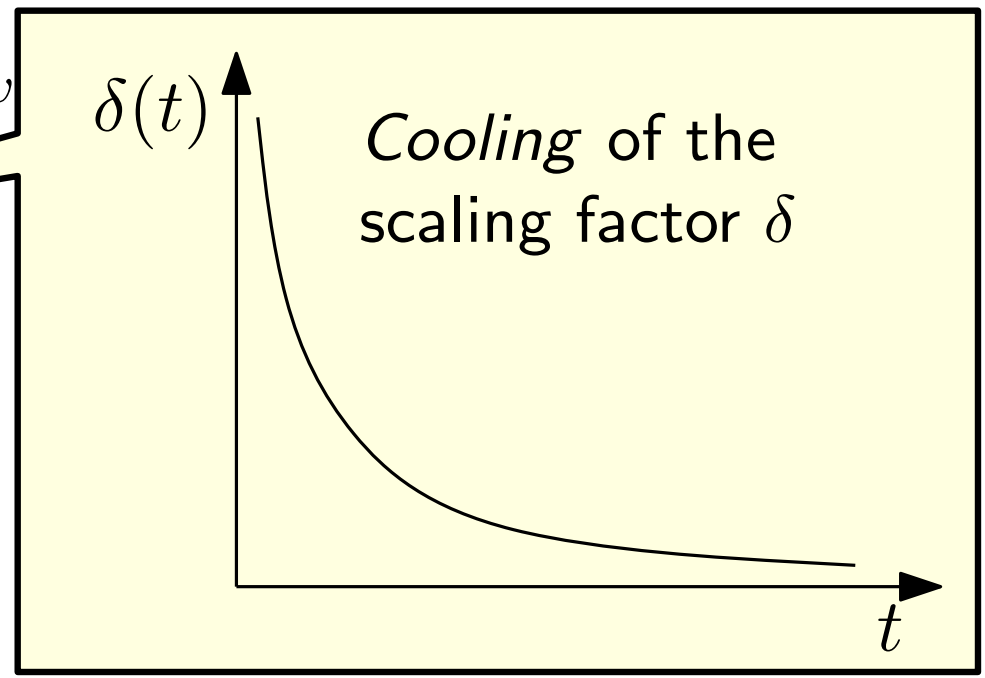
  foreach $v \in V$ **do**

    $F_v(t) \leftarrow \sum_{uv \notin E} f_{\text{rep}}(p_u, p_v$

  foreach $v \in V$ **do**

    $p_v \leftarrow p_v + \delta(t) \cdot F_v(t)$

  $t \leftarrow t + 1$



Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, threshold $\varepsilon > 0$, constant $\delta > 0$

Output: Layout p with low energy

$t \leftarrow 1$

while $t < K$ **and** $\max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**

  foreach $v \in V$ **do**

    $F_v(t) \leftarrow \sum_{uv \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{uv \in E} f_{\text{spring}}(p_u, p_v)$

  foreach $v \in V$ **do**

    $p_v \leftarrow p_v + \delta(t) \cdot F_v(t)$

  $t \leftarrow t + 1$

$O(|V|^2)$ $O(|E|)$

What is the running time of the algorithm?

Discussion

Advantages

- very simple and easy-to-implement algorithm
- good results for small to medium-sized graphs
- empirically good representation of symmetries and structure

Discussion

Advantages

- very simple and easy-to-implement algorithm
- good results for small to medium-sized graphs
- empirically good representation of symmetries and structure

Disadvantages

- system is not stable at the end
- converging to local minima
- computing time: f_{spring} in $\mathcal{O}(|E|)$ and f_{rep} in $\mathcal{O}(|V|^2)$

Discussion

Advantages

- very simple and easy-to-implement algorithm
- good results for small to medium-sized graphs
- empirically good representation of symmetries and structure

Disadvantages

- system is not stable at the end
- converging to local minima
- computing time: f_{spring} in $\mathcal{O}(|E|)$ and f_{rep} in $\mathcal{O}(|V|^2)$

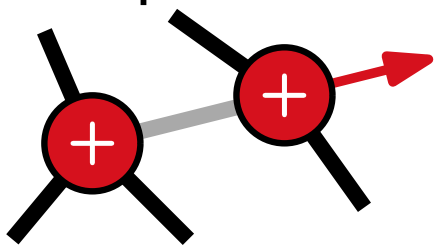
Influence

- original paper by Peter Eades has 1819 citations
- basis for many subsequent ideas and algorithms

Variant: Fruchterman & Reingold (1991)

Model:

- repulsive force between **all** vertex pairs u and v

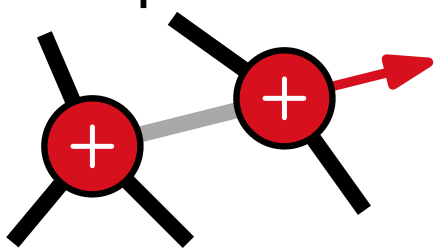


$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

Variant: Fruchterman & Reingold (1991)

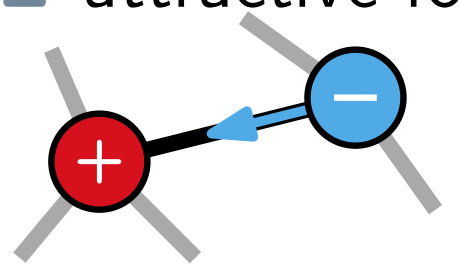
Model:

- repulsive force between **all** vertex pairs u and v



$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

- attractive force between two adjacent vertices u and v

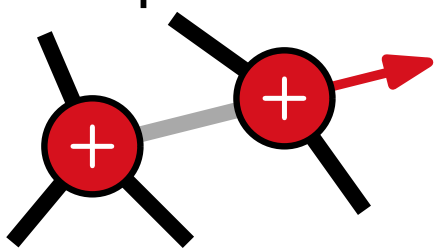


$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

Variant: Fruchterman & Reingold (1991)

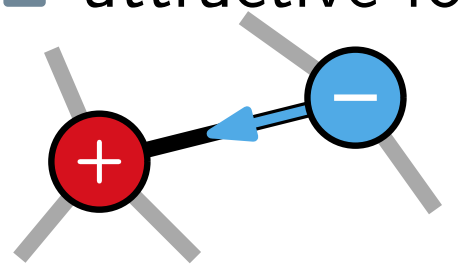
Model:

- repulsive force between **all** vertex pairs u and v



$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

- attractive force between two adjacent vertices u and v

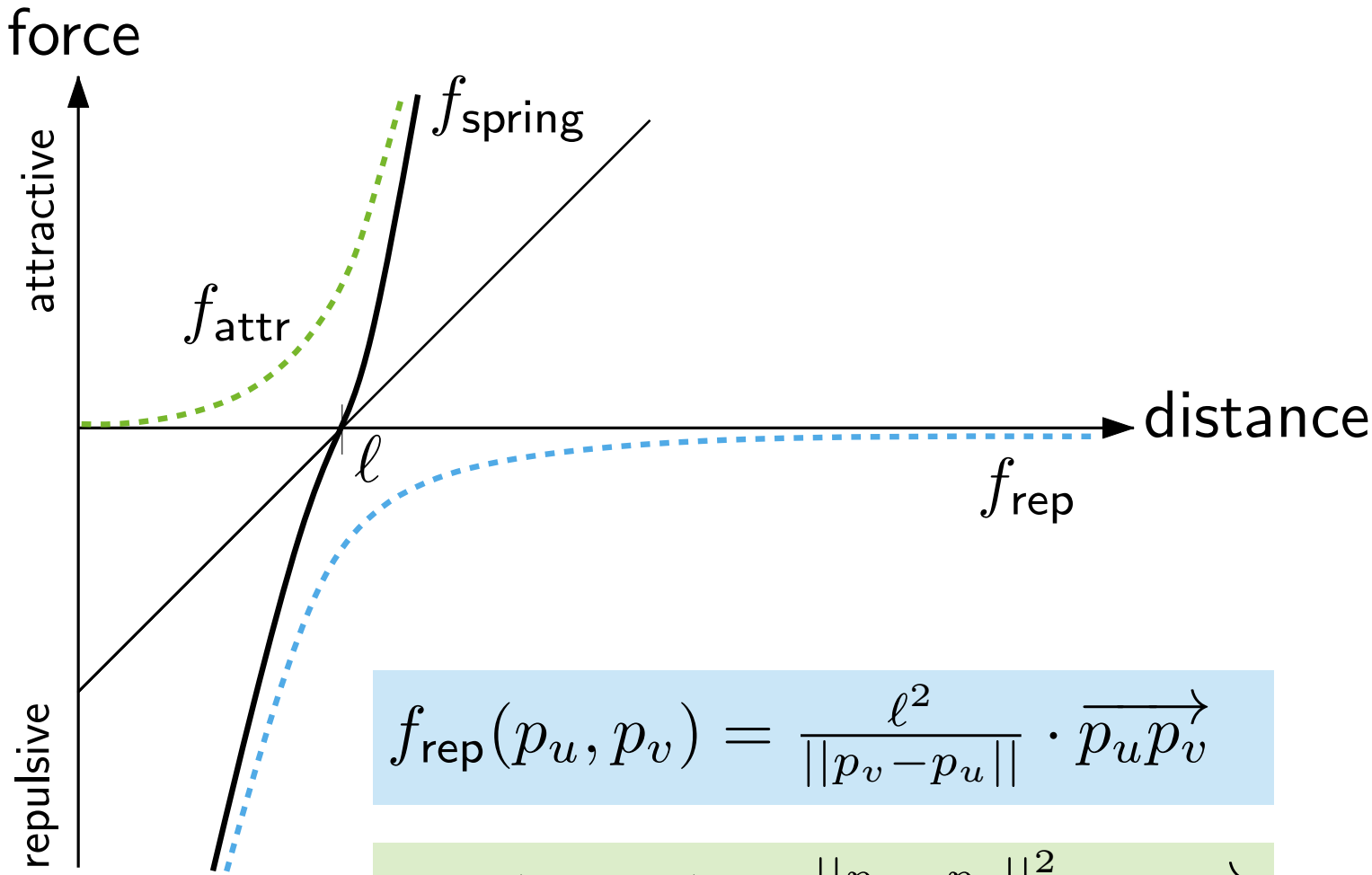


$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

- resulting force between adjacent vertices u and v

$$f_{\text{spring}}(p_u, p_v) = f_{\text{rep}}(p_u, p_v) + f_{\text{attr}}(p_u, p_v)$$

Diagram of Fruchtermann & Reingold Forces

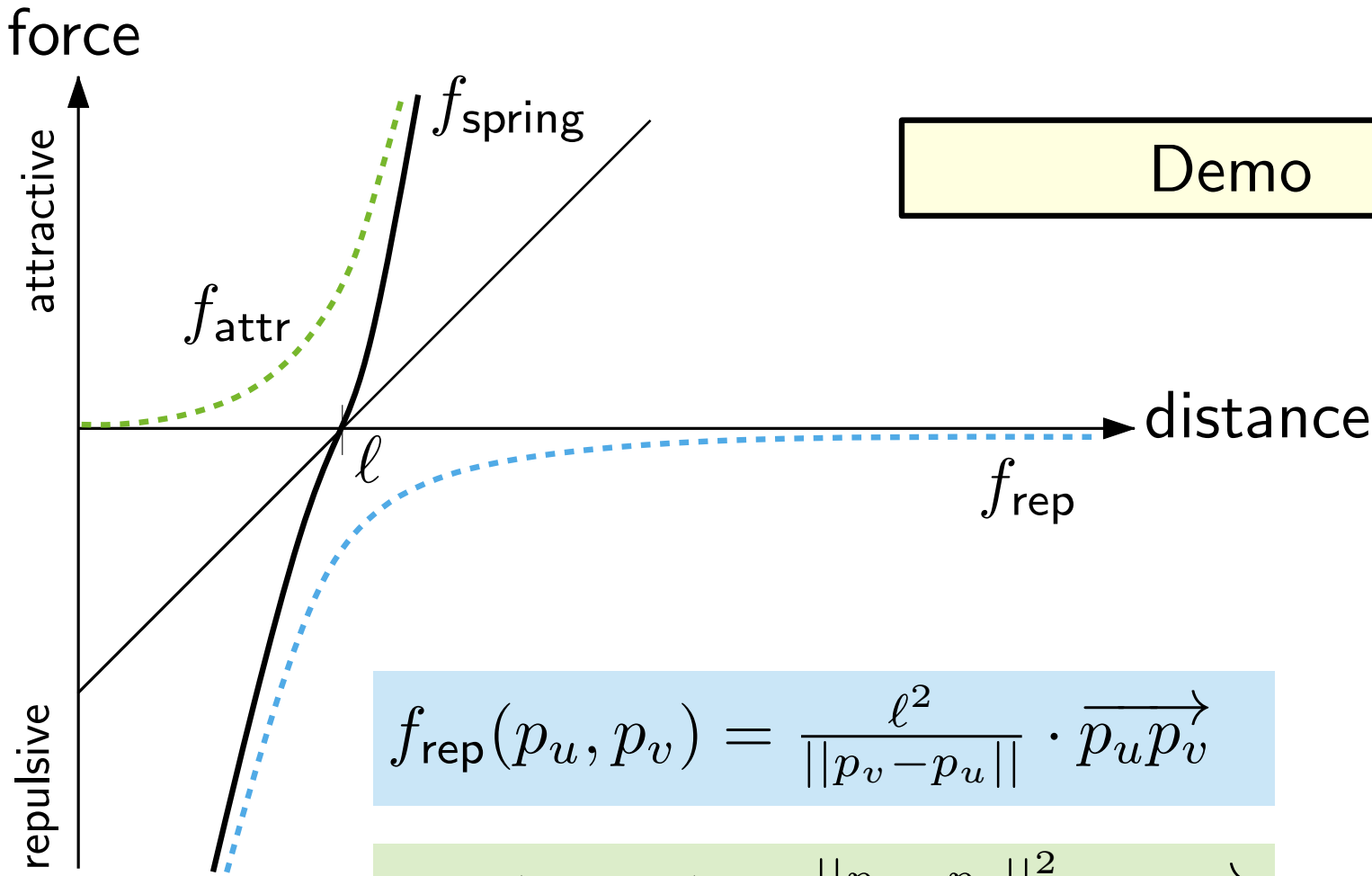


$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

$$f_{\text{spring}}(p_u, p_v) = f_{\text{rep}}(p_u, p_v) + f_{\text{attr}}(p_u, p_v)$$

Diagram of Fruchtermann & Reingold Forces



Demo

$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

$$f_{\text{spring}}(p_u, p_v) = f_{\text{rep}}(p_u, p_v) + f_{\text{attr}}(p_u, p_v)$$

Other Possible Modifications

- **Inertia**
- **Gravitation**
- **Magnetic forces**

Other Possible Modifications

- **Inertia**

  define vertex mass as $\Phi(v) = 1 + \deg(v)/2$

  set $f_{\mathsf{attr}}(p_u, p_v) \leftarrow f_{\mathsf{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

- **Gravitation**

- **Magnetic forces**

Other Possible Modifications

- **Inertia**

  define vertex mass as $\Phi(v) = 1 + \deg(v)/2$

  set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

- **Gravitation**

  define barycenter $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$

  $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$

- **Magnetic forces**

Other Possible Modifications

- **Inertia**

  define vertex mass as $\Phi(v) = 1 + \deg(v)/2$

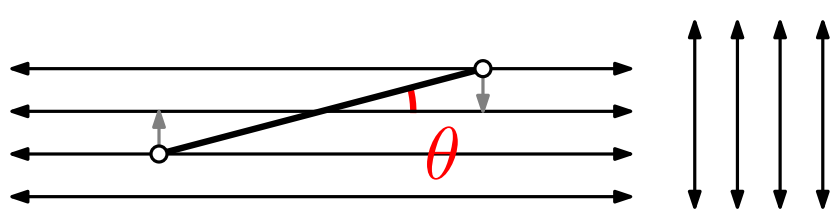
  set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

- **Gravitation**

  define barycenter $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$

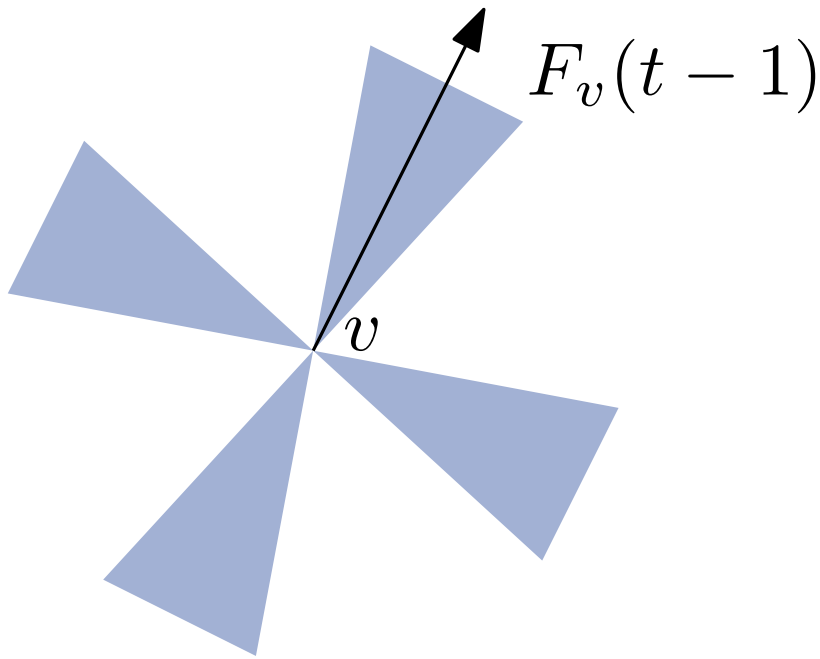
  $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$

