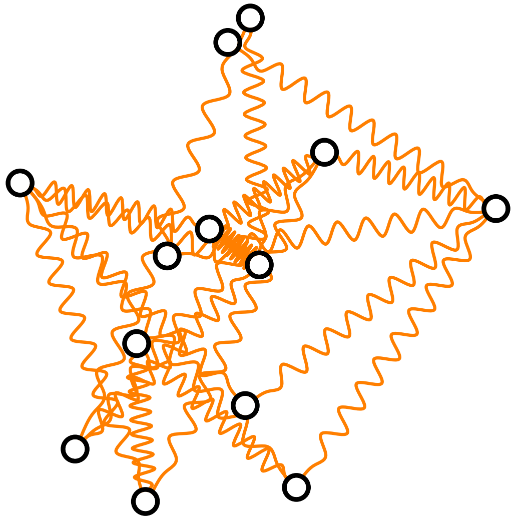
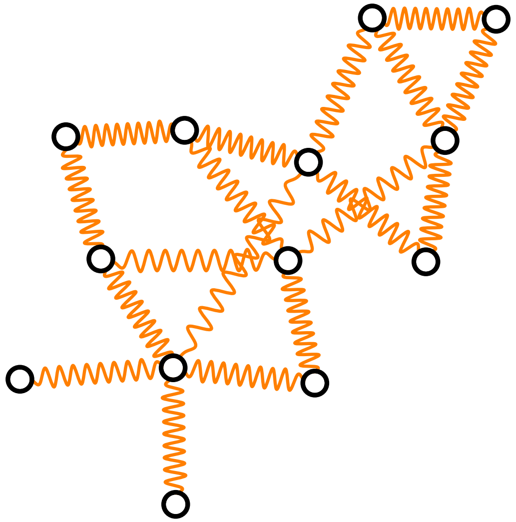
Visualization of graphs

Force-directed algorithms

Drawing with physical analogies

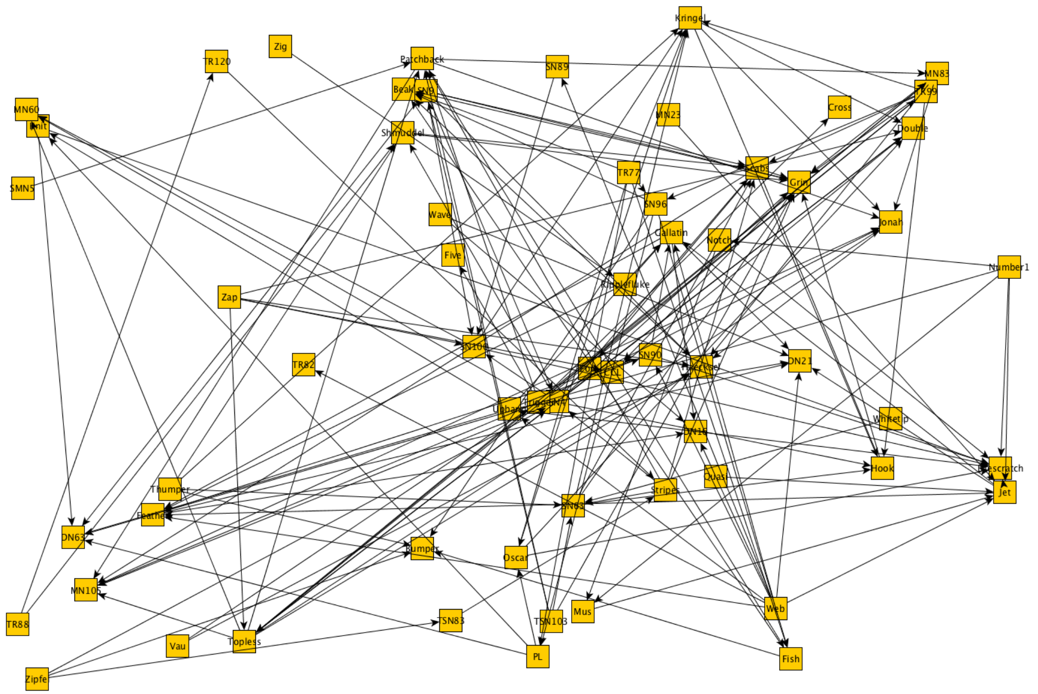
Jonathan Klawitter · Summer semester 2020