- **Magnetic forces**
  - define magnetic fields (e.g. vertical, horizontal)
  - angle θ between edge and the direction of the field
  - define force that reduces this angle

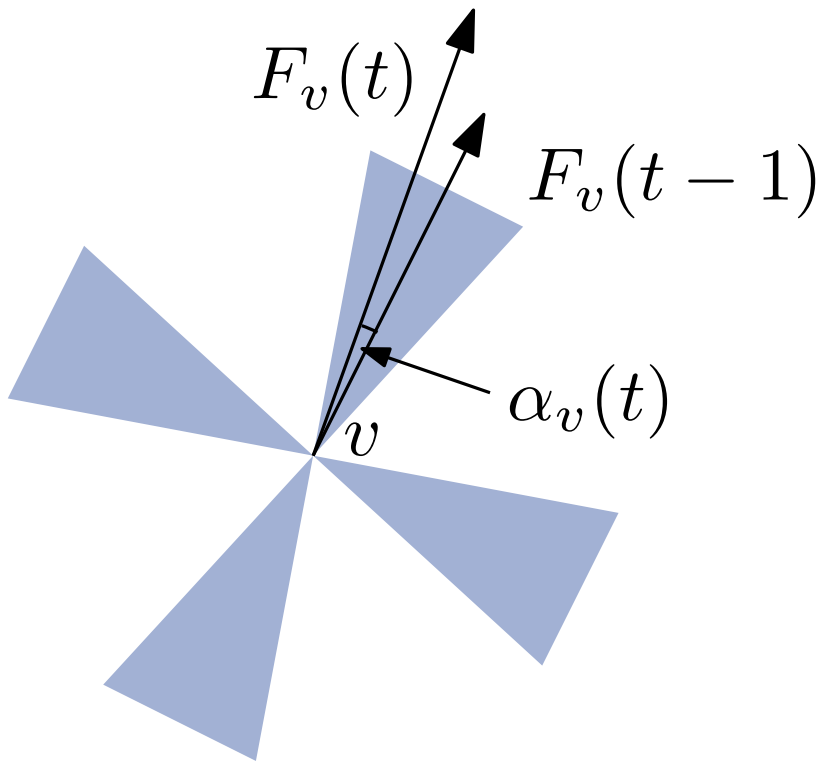


Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

- store previous displacement vector $F_v(t-1)$



Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

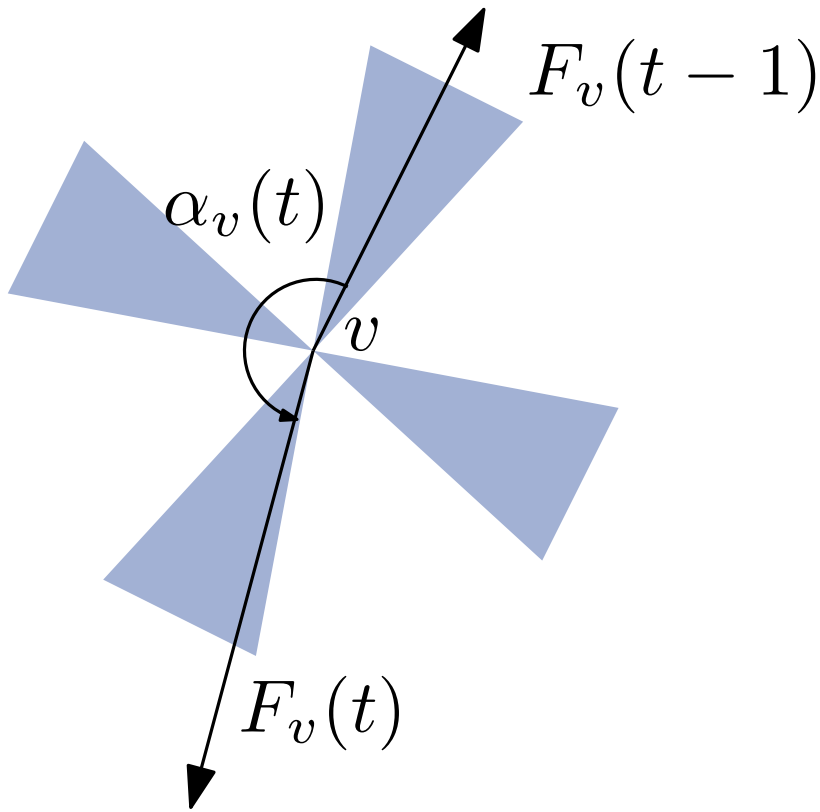


- store previous displacement vector $F_v(t-1)$

local temperature

- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

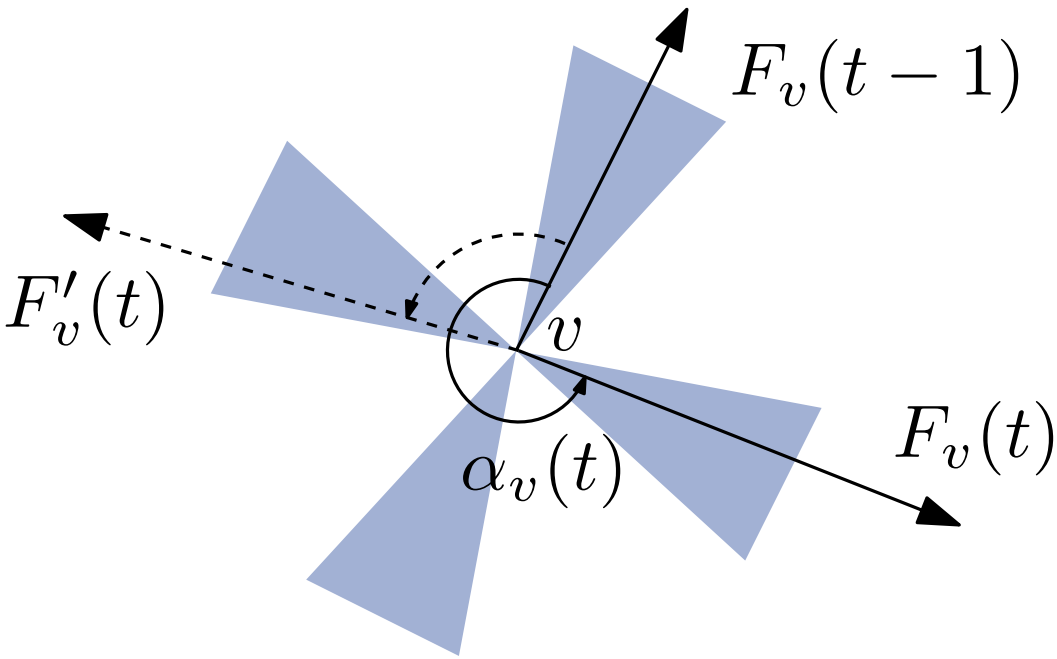


- store previous displacement vector $F_v(t-1)$

local temperature

- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature
- $\cos(\alpha_v(t)) \approx -1$:
  oscillation
  $\rightarrow$ reduce the temperature

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)



- store previous displacement vector $F_v(t-1)$

local temperature

- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature
- $\cos(\alpha_v(t)) \approx -1$:
  oscillation
  $\rightarrow$ reduce the temperature
- $\cos(\alpha_v(t)) \approx 0$:
  rotation if it occurs repeatedly
  $\rightarrow$ reduce the temperature

Discussion

Advantages

- algorithm remains simple and implementation friendly
- additional modifications improve layout quality and lead to faster convergence

Discussion

Advantages

- algorithm remains simple and implementation friendly
- additional modifications improve layout quality and lead to faster convergence

Disadvantages

- stability is not guaranteed
- local minima possible
- quadratic time for repulsive forces

Discussion

Advantages

- algorithm remains simple and implementation friendly
- additional modifications improve layout quality and lead to faster convergence

Disadvantages

- stability is not guaranteed
- local minima possible
- quadratic time for repulsive forces

Influence

- Variants of the Fruchterman & Reingold algorithm are probably the most popular force-based methods (paper cited 4350 times)

Discussion

Advantages

- algorithm remains simple and implementation friendly
- additional modifications improve layout quality and lead to faster convergence

Disadvantages

- stability is not guaranteed
- local minima possible
- quadratic time for repulsive forces

Can we reduce this?

Influence

- Variants of the Fruchterman & Reingold algorithm are probably the most popular force-based methods (paper cited 4350 times)

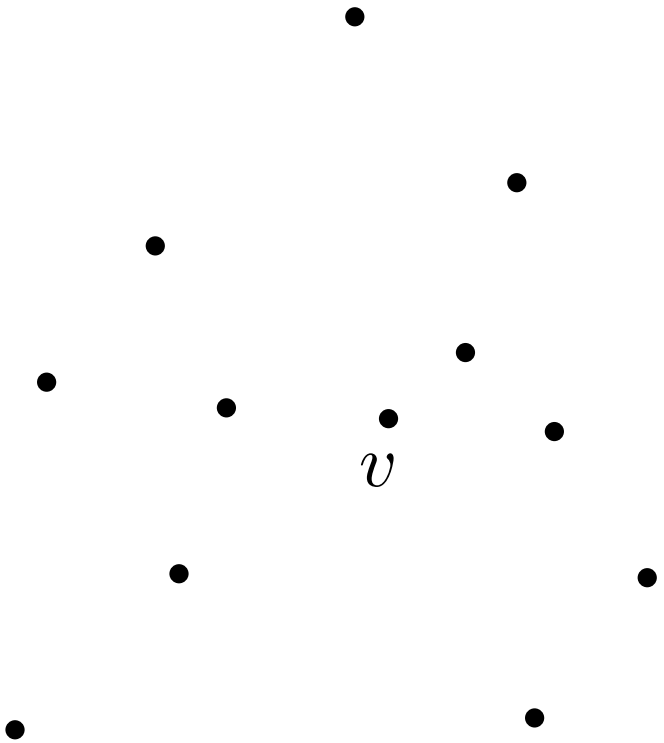
Overview

Force-based graph layout

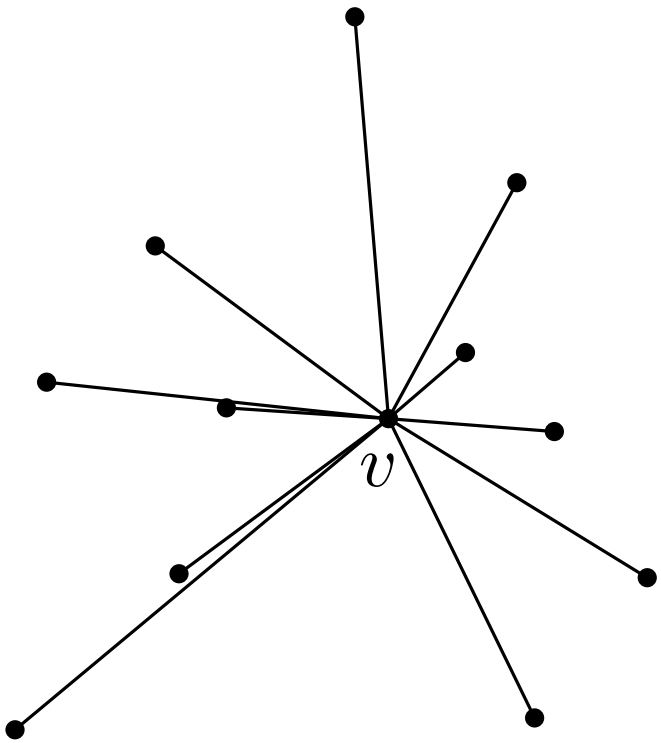
Scalability and faster force computation

Stress-based graph layout

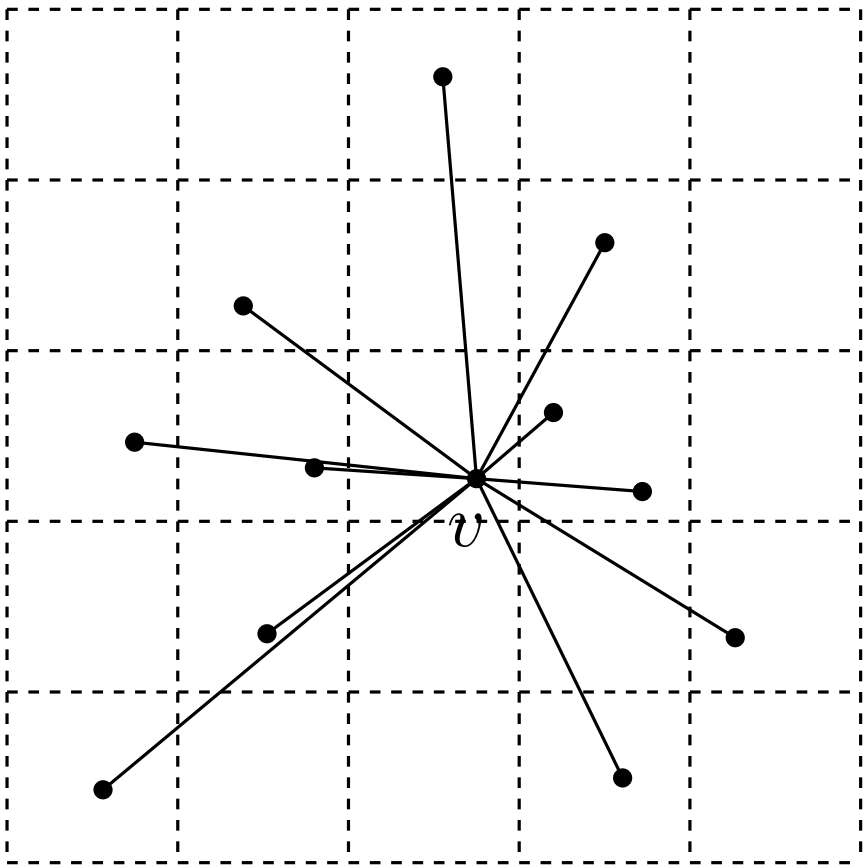
Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)



Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)

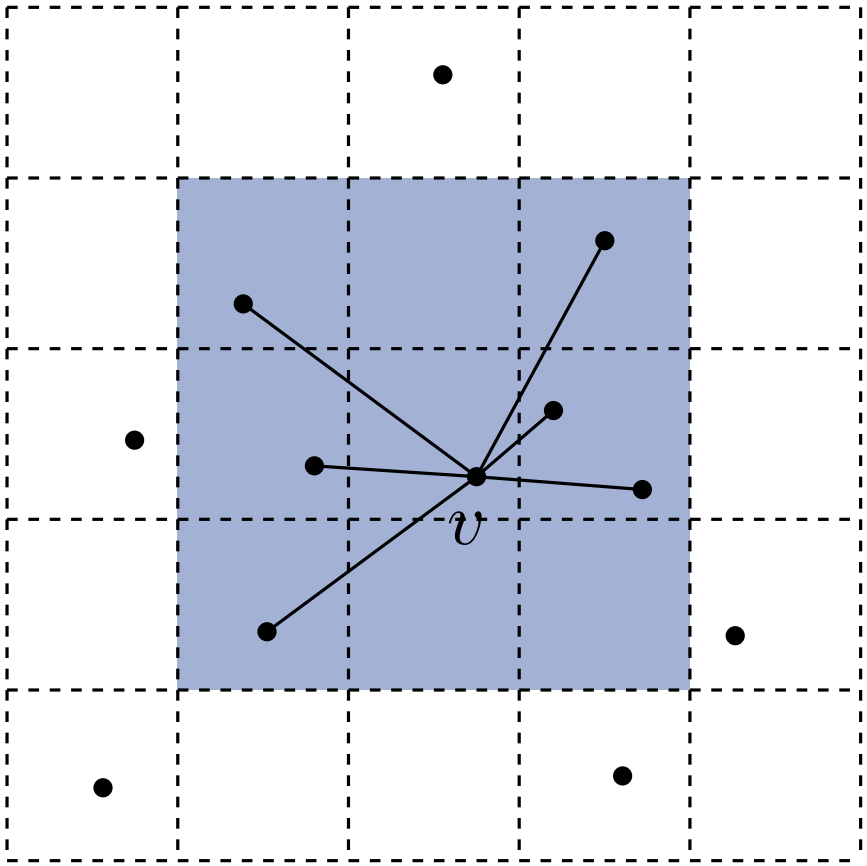


Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)