General Layout Problem

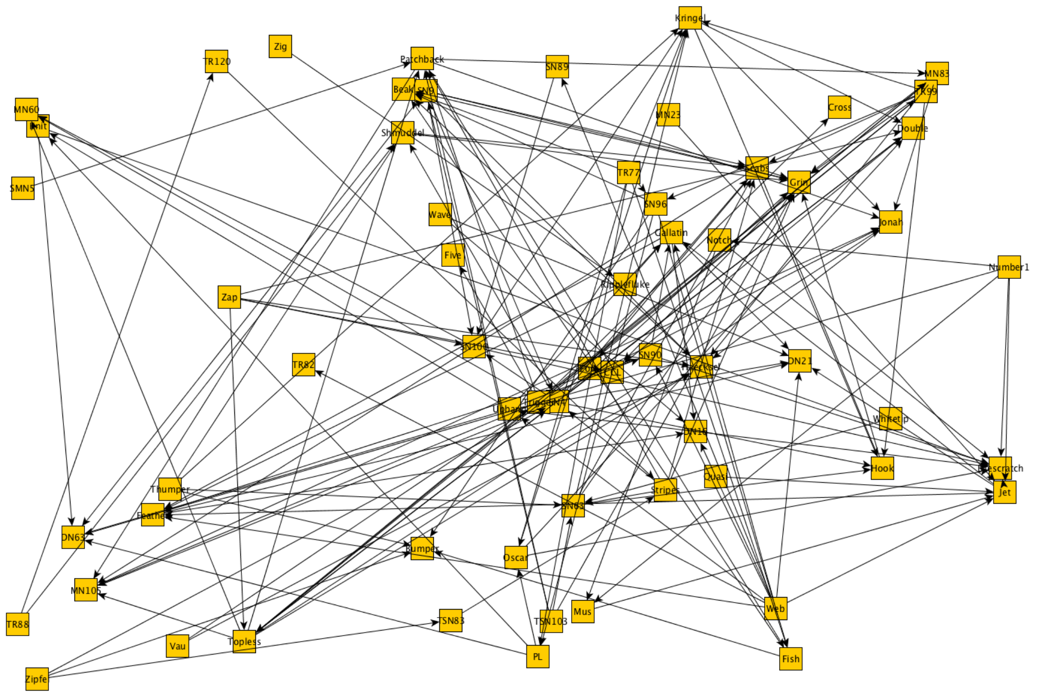
Input: Graph $G = (V, E)$

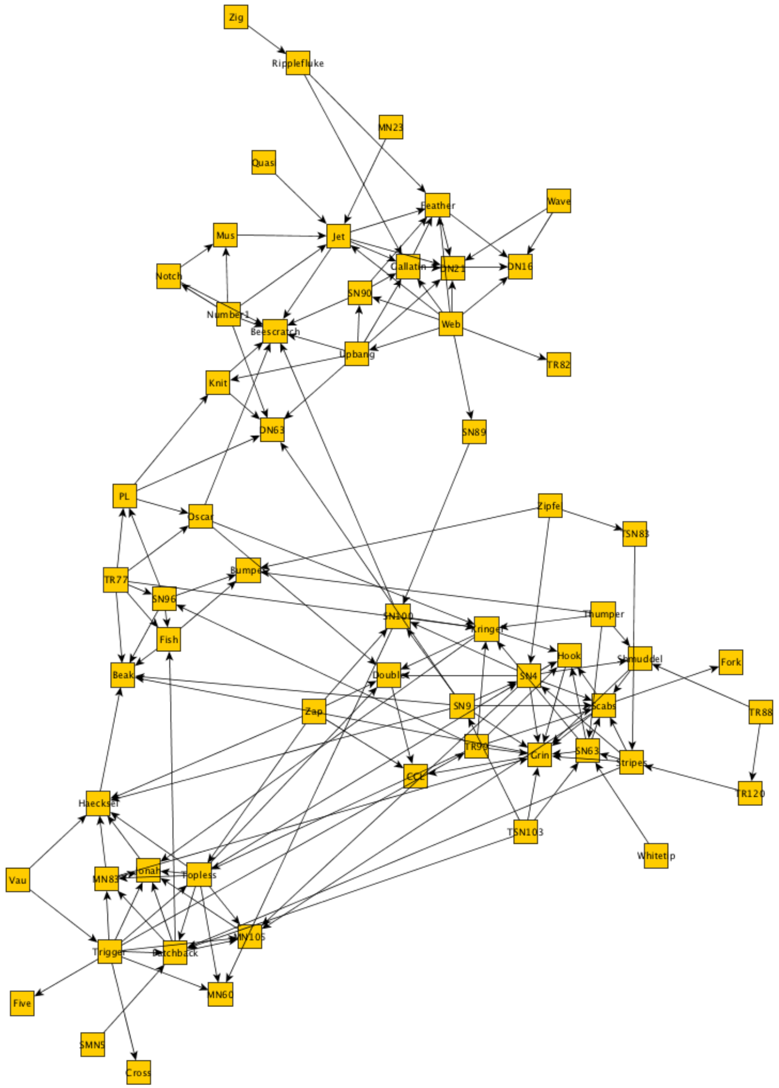


General Layout Problem

Input: Graph $G = (V, E)$

Output: Clear and readable straight-line drawing of G

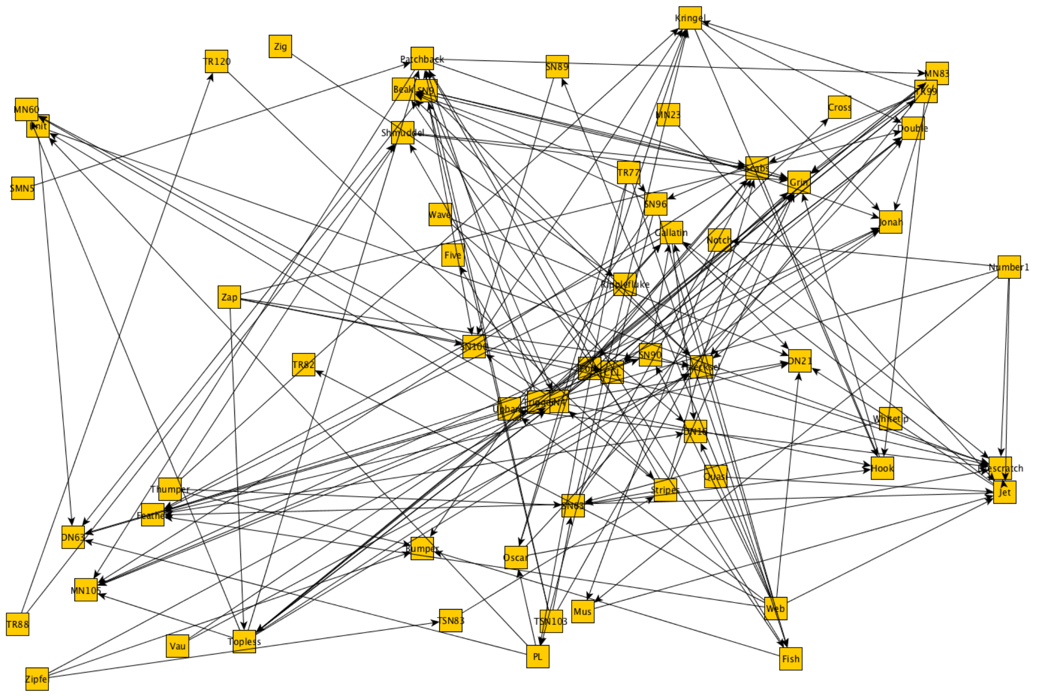


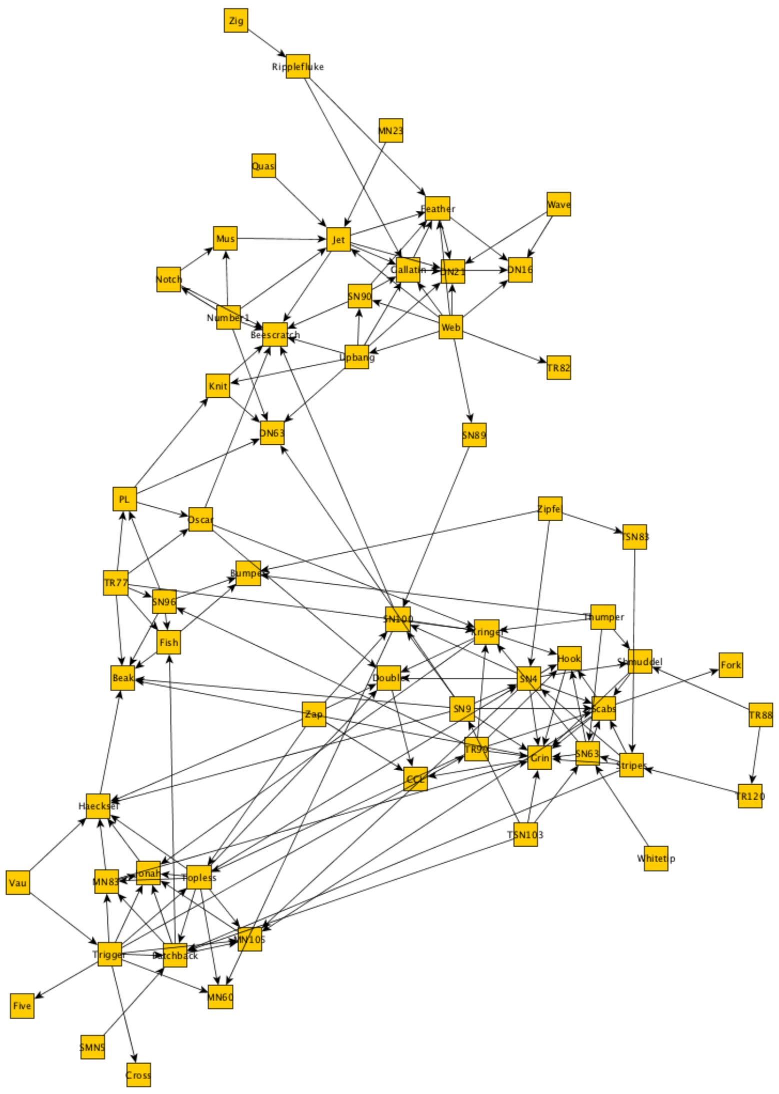


General Layout Problem

Input: Graph $G = (V, E)$

Output: Clear and readable straight-line drawing of G





- Which aesthetic criteria would you optimize?

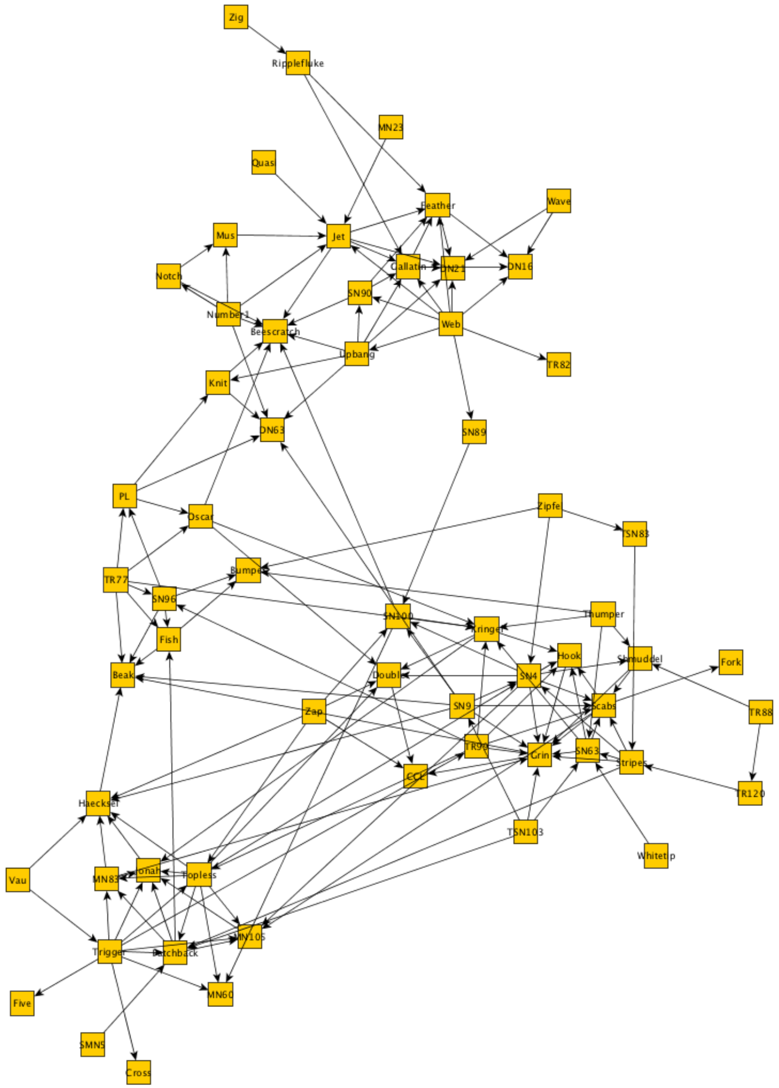
General Layout Problem

Input: Graph $G = (V, E)$

Output: Clear and readable straight-line drawing of G

Aesthetic criteria:

- adjacent vertices are close
- non-adjacent vertices are far apart
- edges short, straight-line, **similar length**
- densely connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly



General Layout Problem

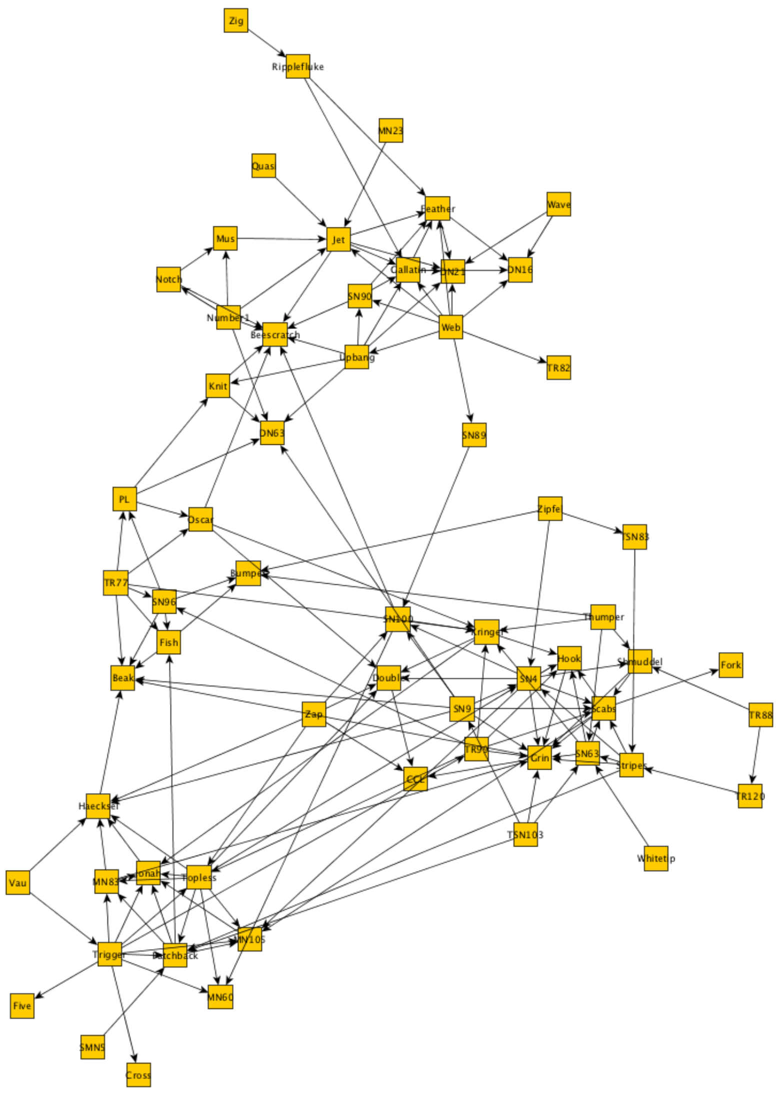
Input: Graph $G = (V, E)$

Output: Clear and readable straight-line drawing of G

Aesthetic criteria:

- adjacent vertices are close
- non-adjacent vertices are far apart
- edges short, straight-line, **similar length**
- densely connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

Optimization criteria partially contradict each other



Fixed edge lengths?

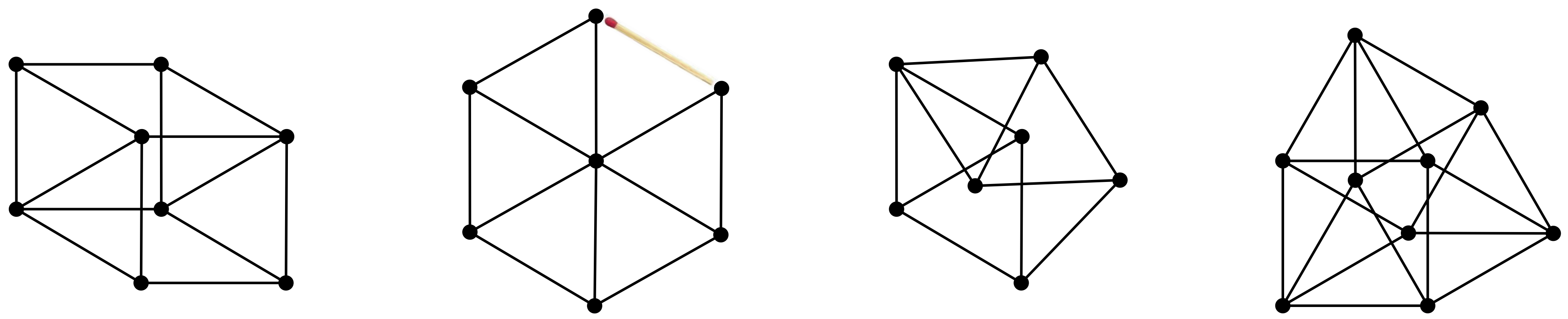
Input: Graph $G = (V, E)$, required edge length $\ell(e)$, $\forall e \in E$

Output: Drawing of G which realizes all the edge lengths

Fixed edge lengths?

Input: Graph $G = (V, E)$, required edge length $\ell(e)$, $\forall e \in E$

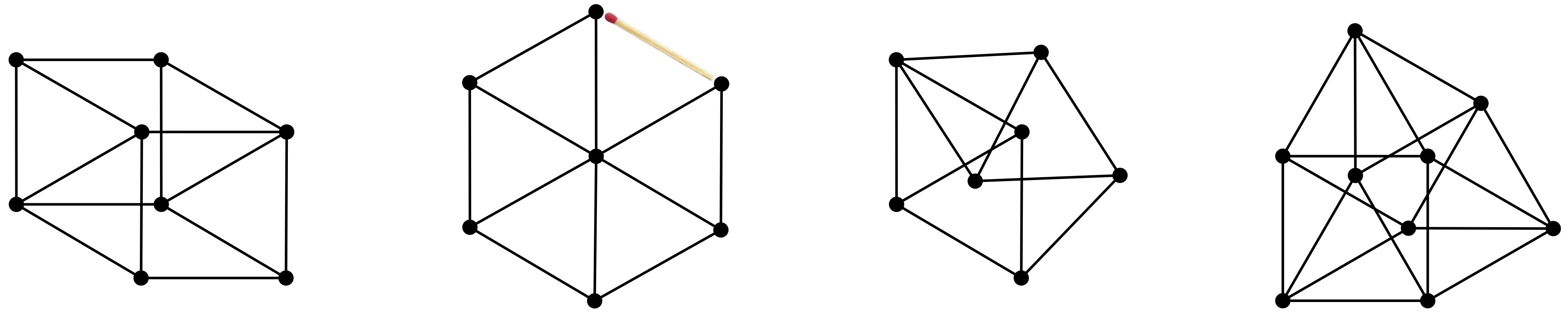
Output: Drawing of G which realizes all the edge lengths



Fixed edge lengths?

Input: Graph $G = (V, E)$, required edge length $\ell(e)$, $\forall e \in E$

Output: Drawing of G which realizes all the edge lengths



NP-hard for

- uniform edge lengths in any dimension [Johnson '82]
- uniform edge lengths in planar drawings [Eades, Wormald '90]
- edge lengths $\{1, 2\}$ [Saxe '80]

Physical analogy

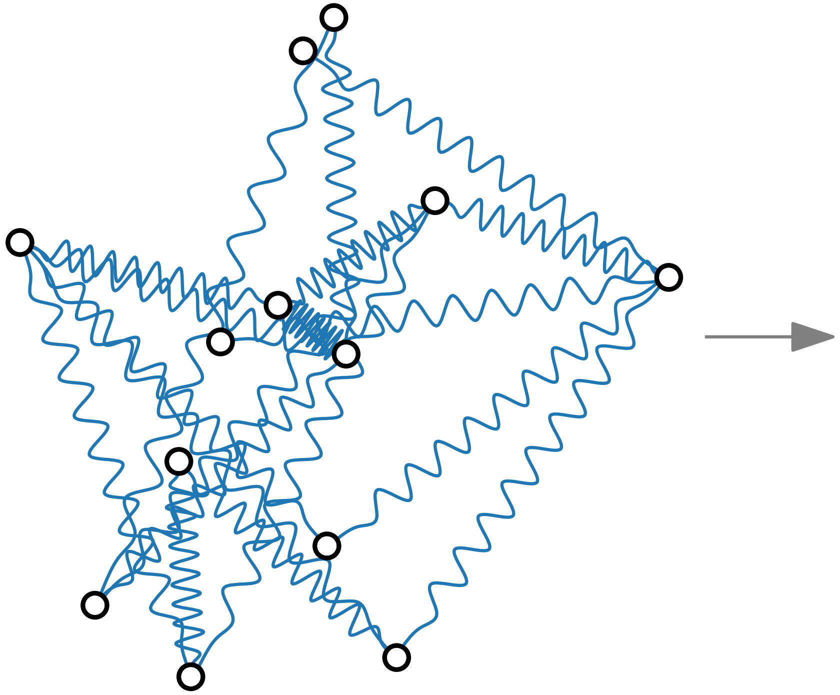
Idea 1.

“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

Physical analogy

Idea 1.

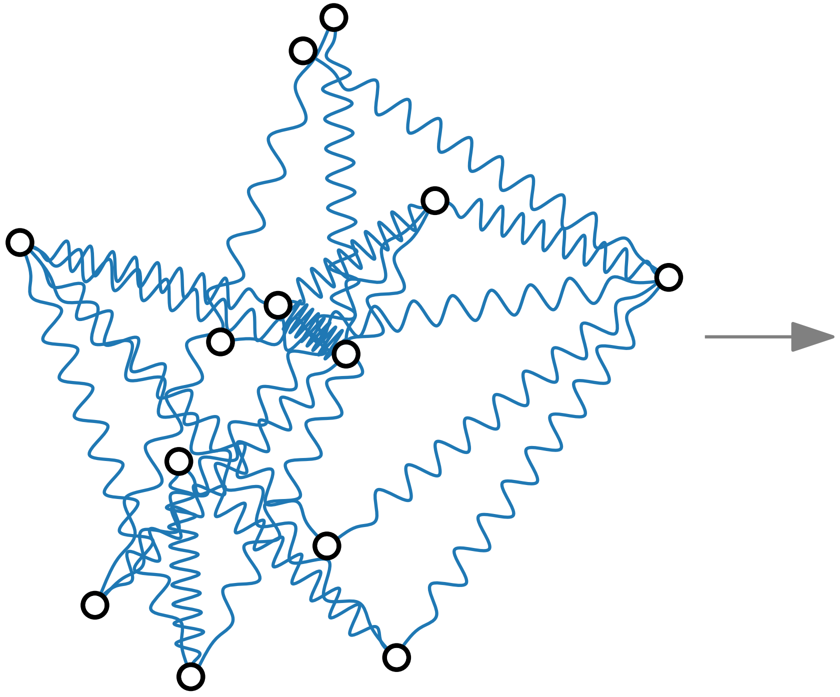
“To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . . The vertices are placed in some initial layout