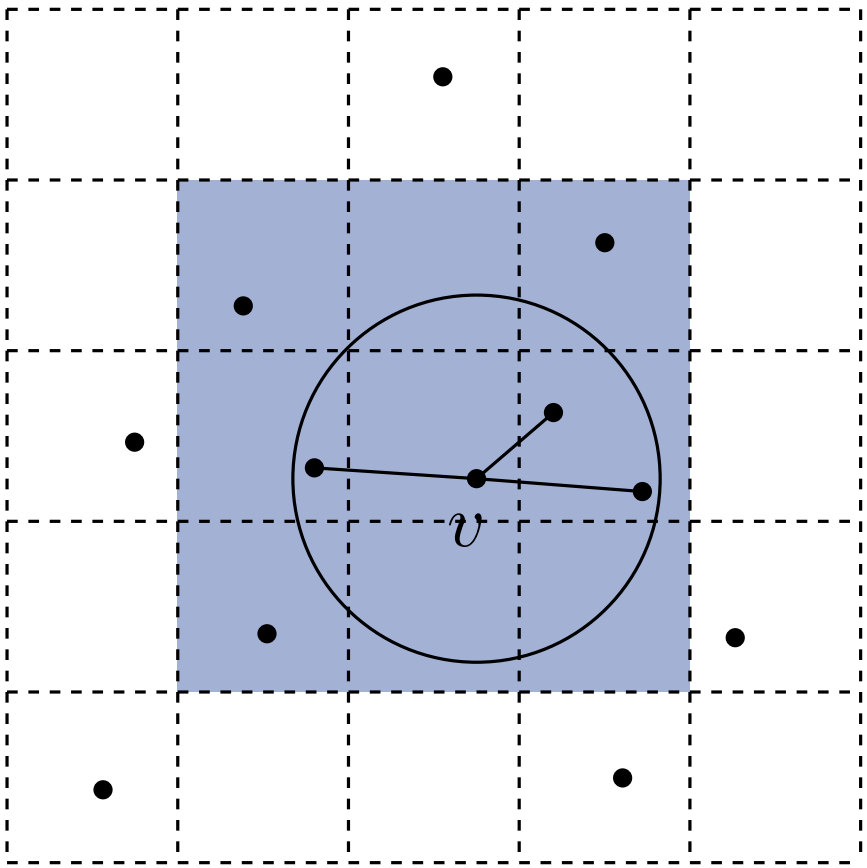
- subdivide plane by a grid

Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)



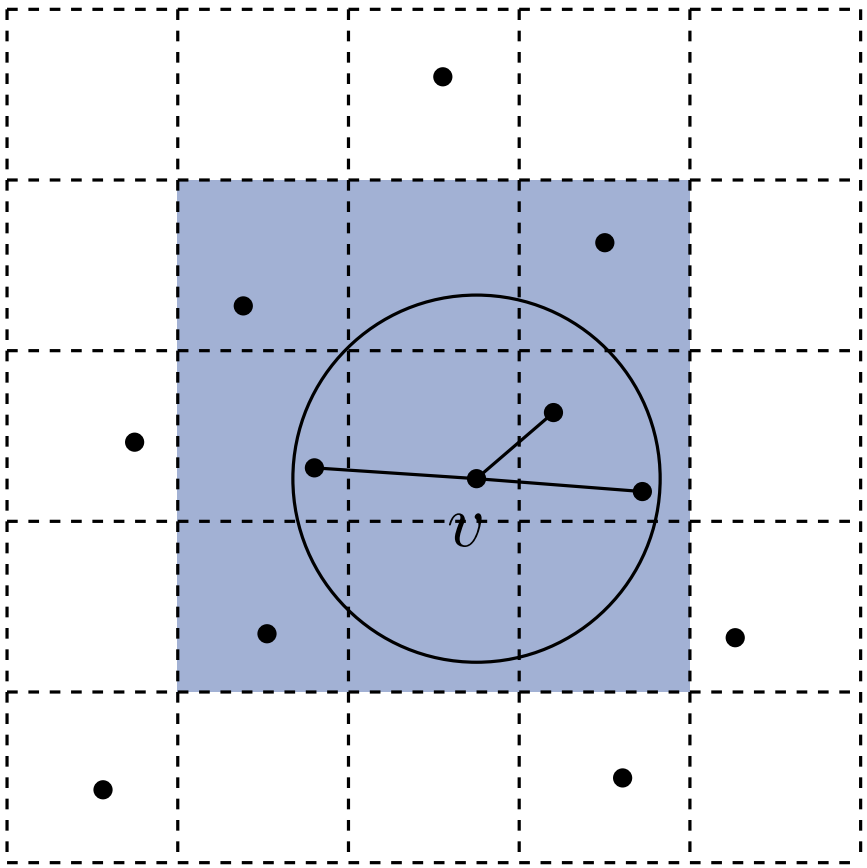
- subdivide plane by a grid
- compute repulsive forces only for vertices in neighbouring cells

Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)



- subdivide plane by a grid
- compute repulsive forces only for vertices in neighbouring cells
- and only when the distance is at most $d_{\max}$

Local Repulsion: Grid Version (Fruchterman, Reingold, 1990)



- subdivide plane by a grid
- compute repulsive forces only for vertices in neighbouring cells
- and only when the distance is at most $d_{\max}$

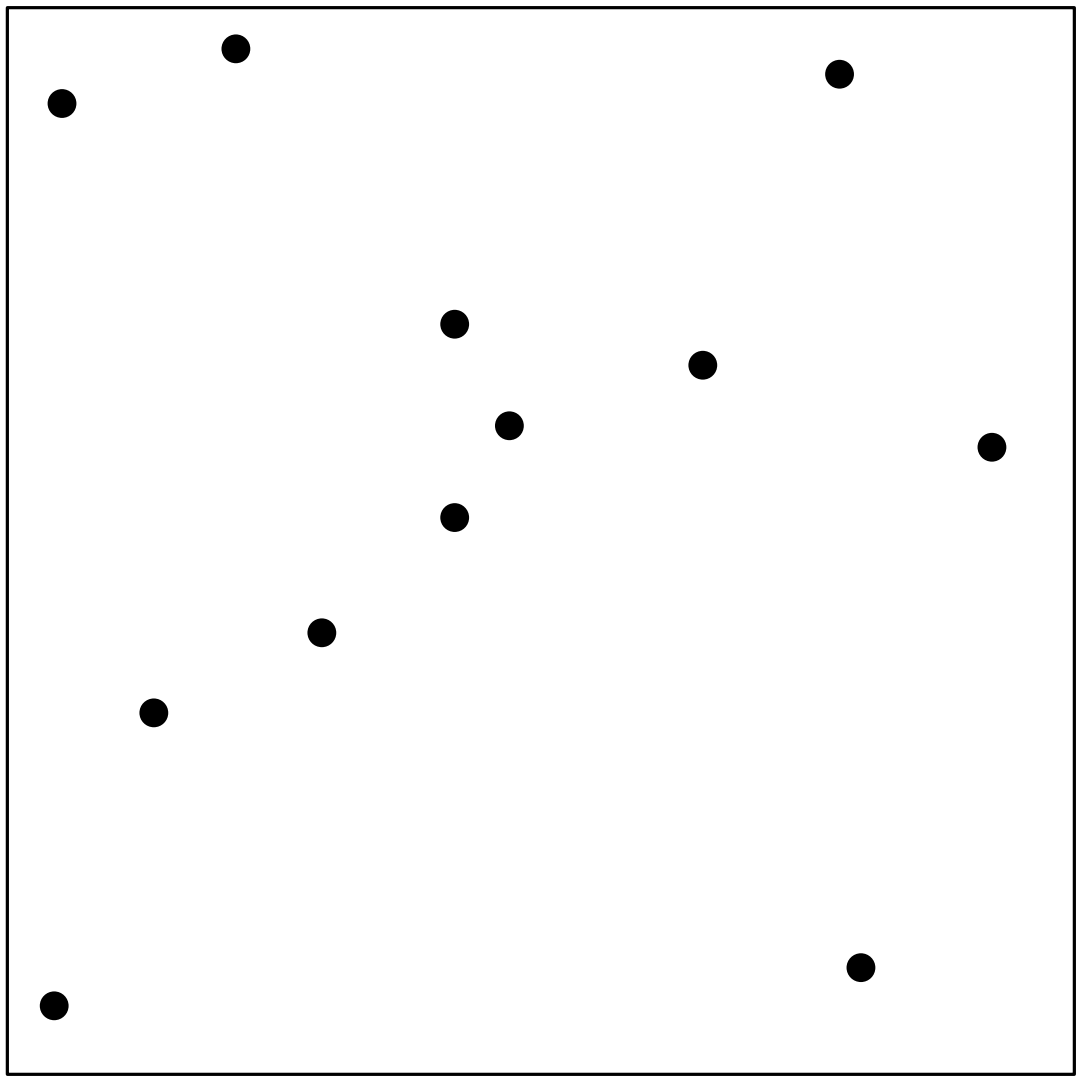
Discussion

- useful idea to improve runtime
- worst-case no improvement
- quality loss (e.g. oscillation around $d_{\max}$)

Quadtree Data Structure

Quadtrees store point sets in $\mathbb{R}^2$ in a hierarchical, geometric data structure

- if square has ≥ 2 points, then split into four subsquares and recurse



R_0

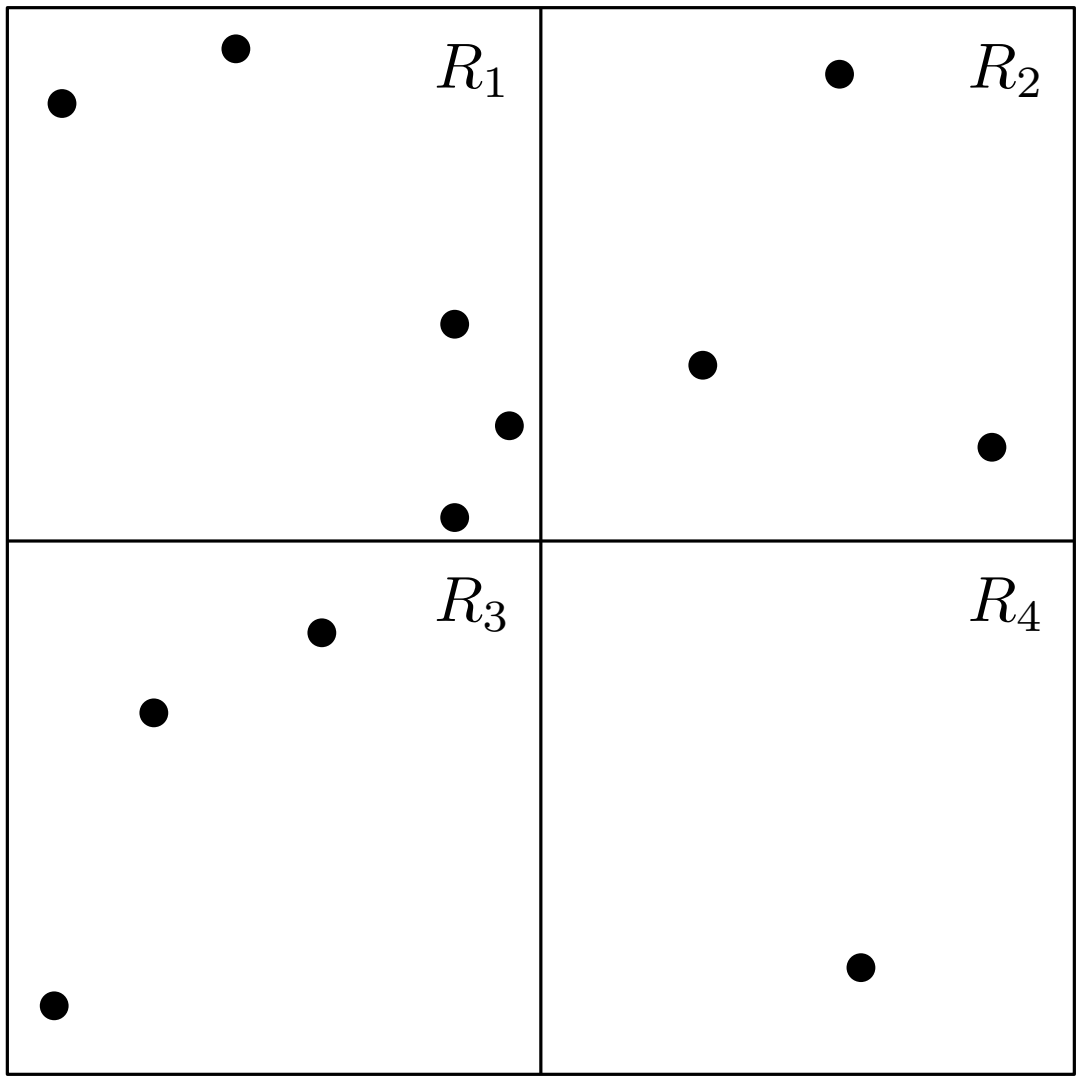
QT

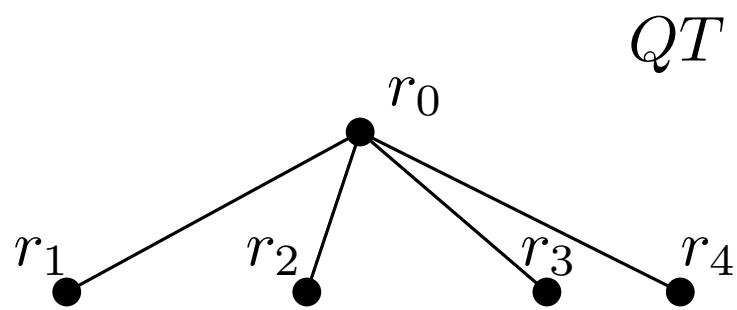
r_0

Quadtree Data Structure

Quadtrees store point sets in $\mathbb{R}^2$ in a hierarchical, geometric data structure

- if square has ≥ 2 points, then split into four subsquares and recurse

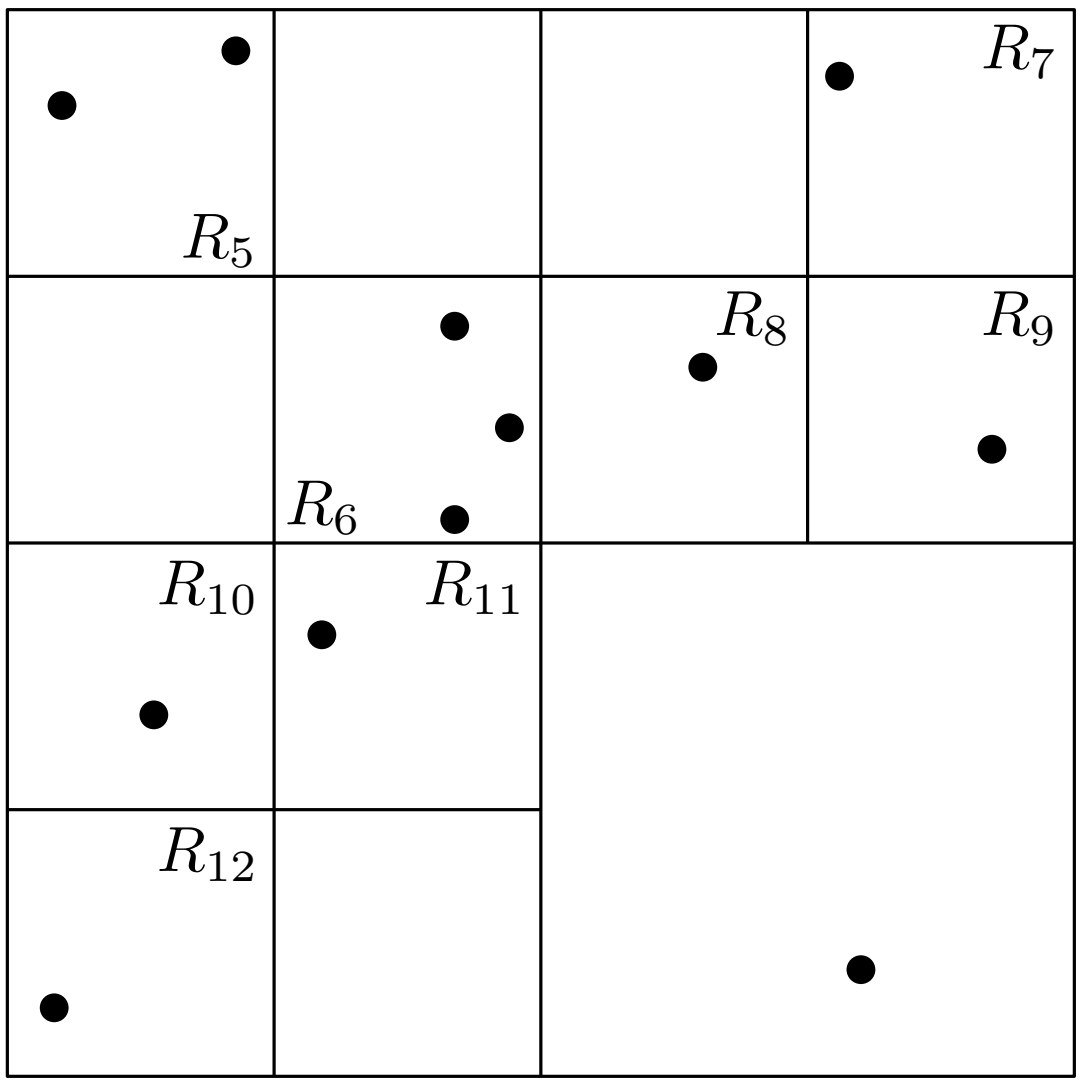


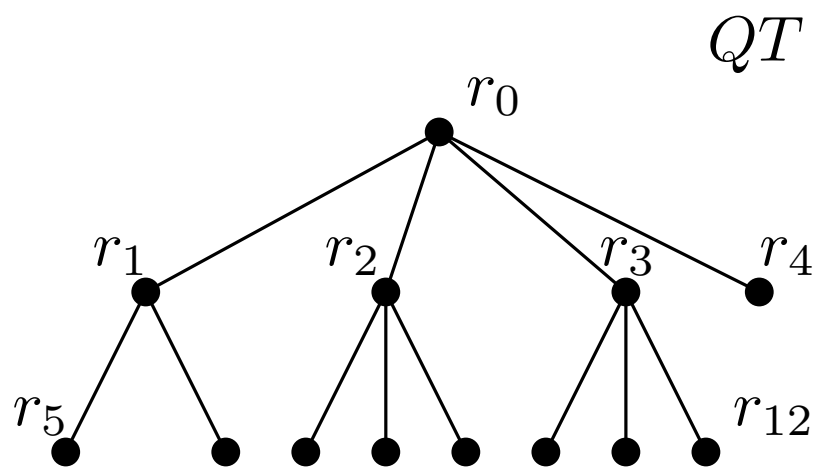


Quadtree Data Structure

Quadtrees store point sets in $\mathbb{R}^2$ in a hierarchical, geometric data structure

- if square has ≥ 2 points, then split into four subsquares and recurse

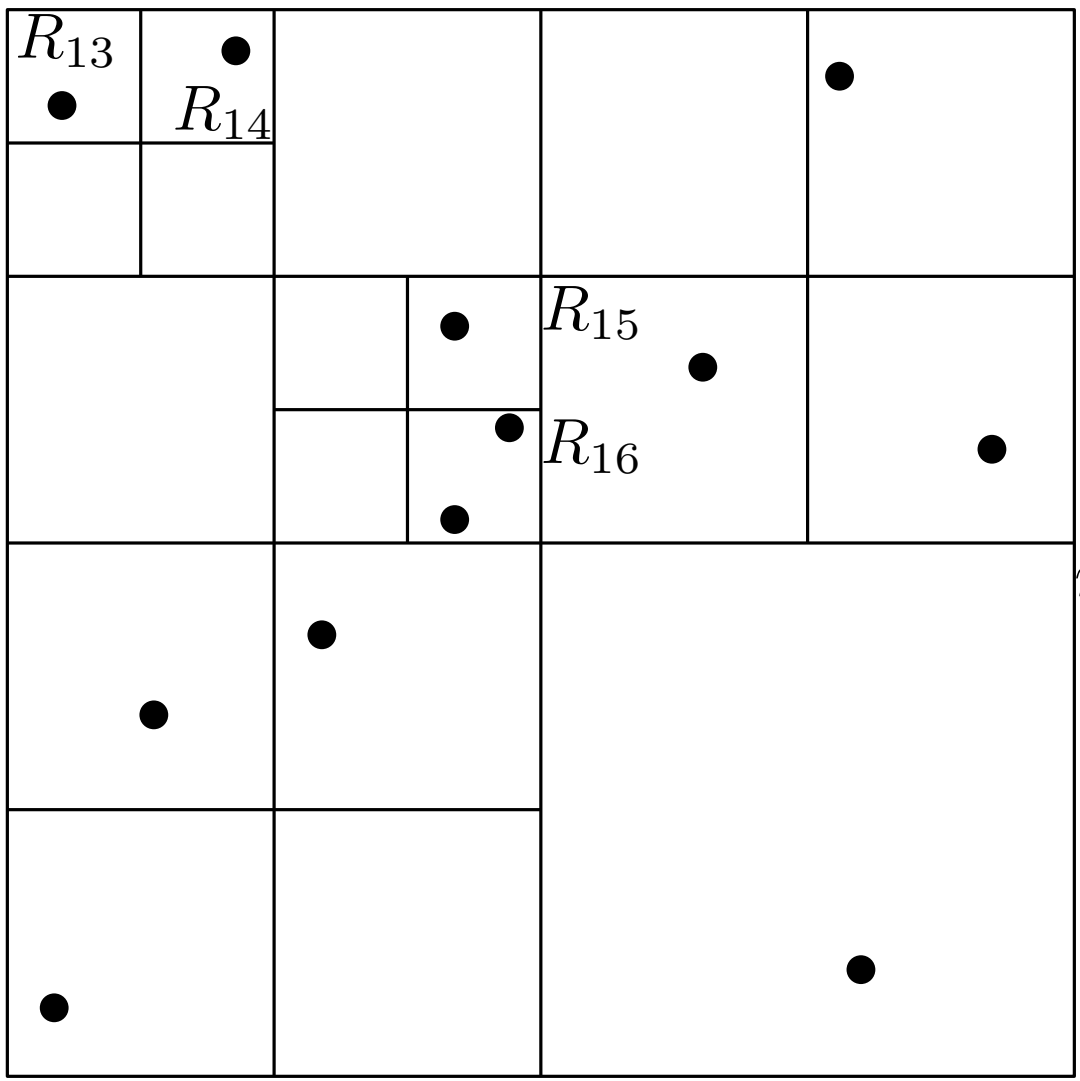


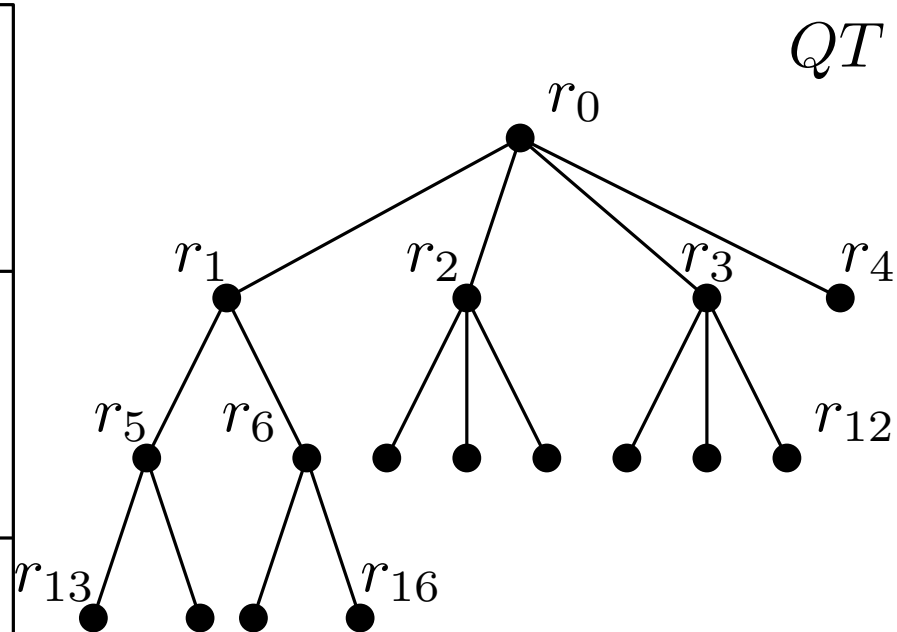


Quadtree Data Structure

Quadtrees store point sets in $\mathbb{R}^2$ in a hierarchical, geometric data structure

- if square has ≥ 2 points, then split into four subsquares and recurse

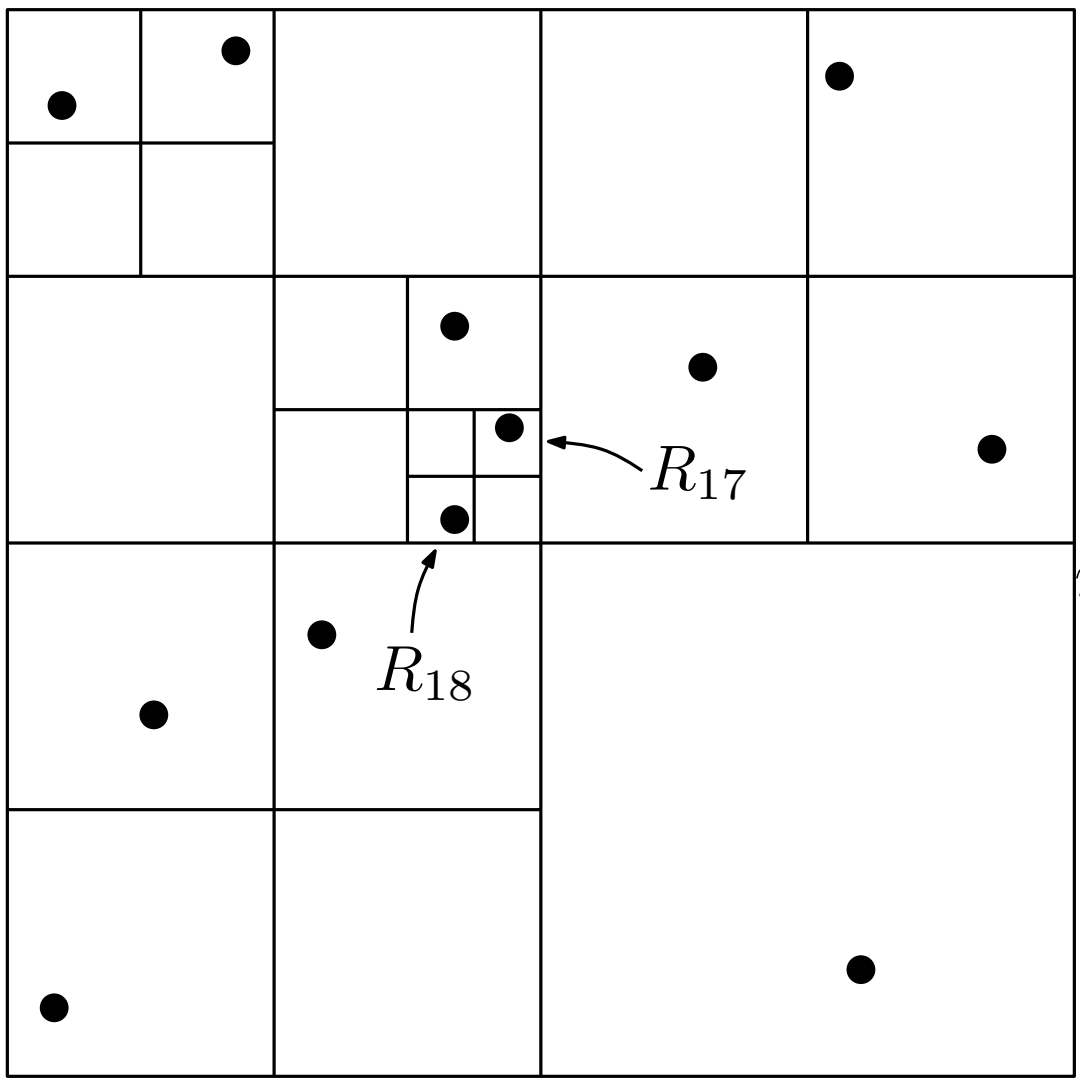


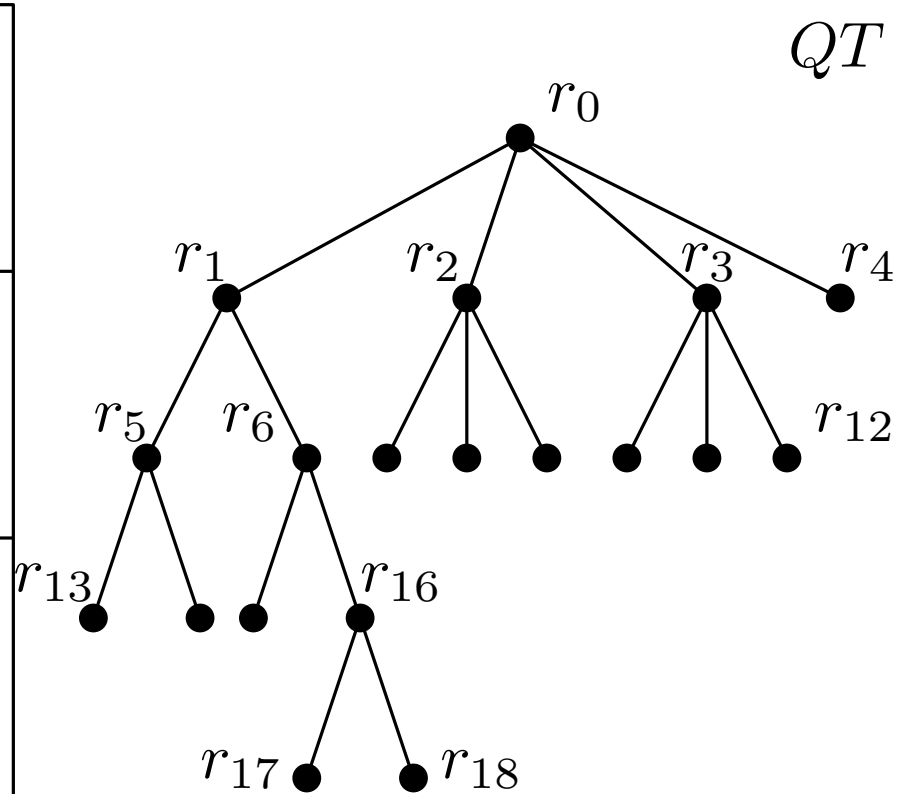


Quadtree Data Structure

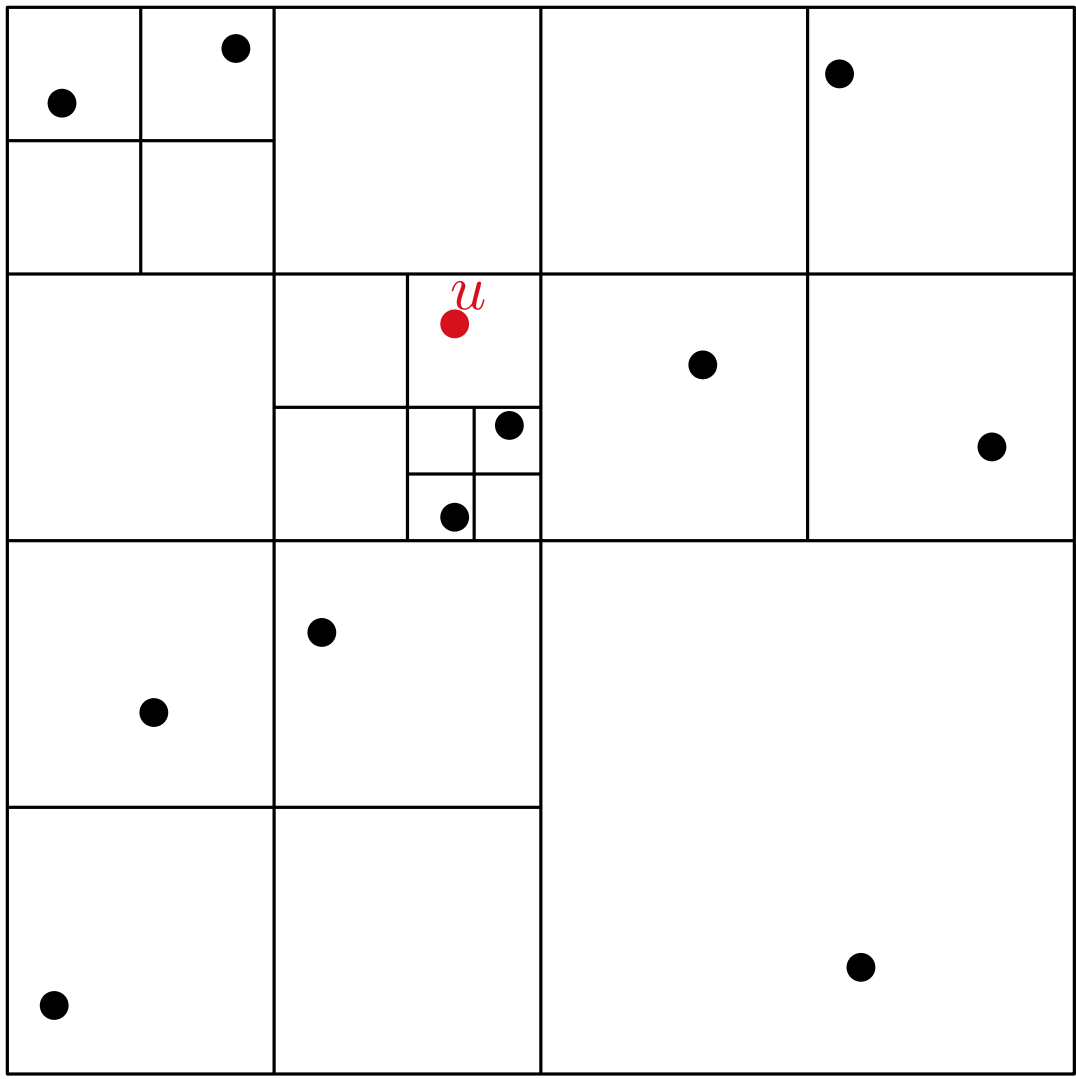
Quadtrees store point sets in $\mathbb{R}^2$ in a hierarchical, geometric data structure