Physical analogy

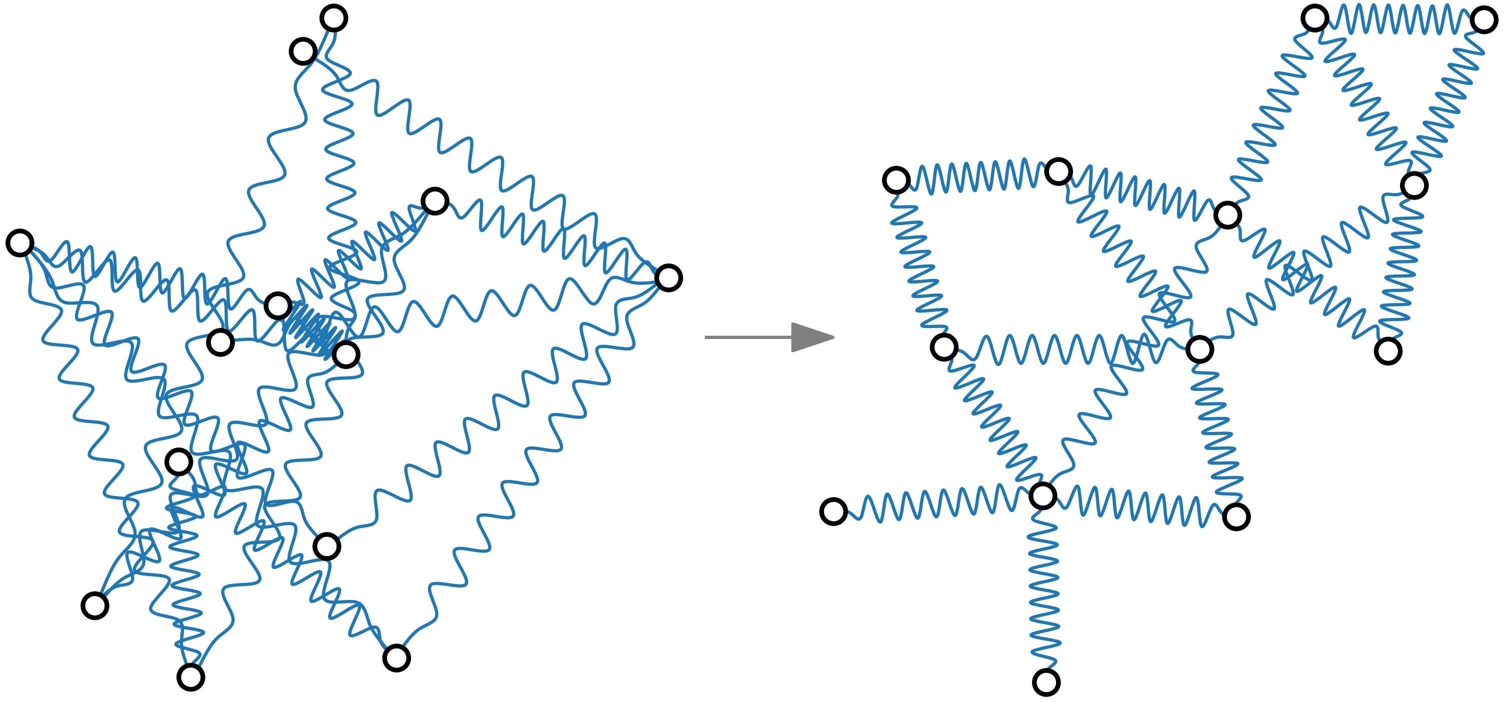
Idea 1.

“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.” [Eades ’84]



Physical analogy

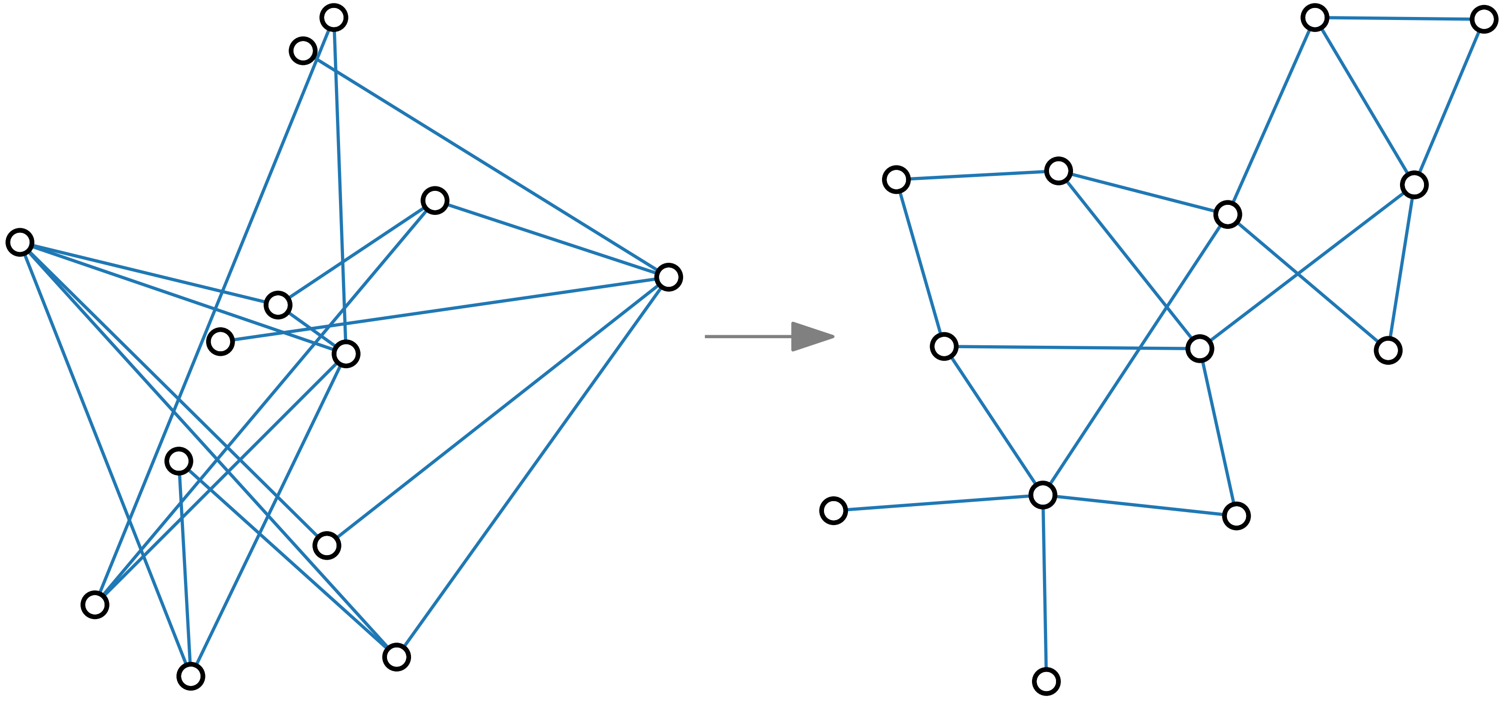
Idea 1.
"To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . . The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state." [Eades '84]



Physical analogy

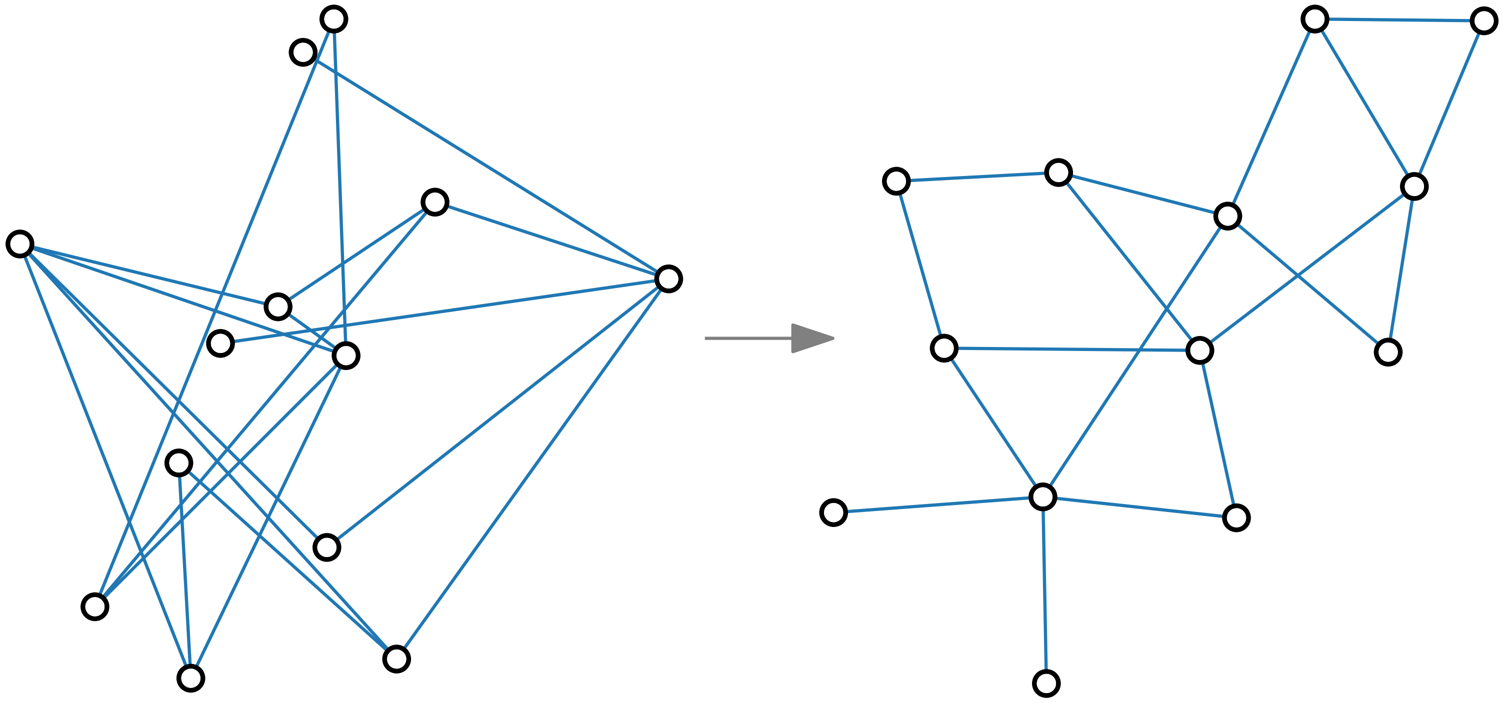
Idea 1.