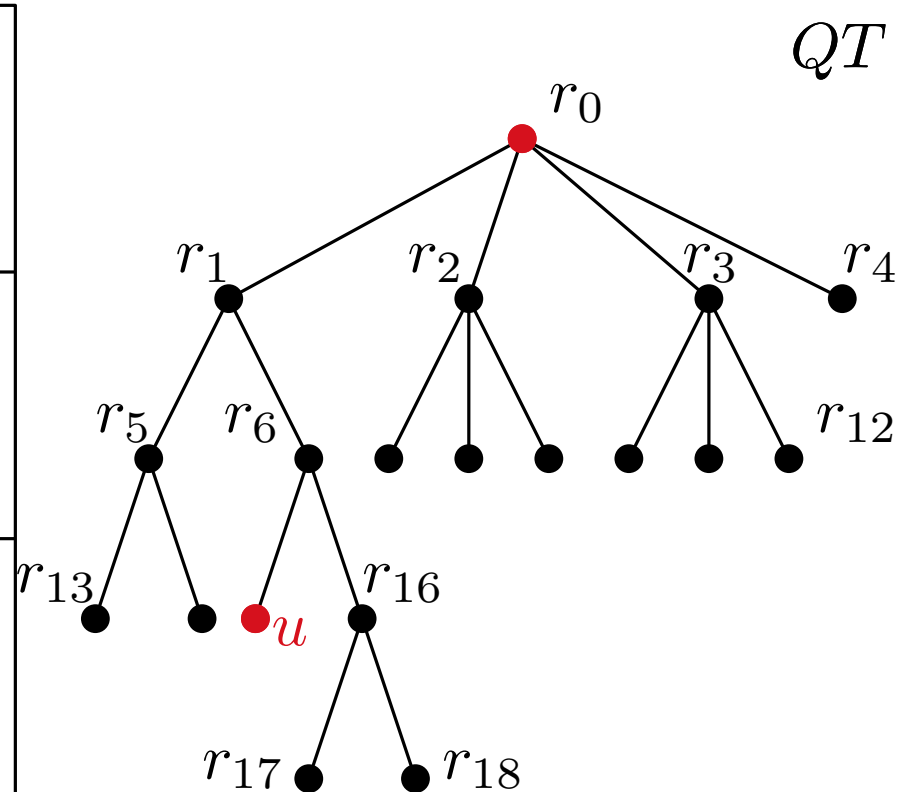
- if square has ≥ 2 points, then split into four subsquares and recurse



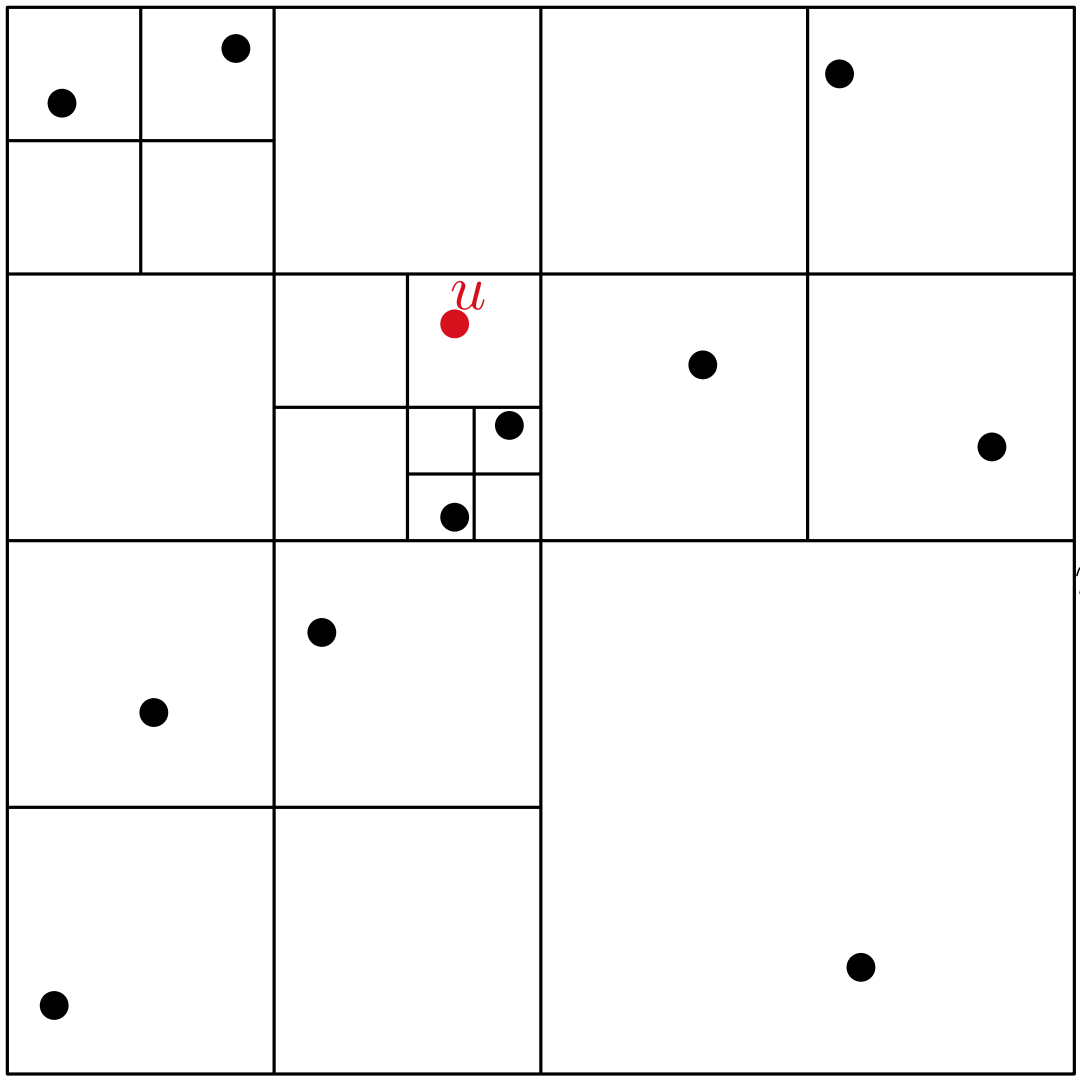


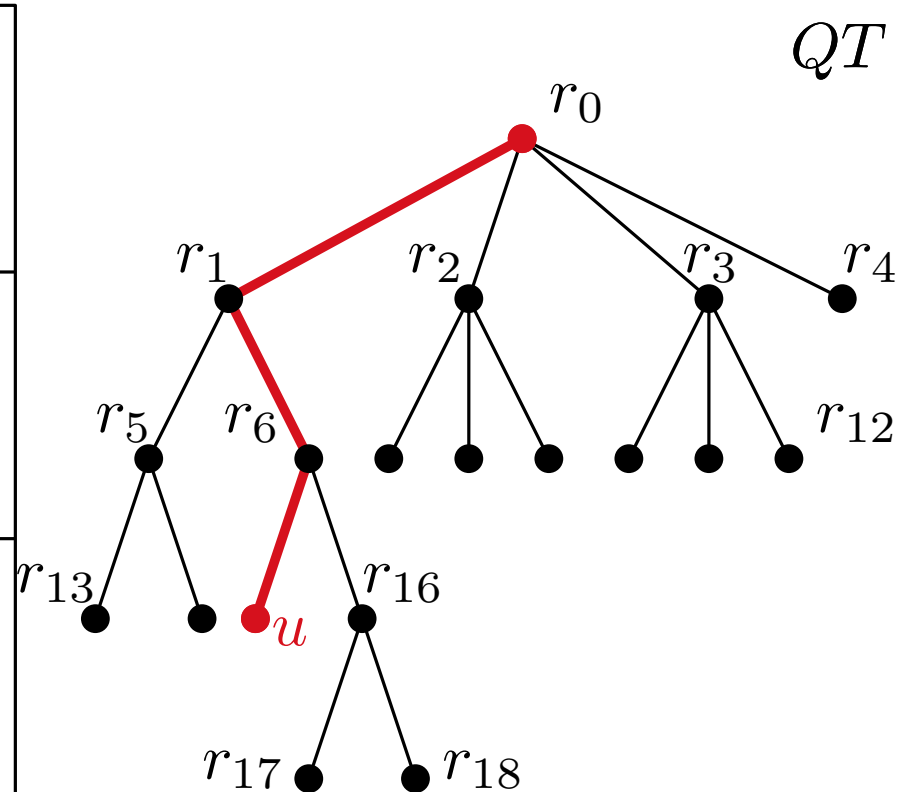
Force Approximation with Quadtrees (Barnes, Hut, 1986)



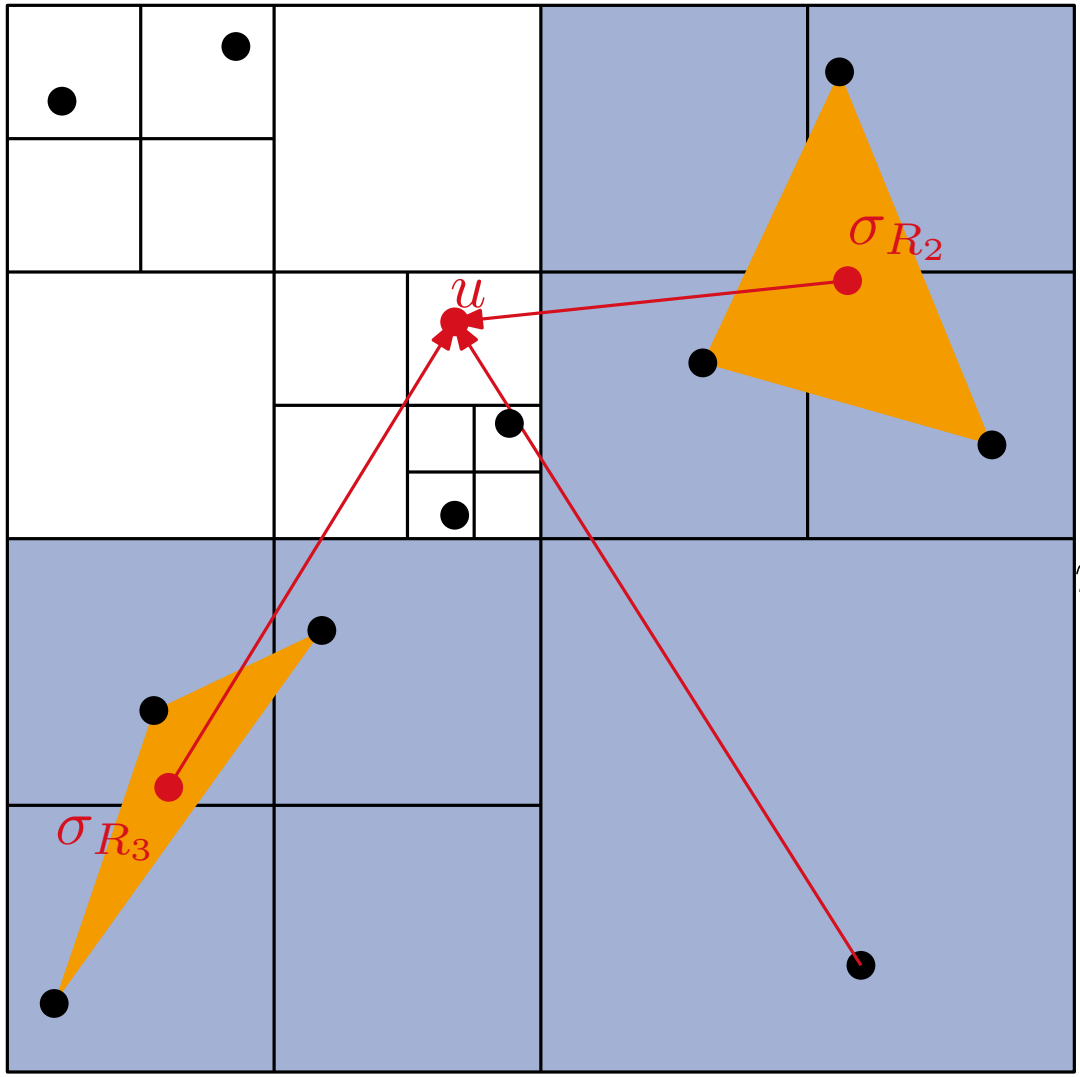


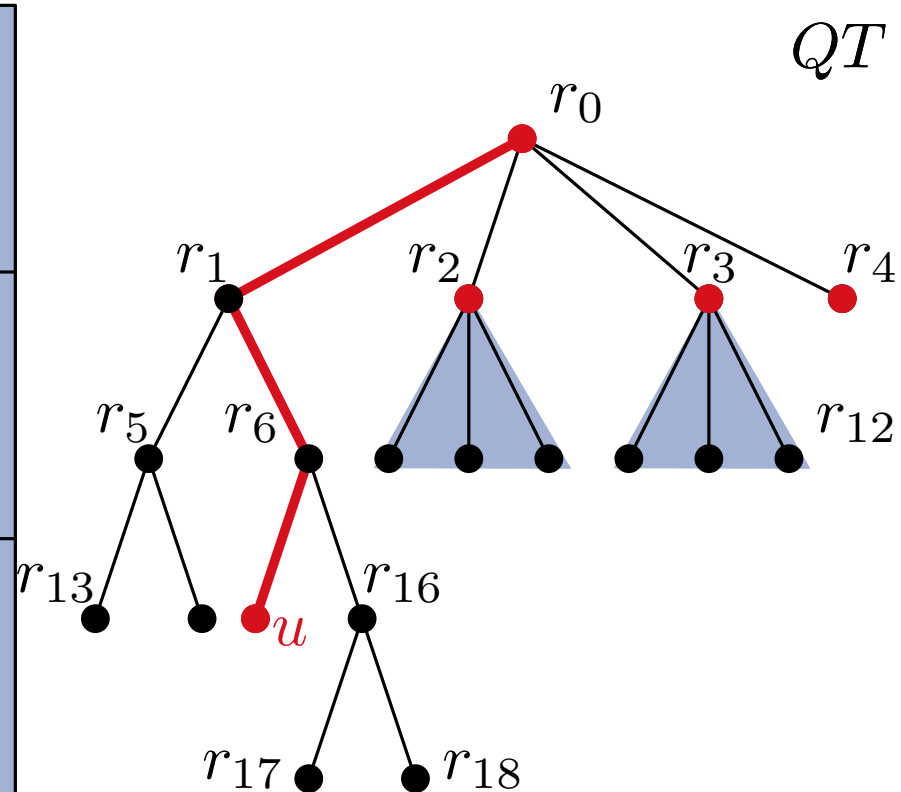
Force Approximation with Quadtrees (Barnes, Hut, 1986)