"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." [Eades '84]



Physical analogy

Idea 1.
“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.” [Eades ’84]

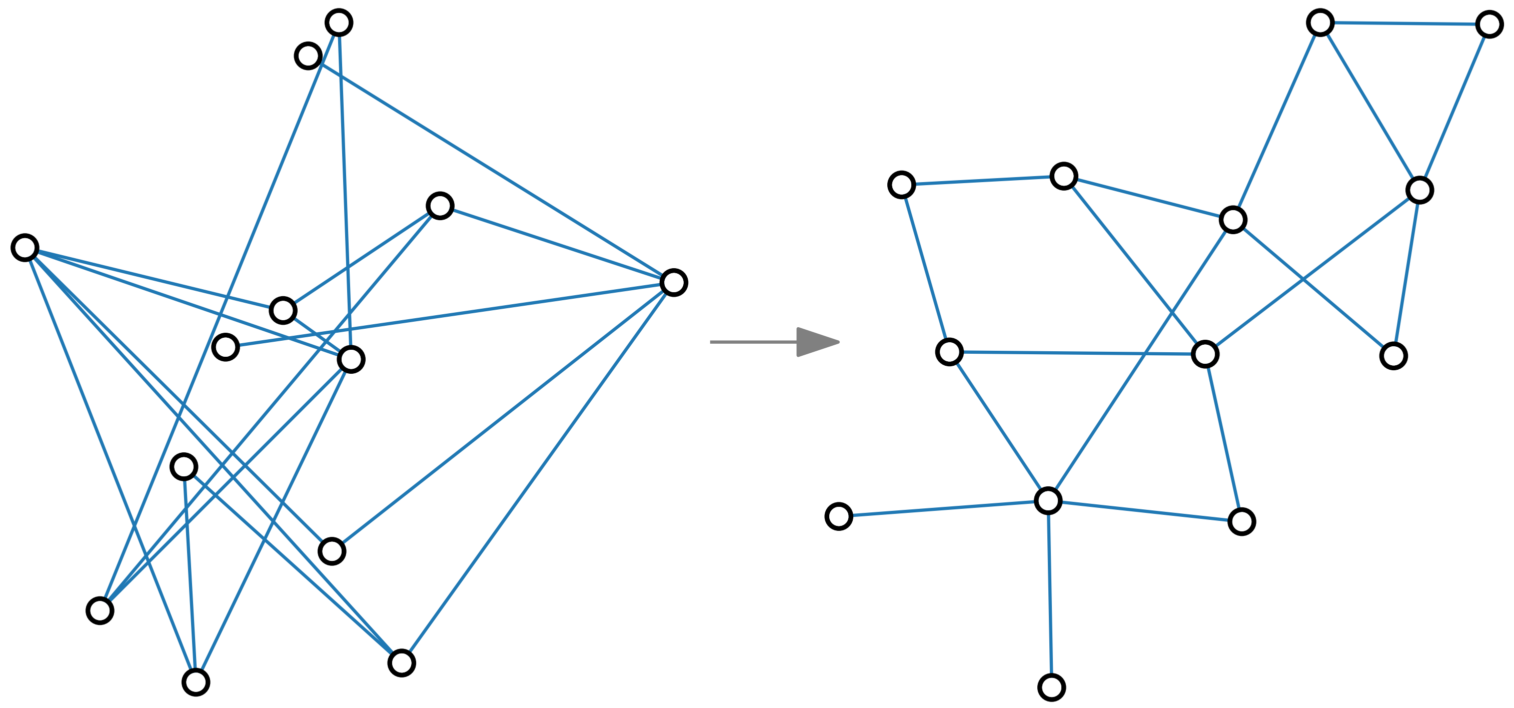


- adjacent vertices u and v:

Physical analogy

Idea 1.

"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." [Eades '84]



- adjacent vertices u and v:

Idea 2.

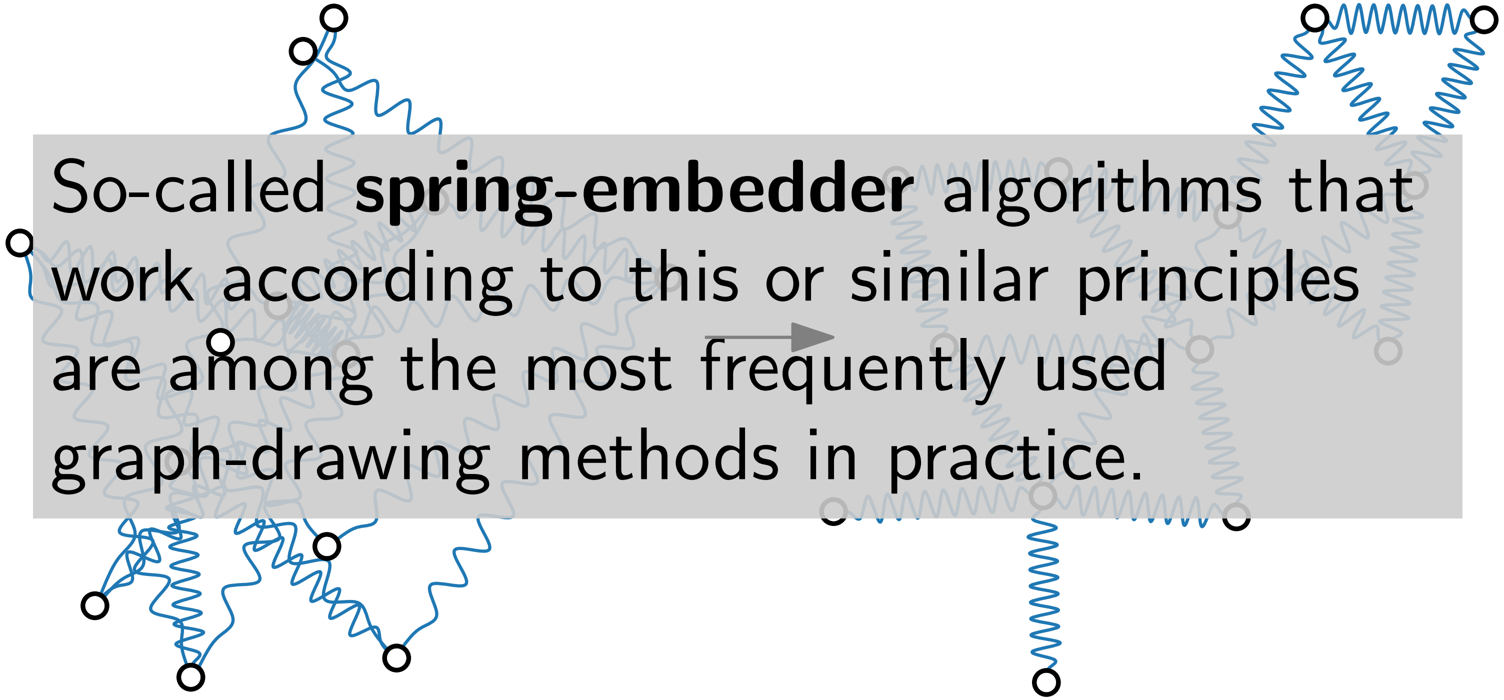
Repulsive forces.

- non-adjacent vertices x and y:

Physical analogy

Idea 1.

"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." [Eades '84]



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

- adjacent vertices u and v:

Idea 2.

Repulsive forces.

- non-adjacent vertices x and y:

Outline

- Spring Embedder by Eades
- Variation by Fruchterman & Reingold
- Ways to speed up computation
- Alternative multidimensional scaling for large graphs

Spring Embedder by Eades – Algorithm

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

return p

Spring Embedder by Eades – Algorithm

initial layout

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

return p

Spring Embedder by Eades – Algorithm

initial layout

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

return p

end layout

Spring Embedder by Eades – Algorithm

initial layout threshold iterations

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

return p

end layout

Spring Embedder by Eades – Algorithm

initial layout · threshold · iterations

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

$t \leftarrow 1$

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

$t \leftarrow t + 1$

return p

end layout

Spring Embedder by Eades – Algorithm

initial layout — threshold — iterations

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

$t \leftarrow 1$

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

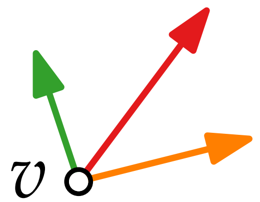
  foreach $v \in V$ **do**

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

  $t \leftarrow t + 1$

return p

end layout



Spring Embedder by Eades – Algorithm

initial layout → $p = (p_v)_{v \in V}$

threshold → $\varepsilon > 0$

iterations → $K \in \mathbb{N}$

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

$t \leftarrow 1$

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

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

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

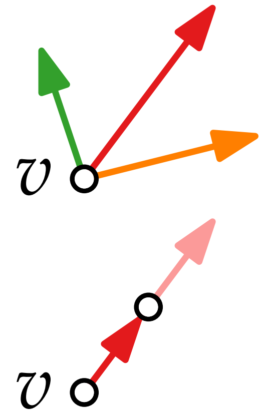
  foreach $v \in V$ **do**

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

  $t \leftarrow t + 1$

return p

end layout → p



Spring Embedder by Eades – Algorithm

initial layout — threshold — iterations

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

  $t \leftarrow 1$

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

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

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

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

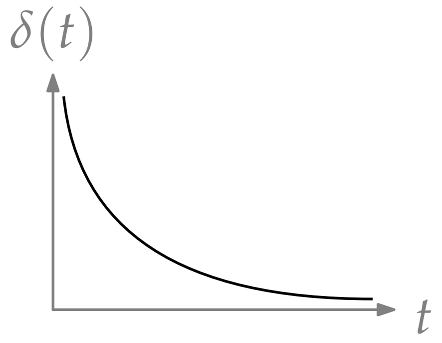
      $p_v \leftarrow p_v + \delta(t) \cdot F_v(t)$

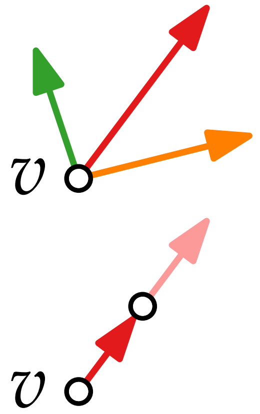
    $t \leftarrow t + 1$

  return p

cooling factor

end layout





Spring Embedder by Eades – Model

Notation.

- $\ell = \ell(e) =$ ideal spring lenght for edge e
- $p_v =$ position of vertex 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 by Eades – 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 node v

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

Notation.

- $\ell = \ell(e) =$ ideal spring lenght for edge e
- $p_v =$ position of vertex 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 by Eades – Model

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

repulsion constant (e.g. 1.0)

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

- 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 node v

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

Notation.

- $\ell = \ell(e)$ = ideal spring lenght for edge e
- p_v = position of vertex 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 by Eades – Model

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

repulsion constant (e.g. 1.0)

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

- attractive force between adjacent vertices u and v

spring constant (e.g. 2.0)

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

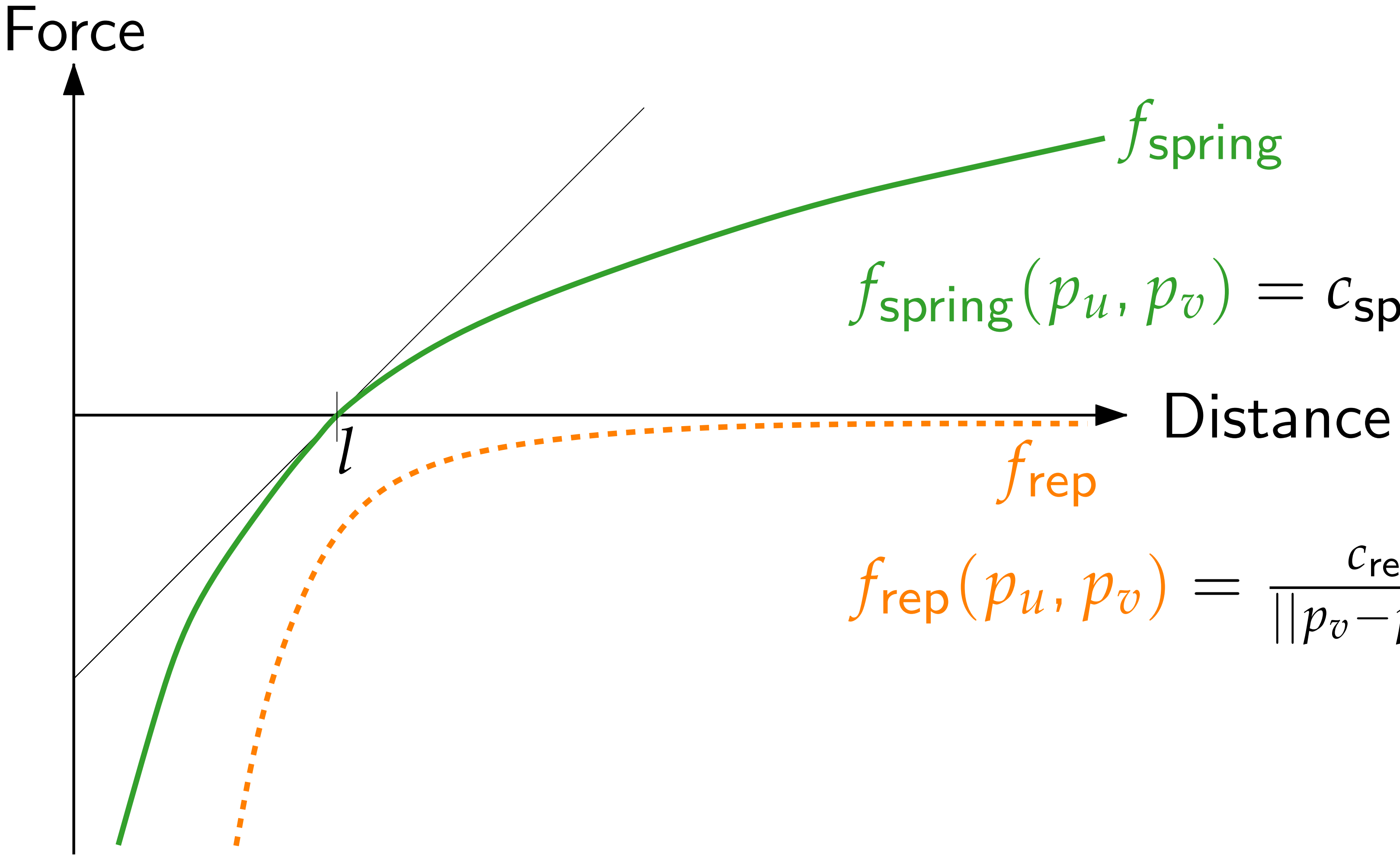
- resulting displacement vector for node v

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

Notation.

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

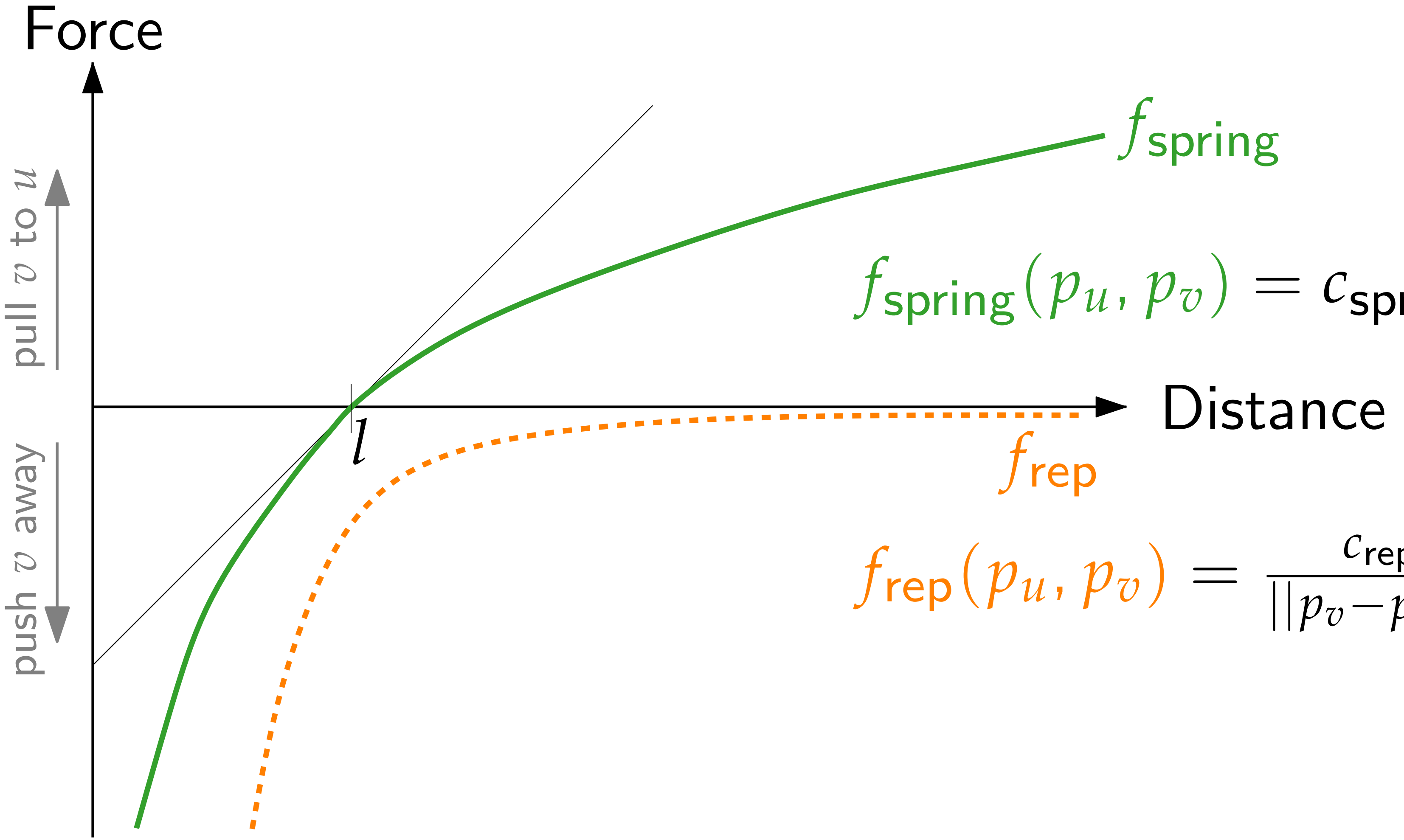
Spring Embedder by Eades – Force diagram