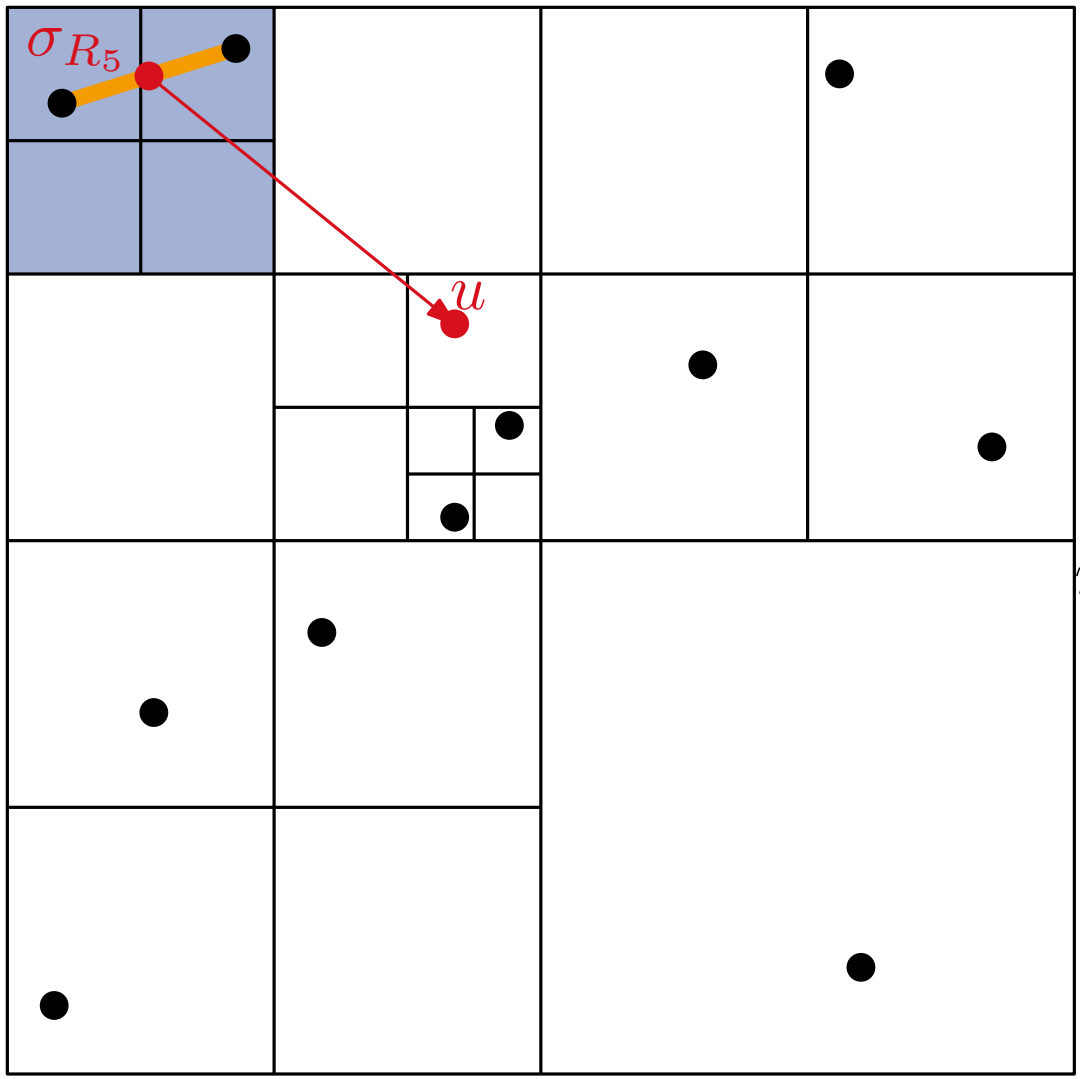


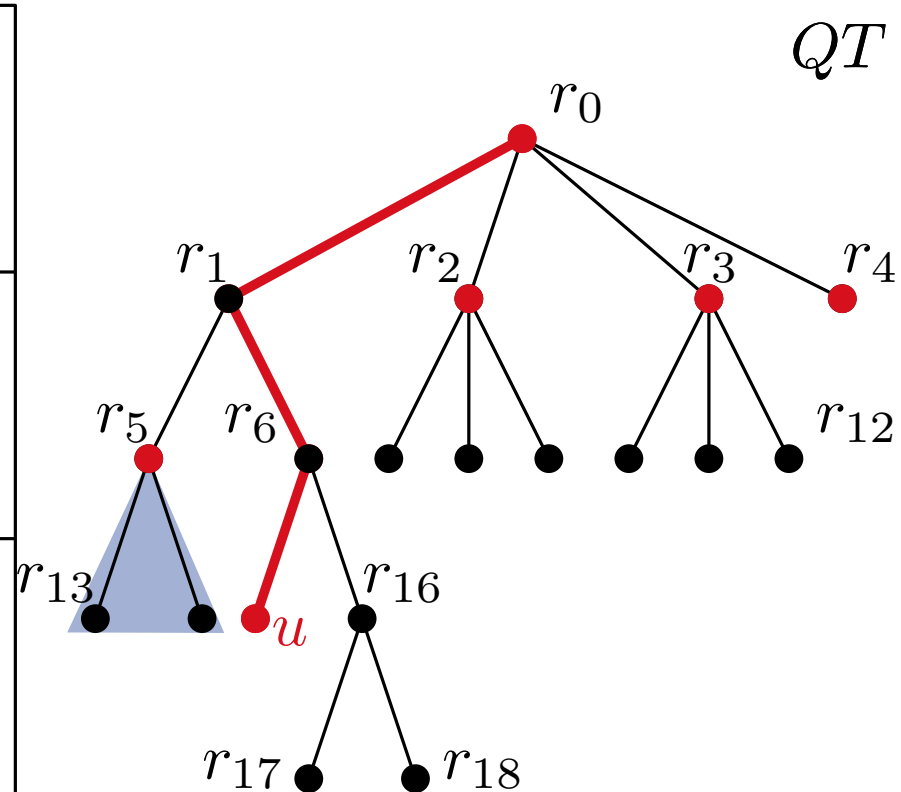
Force Approximation with Quadtrees (Barnes, Hut, 1986)



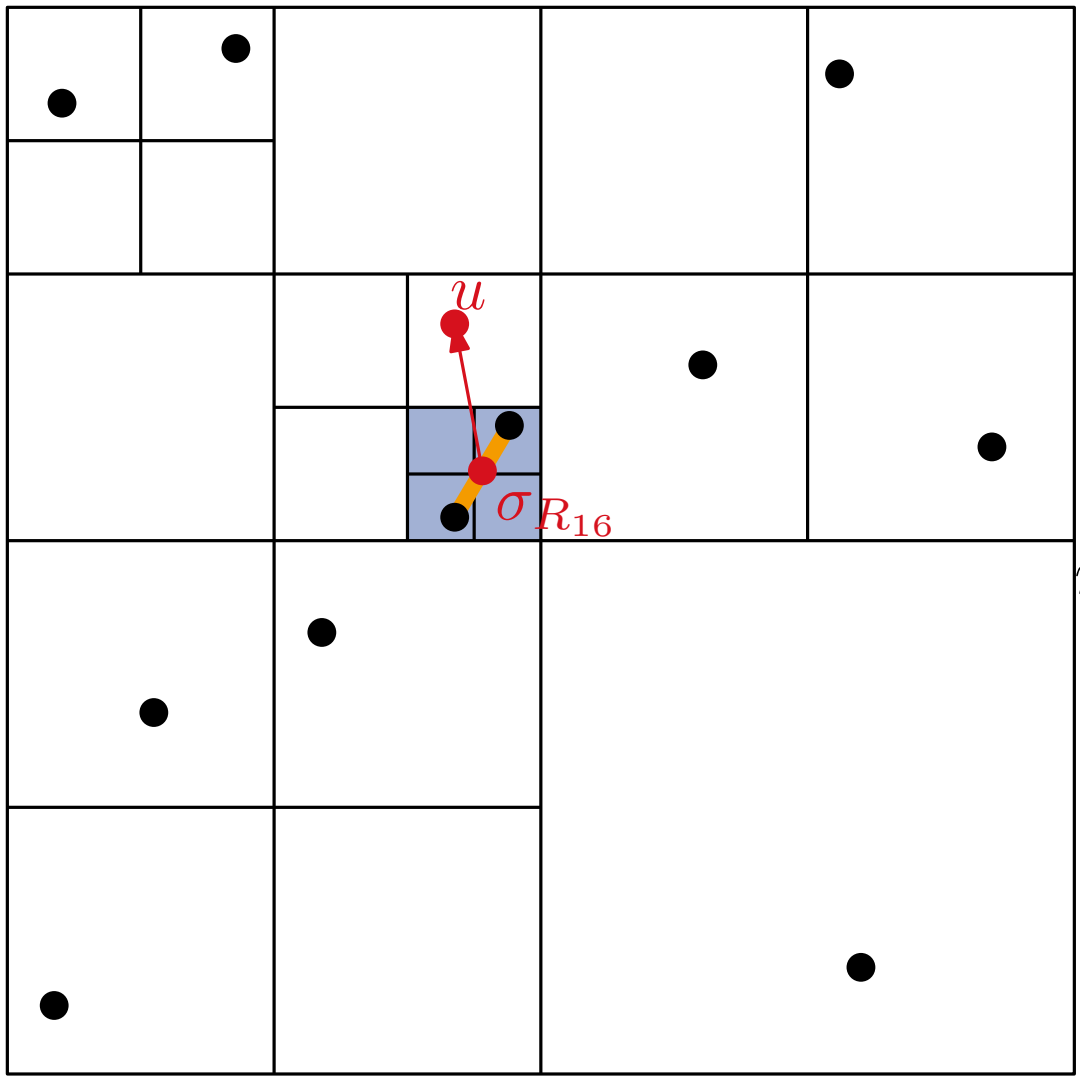


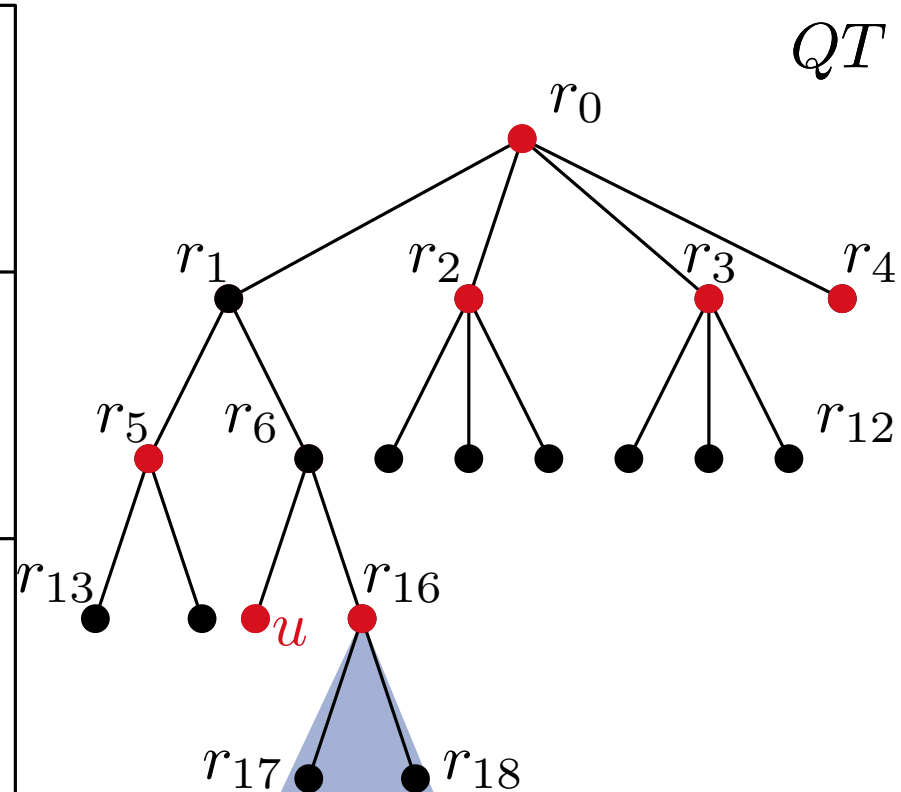
Force Approximation with Quadtrees (Barnes, Hut, 1986)



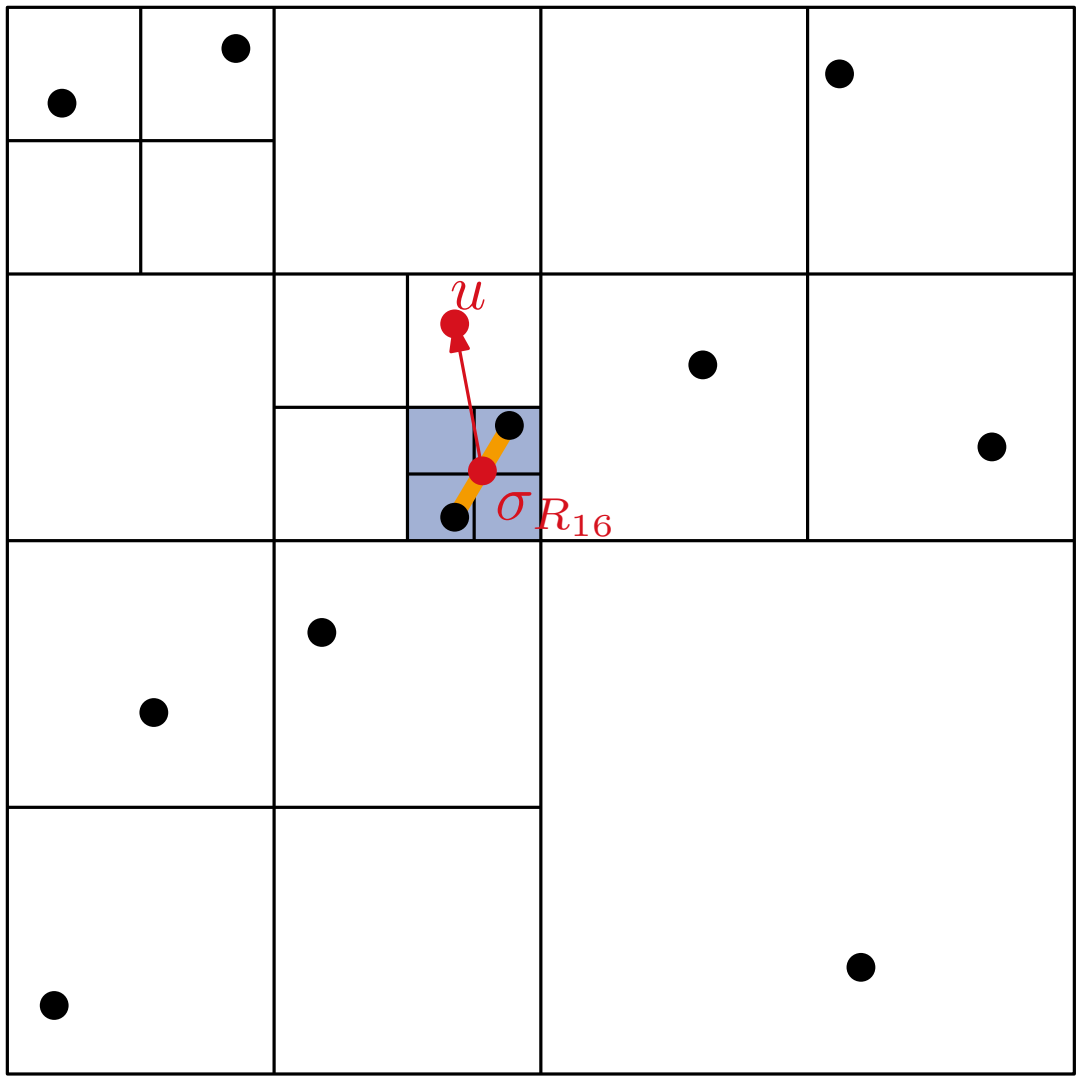


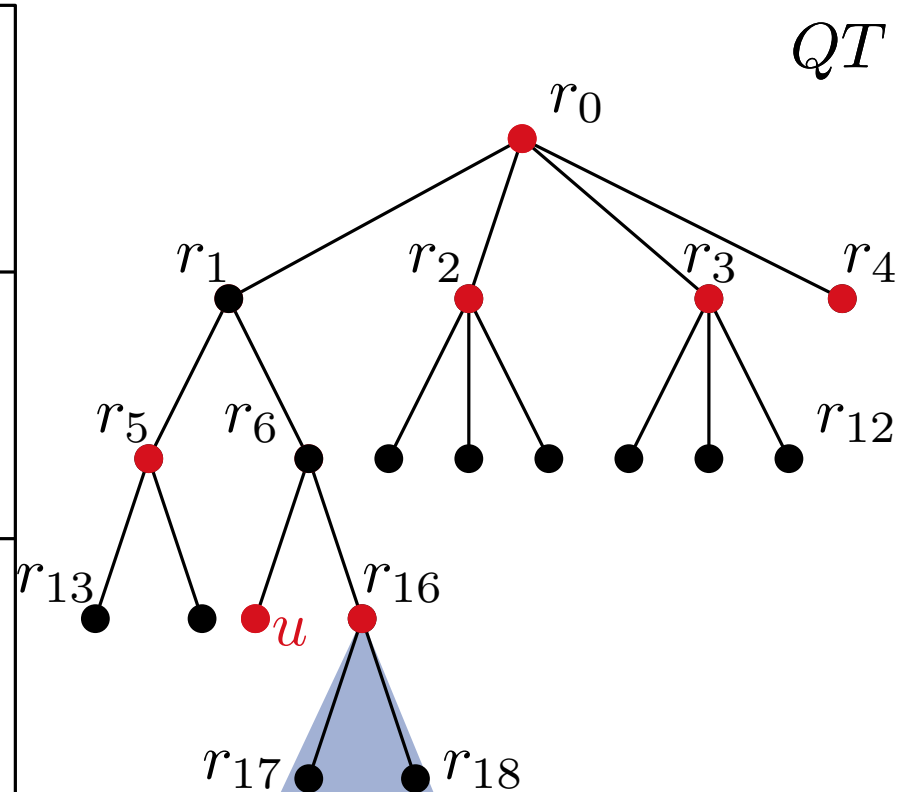
Force Approximation with Quadtrees (Barnes, Hut, 1986)





Force Approximation with Quadtrees (Barnes, Hut, 1986)





If the point distribution is more or less balanced, the quadtree has logarithmic height

$\Rightarrow$ compute repulsive forces in $O(n \log n)$ time.