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

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

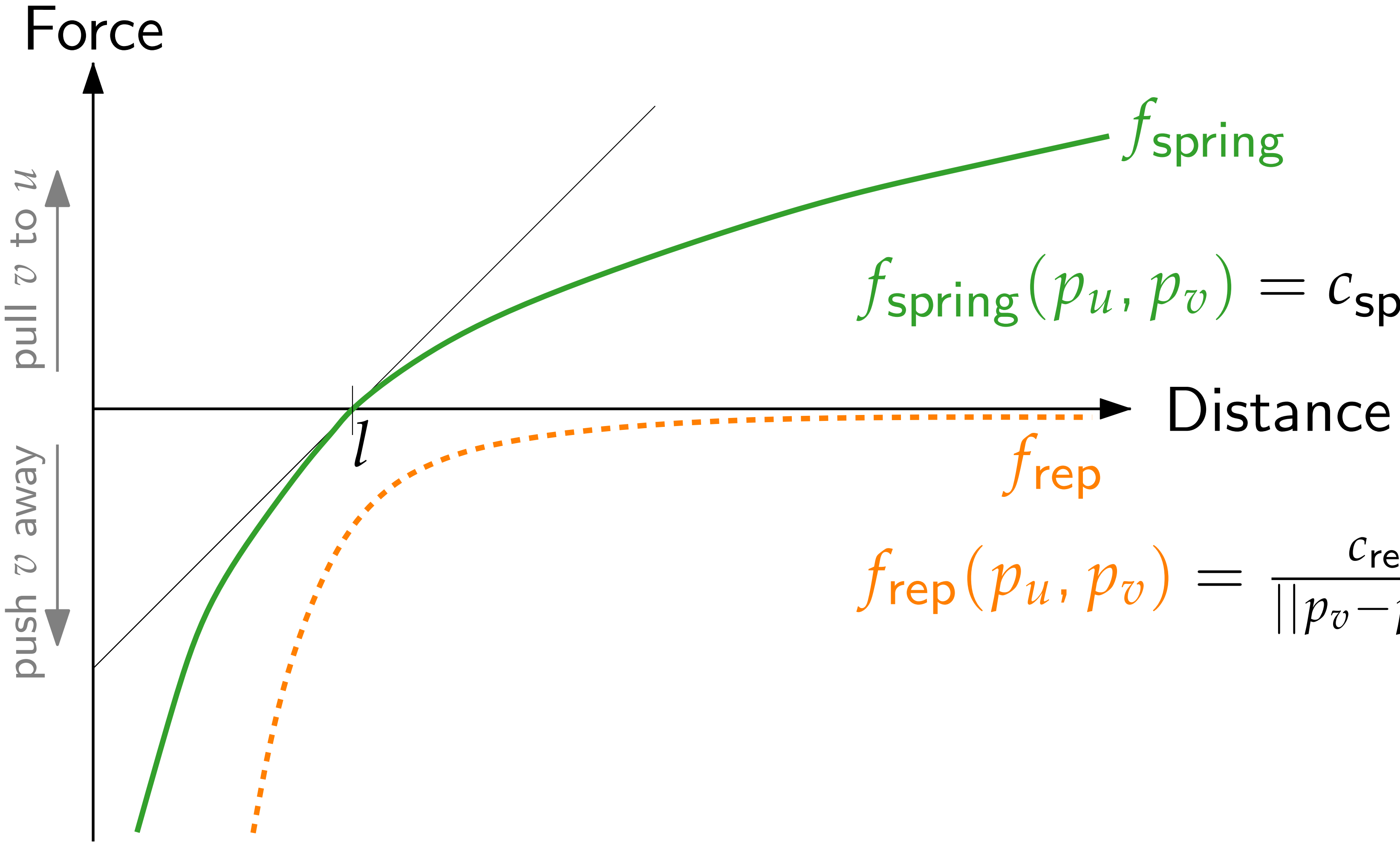
Spring Embedder by Eades – Force diagram



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

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

Spring Embedder by Eades – Force diagram



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

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

Spring Embedder by Eades – Discussion

Advantages.

- very simple algorithm
- good results for small and medium-sized graphs
- empirically good representation of symmetry and structure

Disadvantages.

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

Influence.

- original paper by Peter Eades [Eades '84] got $\sim$ 2000 citations
- basis for many further ideas

Variant by Fruchterman & Reingold

Model.

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

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

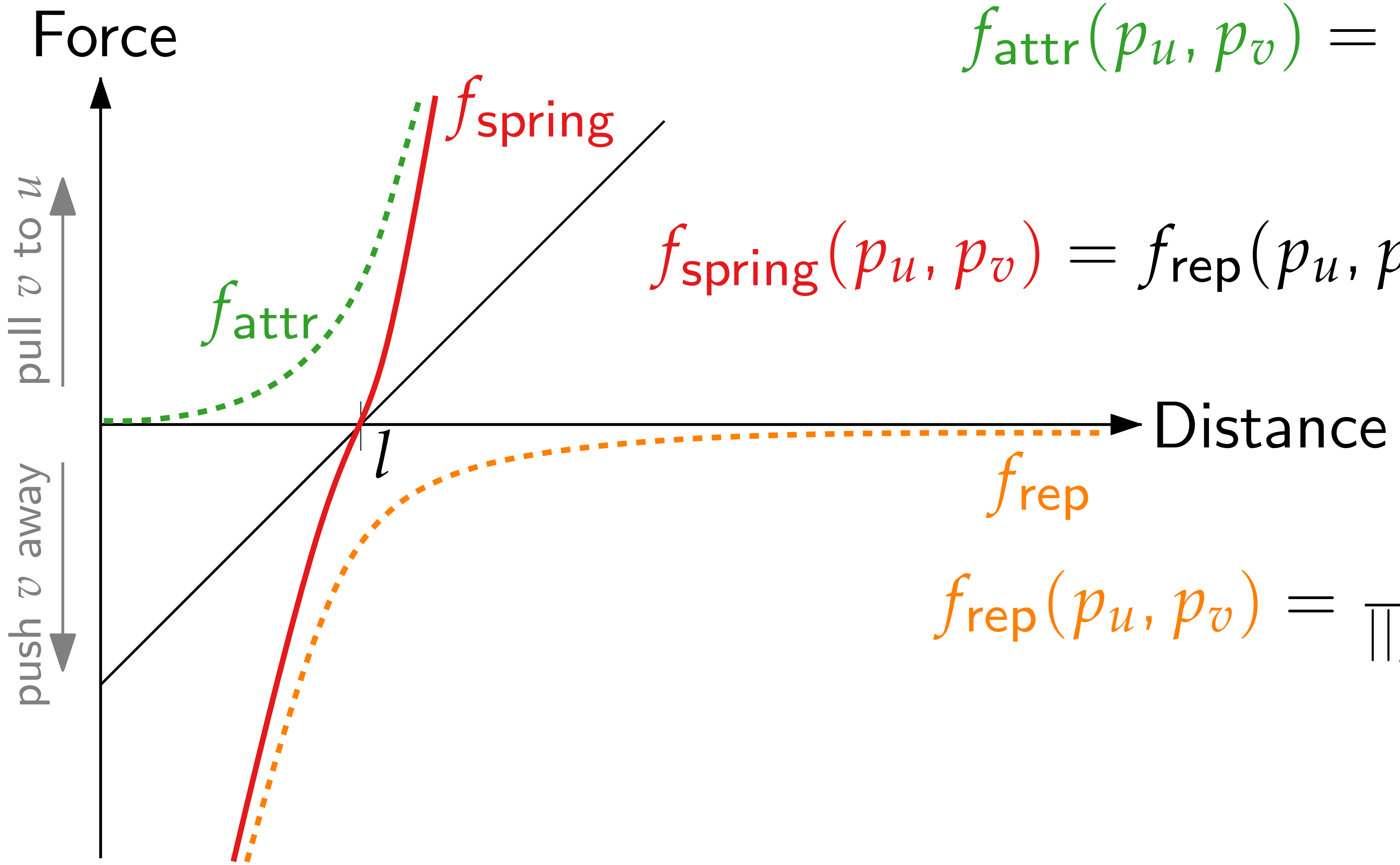
- attractive force between two adjacent vertices u and v

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

- resulting force between adjacent vertices u and v

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

Fruchtermann & Reingold – Force diagram



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

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

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

Adaptability

Inertia.

- Define vertex mass $\Phi(v) = 1 + \text{deg}(v)/2$
- Set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Adaptability

Inertia.

- Define vertex mass $\Phi(v) = 1 + \deg(v)/2$
- Set $f_{\mathsf{attr}}(p_u, p_v) \leftarrow f_{\mathsf{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Gravitation.

- Define centroid $p_{\mathsf{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
- Add force $f_{\mathsf{grav}}(p_v) = c_{\mathsf{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\mathsf{bary}}}$

Adaptability

Inertia.

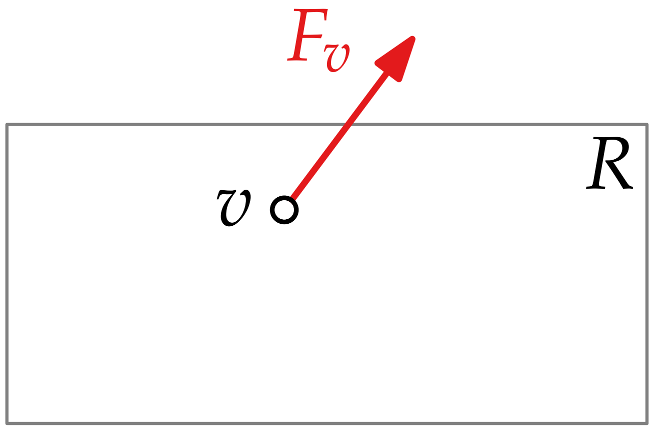
- Define vertex mass $\Phi(v) = 1 + \deg(v)/2$
- Set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Gravitation.

- Define centroid $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
- Add force $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$

Restricted drawing area.

If F_v points beyond area R, clip vector appropriately at the border of R.



Adaptability

Inertia.