Overview

Force-based graph layout

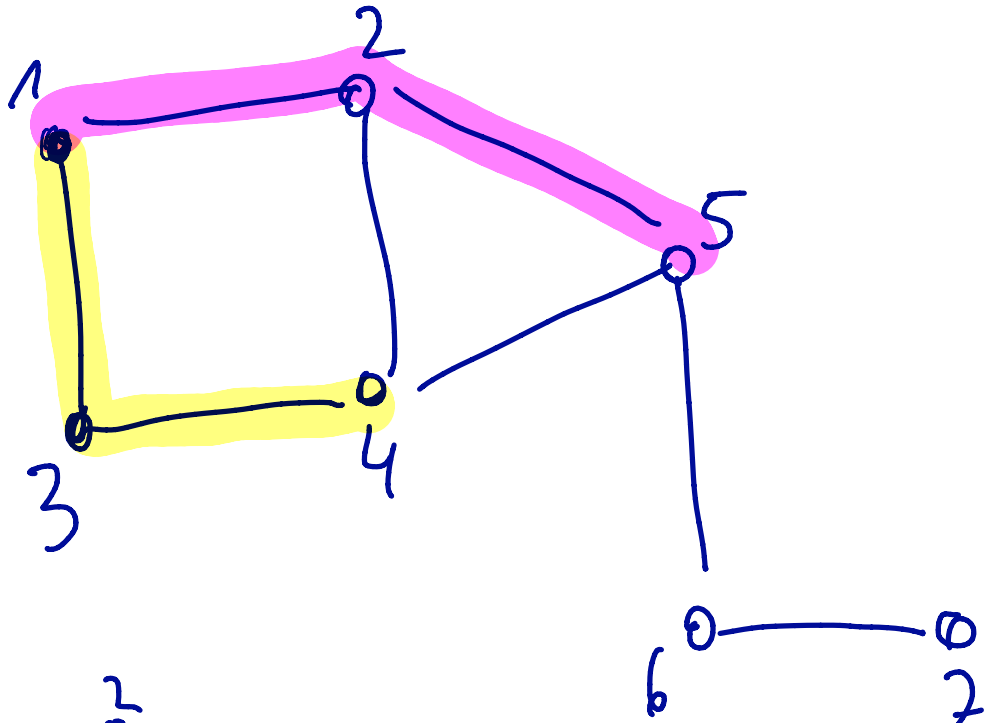
Scalability and faster force computation

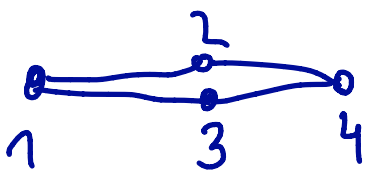
Stress-based graph layout

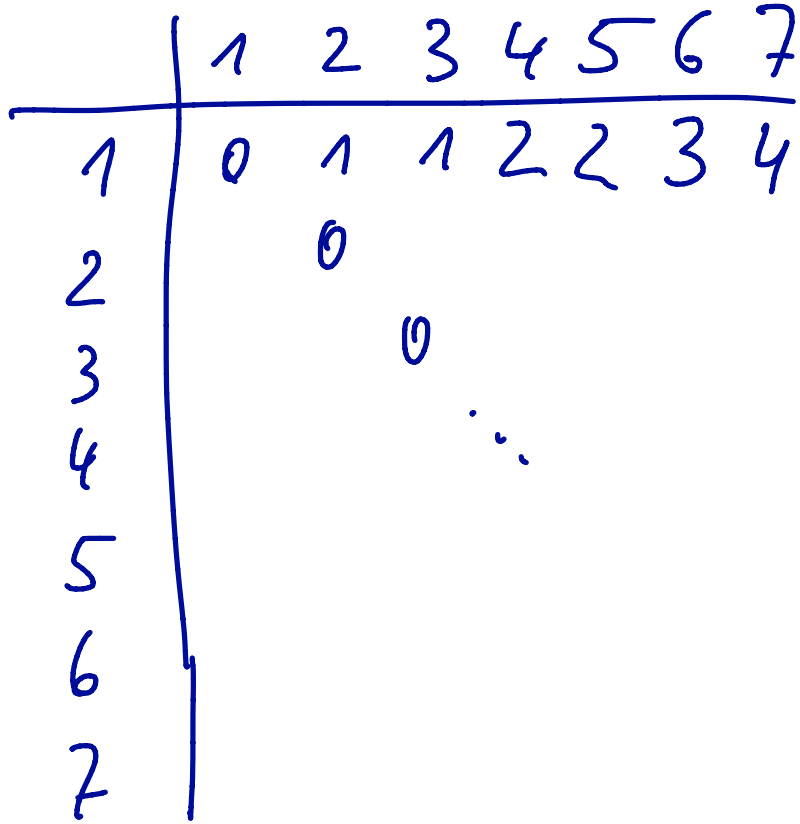
Stress Model

Idea: compute ideal distances between any two vertices and minimize deviation

What would be suitable pairwise distances?







Stress Model

Idea: compute ideal distances between any two vertices and minimize deviation

What would be suitable pairwise distances?

For a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ define distance matrix $D = (d_{ij}) \in \mathbb{R}^{n \times n}$, where d_{ij} is the shortest-path distance between v_i and v_j in G.

Stress Model

Idea: compute ideal distances between any two vertices and minimize deviation

What would be suitable pairwise distances?

For a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ define distance matrix $D = (d_{ij}) \in \mathbb{R}^{n \times n}$, where d_{ij} is the shortest-path distance between v_i and v_j in G.

Layout task: Find positions $p = (p_i)_{1 \leq i \leq n}$ in $\mathbb{R}^2$ that minimize the **stress**

$$s(p) = \sum_{i<j} w_{ij}(||p_i - p_j|| - d_{ij})^2$$

of the resulting layout, where w_{ij} is a weight term, e.g., $w_{ij} = 1/d_{ij}^2$ or $w_{ij} = d_{ij}^{-\alpha}$ for a parameter α.

Stress Model

Idea: compute ideal distances between any two vertices and minimize deviation

What would be suitable pairwise distances?

For a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ define distance matrix $D = (d_{ij}) \in \mathbb{R}^{n \times n}$, where d_{ij} is the shortest-path distance between v_i and v_j in G.

Layout task: Find positions $p = (p_i)_{1 \leq i \leq n}$ in $\mathbb{R}^2$ that minimize the **stress**

$$s(p) = \sum_{i<j} w_{ij}(||p_i - p_j|| - d_{ij})^2$$

of the resulting layout, where w_{ij} is a weight term, e.g., $w_{ij} = 1/d_{ij}^2$ or $w_{ij} = d_{ij}^{-\alpha}$ for a parameter α.

This model is also known as **multidimensional scaling (MDS)**.

Stress Minimization

Step 1: compute all pairwise distances in matrix $D = (d_{ij})$
$\rightarrow$ takes $O(mn + n^2 \log n)$ time and $O(n^2)$ space

Stress Minimization

Step 1: compute all pairwise distances in matrix $D = (d_{ij})$
$\rightarrow$ takes $O(mn + n^2 \log n)$ time and $O(n^2)$ space

Step 2: minimize $s(p)$, e.g., **stress majorization**

Stress Minimization

Step 1: compute all pairwise distances in matrix $D = (d_{ij})$
$\rightarrow$ takes $O(mn + n^2 \log n)$ time and $O(n^2)$ space

Step 2: minimize $s(p)$, e.g., **stress majorization**

- Gansner at el. (2004) showed that the following iterative solution process minimizes $s(p)$ using systems of linear equations:

$$L^w \cdot p(t+1) = L^{w,d}(t) \cdot p(t)$$

$$L^w_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{k \neq i} w_{ik} & \text{if } i = j \end{cases} \qquad L^{w,d}_{ij}(t) = \begin{cases} -\frac{w_{ij} d_{ij}}{||p(t)_i - p(t)_j||} & \text{if } i \neq j \\ -\sum_{k \neq i} L^{w,d}_{ik} & \text{if } i = j \end{cases}$$

until $\frac{s(p(t)) - s(p(t+1))}{s(p(t))} < \varepsilon$.

Stress Minimization

Step 1: compute all pairwise distances in matrix $D = (d_{ij})$
$\rightarrow$ takes $O(mn + n^2 \log n)$ time and $O(n^2)$ space

Step 2: minimize $s(p)$, e.g., **stress majorization**

- Gansner at el. (2004) showed that the following iterative solution process minimizes $s(p)$ using systems of linear equations:

$$L^w \cdot p(t+1) = L^{w,d}(t) \cdot p(t)$$

$$L^w_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{k \neq i} w_{ik} & \text{if } i = j \end{cases} \qquad L^{w,d}_{ij}(t) = \begin{cases} -\frac{w_{ij} d_{ij}}{||p(t)_i - p(t)_j||} & \text{if } i \neq j \\ -\sum_{k \neq i} L^{w,d}_{ik} & \text{if } i = j \end{cases}$$

until $\frac{s(p(t)) - s(p(t+1))}{s(p(t))} < \varepsilon$.

Practically efficient solvers are **Cholesky factorization** or the **conjugate gradient** (CG) method with $O(n^2)$ time per iteration.

Scalability of Stress-Based Methods

For large graphs the full stress model is too slow to solve with super-quadratic running times.

Several faster approximations have been developed, e.g.

- **PivotMDS** uses distances to a smaller set of $k \ll n$ pivot elements and saves a linear factor in Step 1 [Brandes, Pich 2006]

Scalability of Stress-Based Methods

For large graphs the full stress model is too slow to solve with super-quadratic running times.

Several faster approximations have been developed, e.g.

- **PivotMDS** uses distances to a smaller set of $k \ll n$ pivot elements and saves a linear factor in Step 1 [Brandes, Pich 2006]

- **MaxEnt stress** uses entropy maximization for non-edges as $m(p) = \sum_{ij \in E} w_{ij}(||p_i - p_j|| - d_{ij})^2 - \alpha \sum_{ij \notin E} \log ||p_i - p_j||$, which allows solving one iteration in $O(m + n \log n)$ time. [Gansner et al. 2013]

Scalability of Stress-Based Methods

For large graphs the full stress model is too slow to solve with super-quadratic running times.

Several faster approximations have been developed, e.g.

- **PivotMDS** uses distances to a smaller set of $k \ll n$ pivot elements and saves a linear factor in Step 1 [Brandes, Pich 2006]

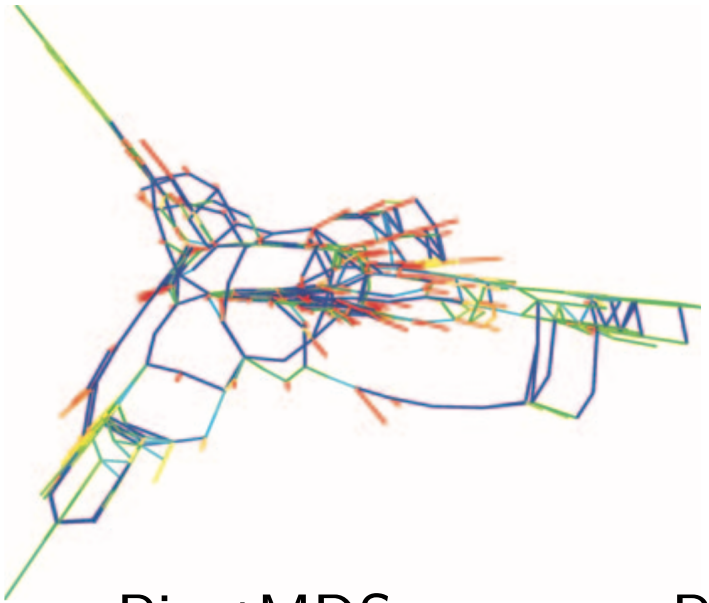
- **MaxEnt stress** uses entropy maximization for non-edges as $m(p) = \sum_{ij \in E} w_{ij}(||p_i - p_j|| - d_{ij})^2 - \alpha \sum_{ij \notin E} \log ||p_i - p_j||$, which allows solving one iteration in $O(m + n \log n)$ time. [Gansner et al. 2013]

- **Sparse Stress** computes stress to a suitable smaller set $\mathcal{P}$ of $k \ll n$ pivot elements

$$s'(p) = \sum_{ij \in E} w_{ij}(||p_i - p_j|| - d_{ij})^2 + \sum_{i \in V} \sum_{j \in \mathcal{P} \setminus N(i)} w'_{ij}(||p_i - p_j|| - d_{ij})^2,$$

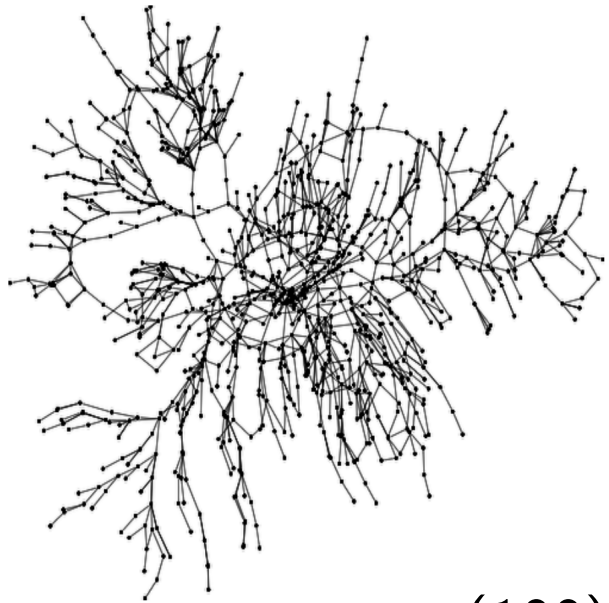
  which allows solving one iteration in $O(kn + m)$ time and precomputing the distances to $\mathcal{P}$ in $O(km + kn \log n)$ time. [Ortmann et al. 2017]

Examples [Gansner et al. 2013] [Ortmann et al. 2017]

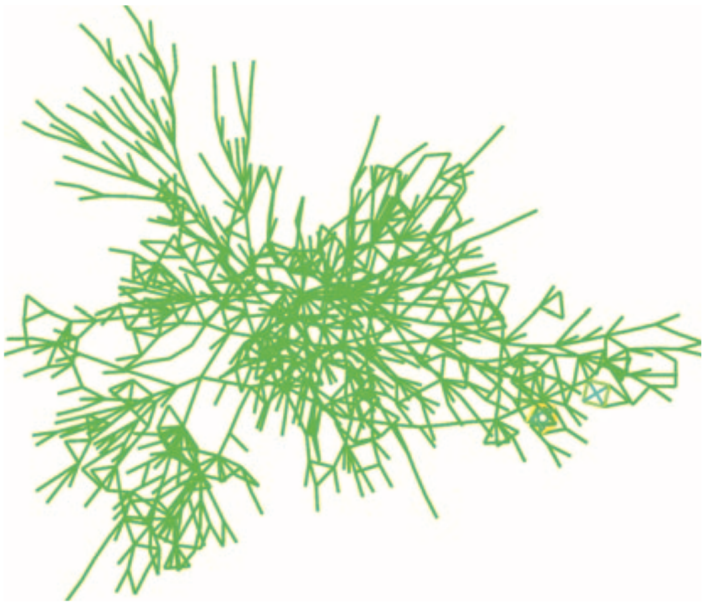


PivotMDS

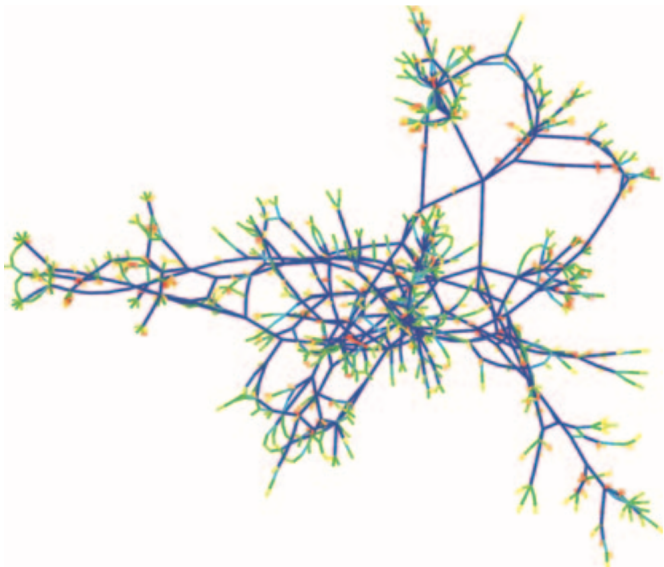
PivotMDS + local sparse stress



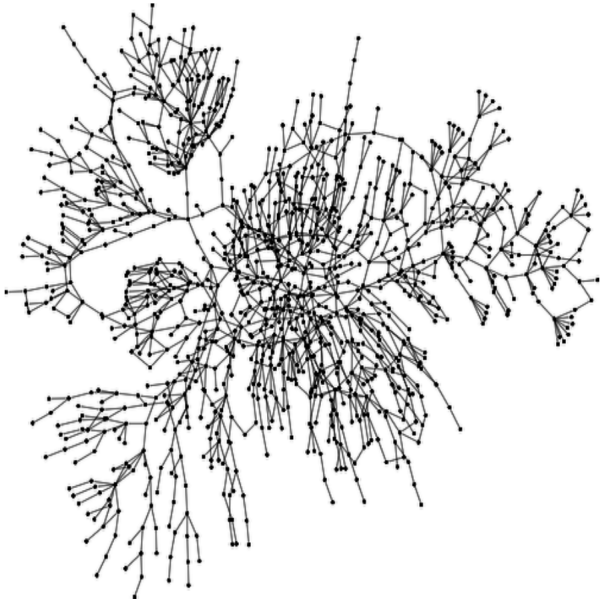
sparse stress (100)



MaxEnt



spring-electrical model



full stress

Some Running Times [Ortmann et al. 2017]

(in seconds)

graph	full stress	sparse 200	sparse 100	sparse 50	maxent*	MARS 200*	MARS 100*	GRIP*	1-stress	Pivot MDS
dwt1005	1.26	0.33	0.15	**0.09**	0.47	1.02	2.36	0.06	0.08	0.06
1138bus	2.20	0.41	0.16	**0.09**	0.91	3.16	1.96	0.20	0.06	0.04
plat1919	9.70	1.00	0.45	**0.24**	1.15	6.80	4.79	0.19	0.31	0.20
3elt	31.82	2.28	0.93	**0.43**	2.26	16.31	8.43	0.71	0.37	0.23
USpowerGrid	36.48	1.85	0.67	**0.37**	2.53	13.54	7.62	1.67	0.28	0.21
commanche	340.10	10.78	3.63	**1.51**	3.60	22.72	12.43	2.29	0.47	0.35
LeHavre	475.05	12.75	4.90	**2.19**	6.31	27.57	19.50	10.18	0.81	0.54
pesa	373.23	9.61	4.14	**1.50**	5.96	50.10	42.68	3.56	0.95	0.60
bodyy5	463.47	12.53	4.31	**2.01**	9.97	46.63	9.27	10.43	1.64	1.04
finance256	1016.92	10.44	4.27	**2.28**	14.76	32.16	24.66	12.12	2.51	1.60
btree	7.79	0.42	0.18	**0.09**	0.63	2.70	1.48	0.06	0.06	0.03
qh882	6.61	0.65	0.28	**0.15**	0.97	8.45	5.79	0.15	0.17	0.14
lpship04l	18.30	0.73	**0.31**	0.18	0.99	7.06	7.63	0.16	0.15	0.10

graph	n	m
dwt1005	1005	3808
1138bus	1138	1458
plat1919	1919	15240
3elt	4740	13722
USpowerGrid	4941	6594
commanche	7920	11880**
LeHavre	11730	15133**
pesa	11738	33914
bodyy5	18589	55346
finance256	20657	71866
btree (binary tree)	1023*	1022
qh882	1764*	3354
lpship04l	2526*	6380

Summary

Force-based approaches

- easily understandable and implementable
- no special requirements on the input graph
- for small and sparse graphs amazingly good layouts (symmetries, clusters, ...)
- easily adaptable and configurable
- robust and scalable (with extra work)

Summary

Force-based approaches

- easily understandable and implementable
- no special requirements on the input graph
- for small and sparse graphs amazingly good layouts (symmetries, clusters, ...)
- easily adaptable and configurable
- robust and scalable (with extra work)

Stress-based approaches

- distance optimization usually gives high quality
- no special requirements on the input graph
- iterative solving of equation systems or gradient descent

Summary

Force-based approaches

- easily understandable and implementable
- no special requirements on the input graph
- for small and sparse graphs amazingly good layouts (symmetries, clusters, ...)
- easily adaptable and configurable
- robust and scalable (with extra work)

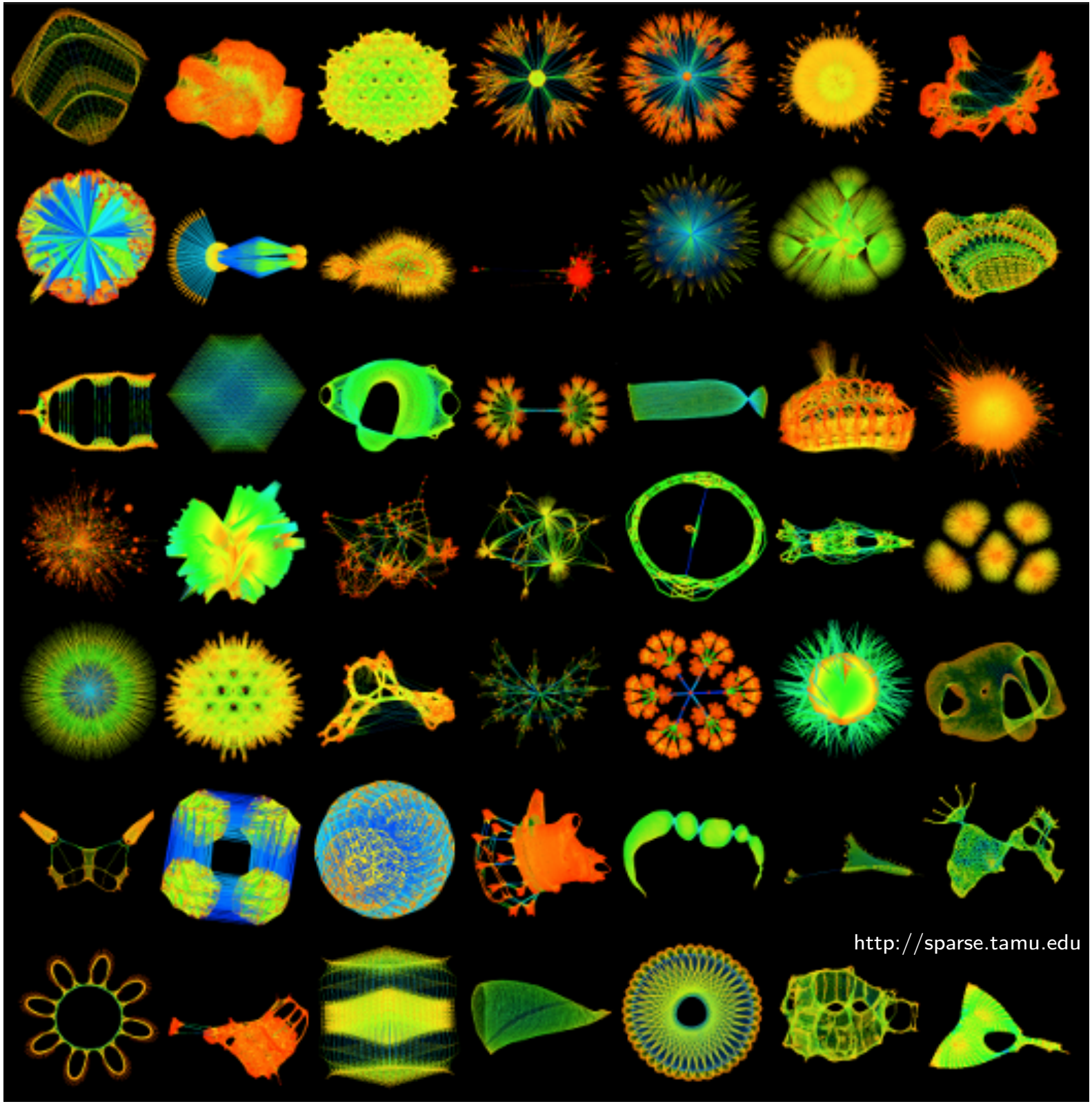
Stress-based approaches

- distance optimization usually gives high quality
- no special requirements on the input graph
- iterative solving of equation systems or gradient descent

But...

- no guarantees on convergence time
- dependence on initial layout $\rightarrow$ slow convergence
- full stress too slow for large graphs, but fast approximations

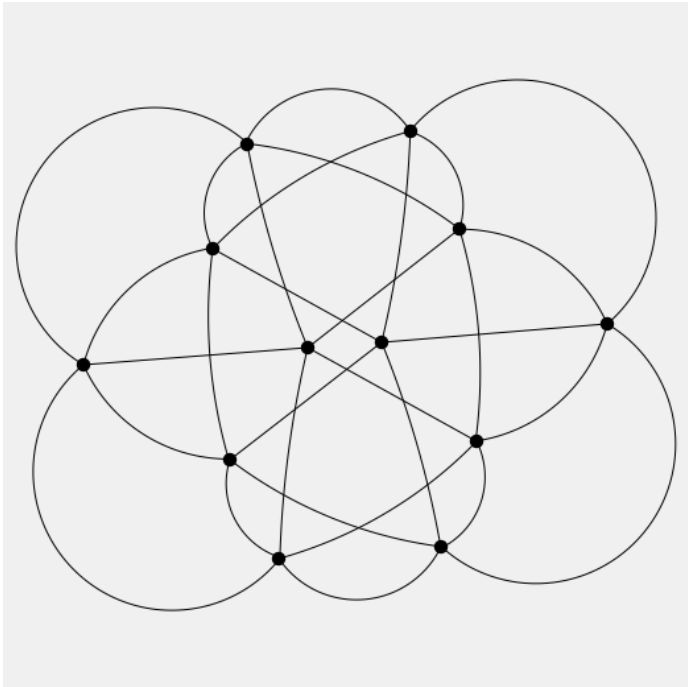
Gallery

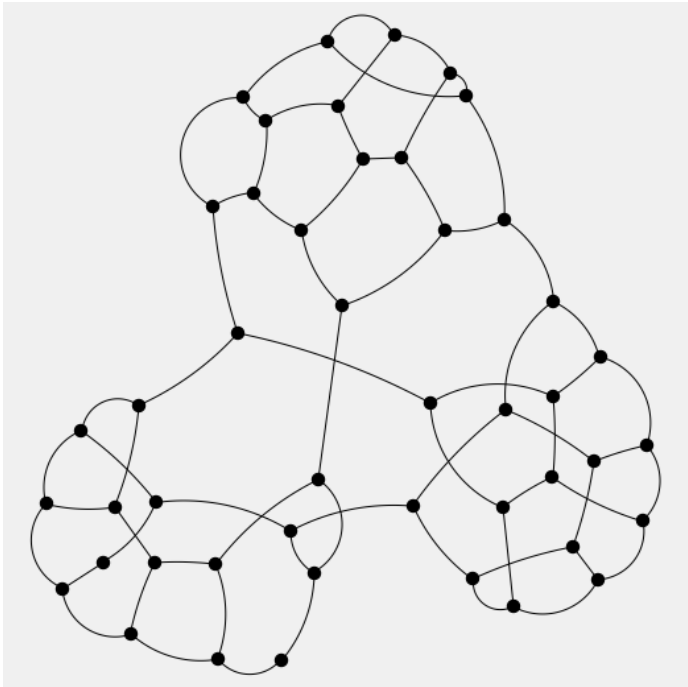


Other Examples

Lombardi-Spring-Embedder (Chernobelskiy et al. 2012)

- edges are circular arcs
- goal: optimal angular resolution $2\pi/\deg(v)$ at each node v
- additional rotational forces





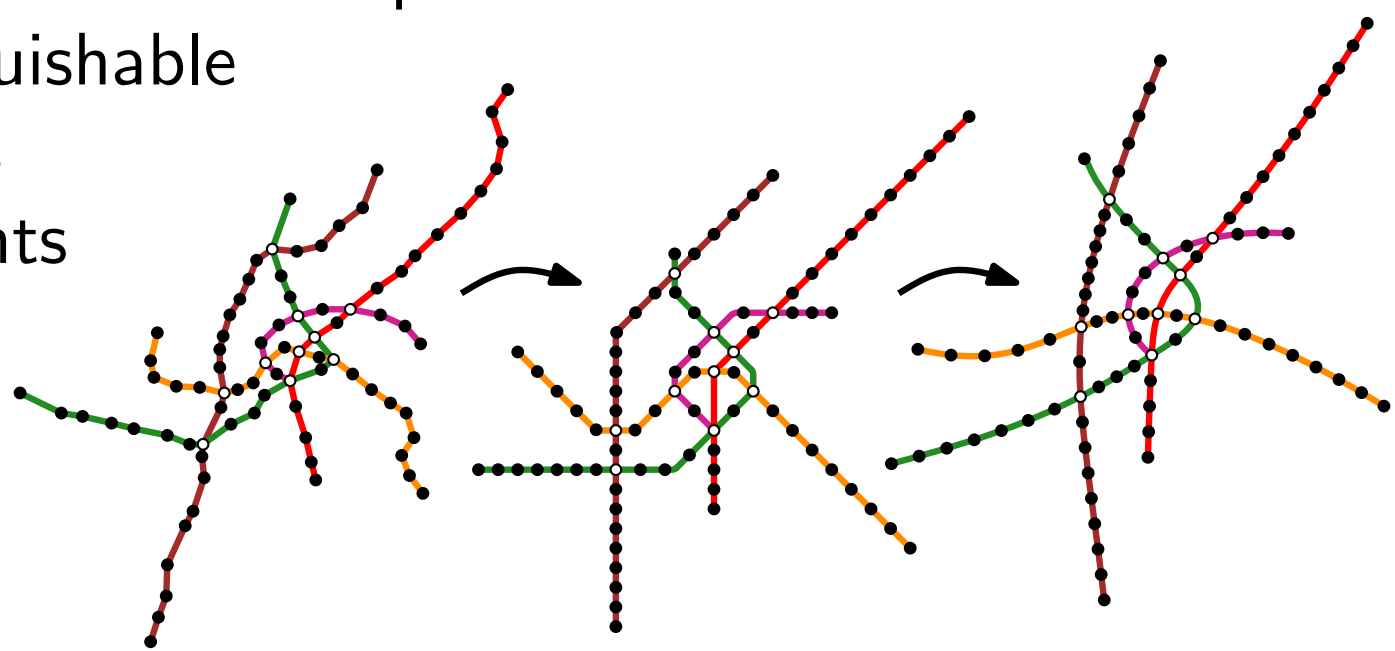
Other Examples

Lombardi-Spring-Embedder (Chernobelskiy et al. 2012)

- edges are circular arcs
- goal: optimal angular resolution $2\pi/\deg(v)$ at each node v
- additional rotational forces

Metro Maps with Bézier curves (Fink et al. 2013)

- model paths as Bézier curves
- forces on nodes and control points:
- lines are distinguishable
- few bend points
- few control points

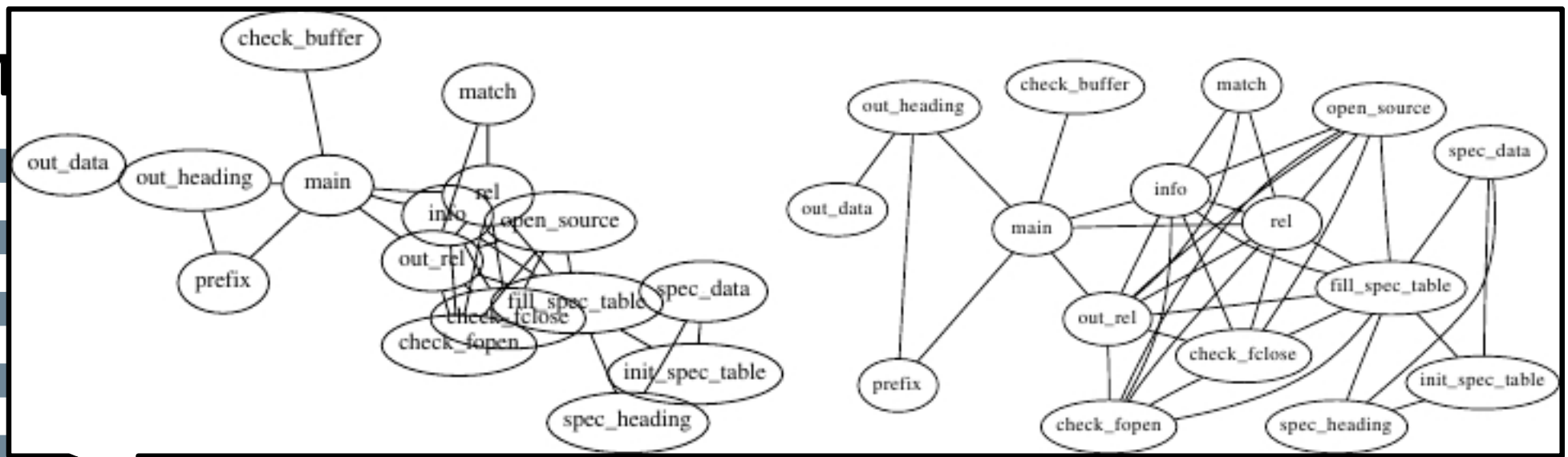


Other Examples

Lombardi-Spring-Embedder (Chernobelskiy et al. 2012)

- edges are circular arcs
- goal: optimal angular resolution $2\pi/\deg(v)$ at each node v
- additional rotational forces

M



- control points

Realistic Node Sizes (Gansner, North 1998)

- node positions are adjusted to avoid overlaps