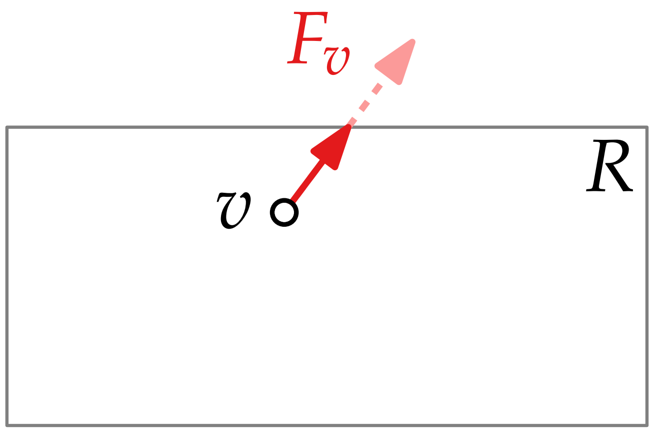
- Define vertex mass $\Phi(v) = 1 + \deg(v)/2$
- Set $f_{\mathsf{attr}}(p_u, p_v) \leftarrow f_{\mathsf{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Gravitation.

- Define centroid $p_{\mathsf{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
- Add force $f_{\mathsf{grav}}(p_v) = c_{\mathsf{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\mathsf{bary}}}$

Restricted drawing area.

If F_v points beyond area R, clip vector appropriately at the border of R.



Adaptability

Inertia.

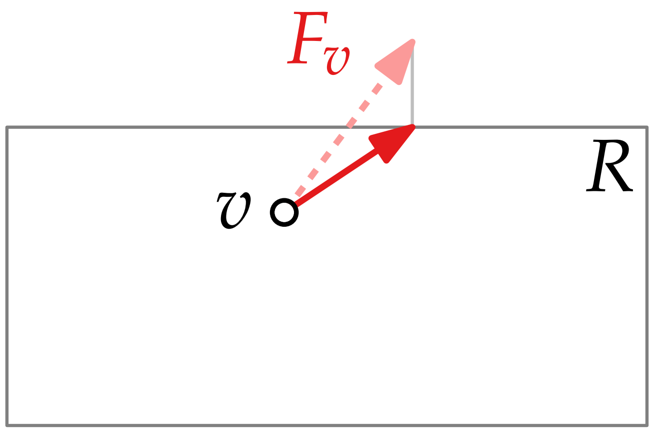
- Define vertex mass $\Phi(v) = 1 + \deg(v)/2$
- Set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Gravitation.

- Define centroid $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
- Add force $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$

Restricted drawing area.

If F_v points beyond area R, clip vector appropriately at the border of R.



Adaptability

Inertia.

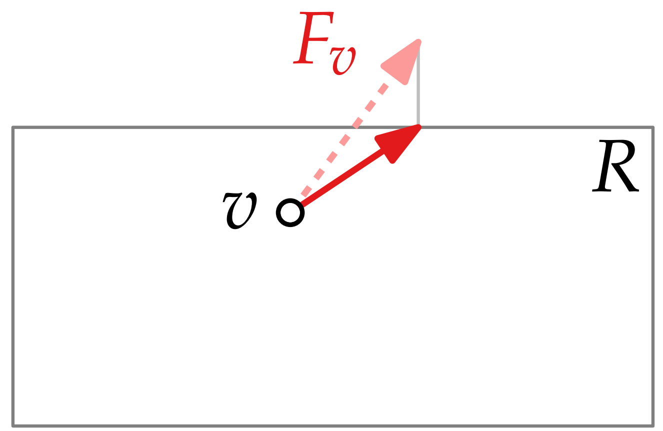
- Define vertex mass $\Phi(v) = 1 + \deg(v)/2$
- Set $f_{\mathsf{attr}}(p_u, p_v) \leftarrow f_{\mathsf{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

Gravitation.

- Define centroid $p_{\mathsf{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
- Add force $f_{\mathsf{grav}}(p_v) = c_{\mathsf{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\mathsf{bary}}}$

Restricted drawing area.

If F_v points beyond area R, clip vector appropriately at the border of R.



And many more...

- magnetic orientation of edges [GD Ch. 10.4]
- other energy models
- planarity preserving
- speedups

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

Reminder...

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

$t \leftarrow 1$

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

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

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

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

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

  $t \leftarrow t + 1$

return p

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

Reminder...

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

$t \leftarrow 1$

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

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

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

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

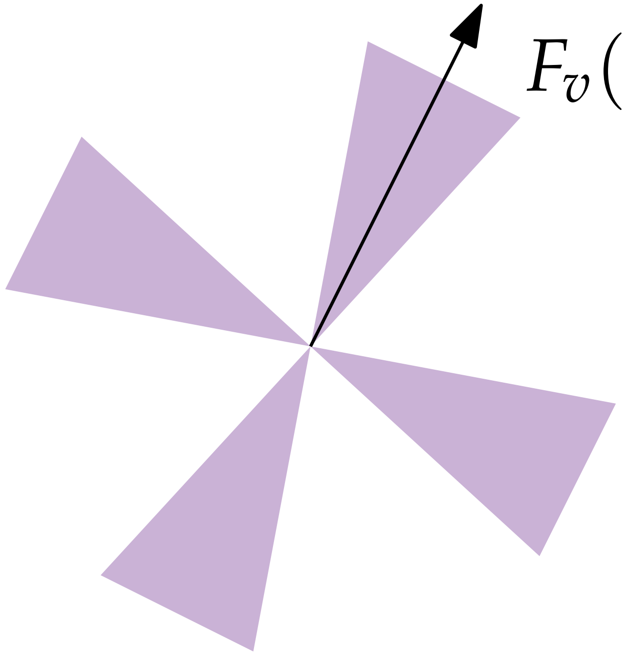
    $p_v \leftarrow p_v + \delta(t) \cdot F_v(t)$

  $t \leftarrow t + 1$ $\delta_v(t)$

return p

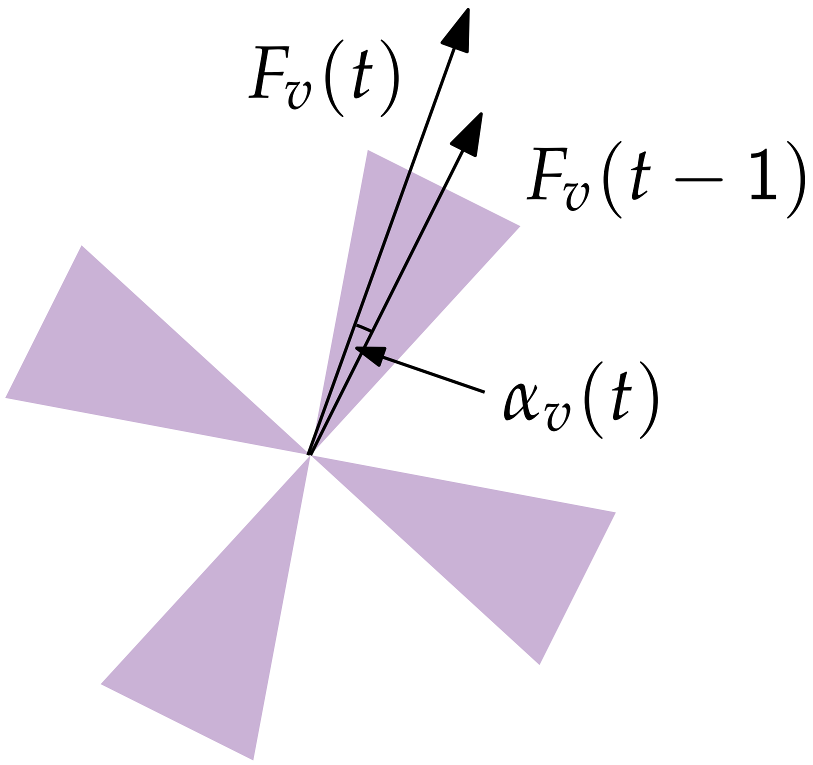
Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]



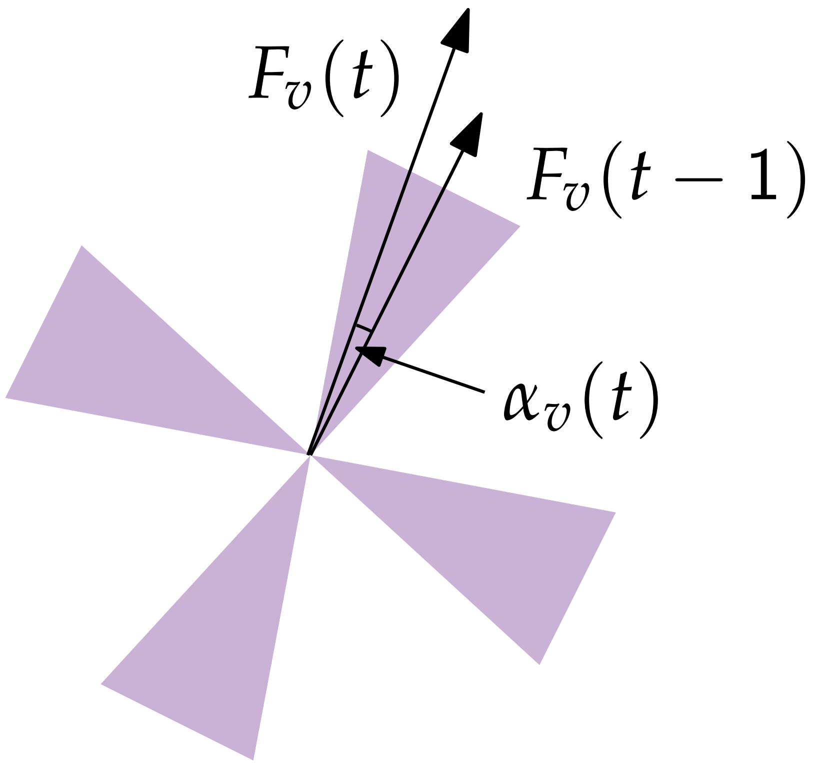
Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]



Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]

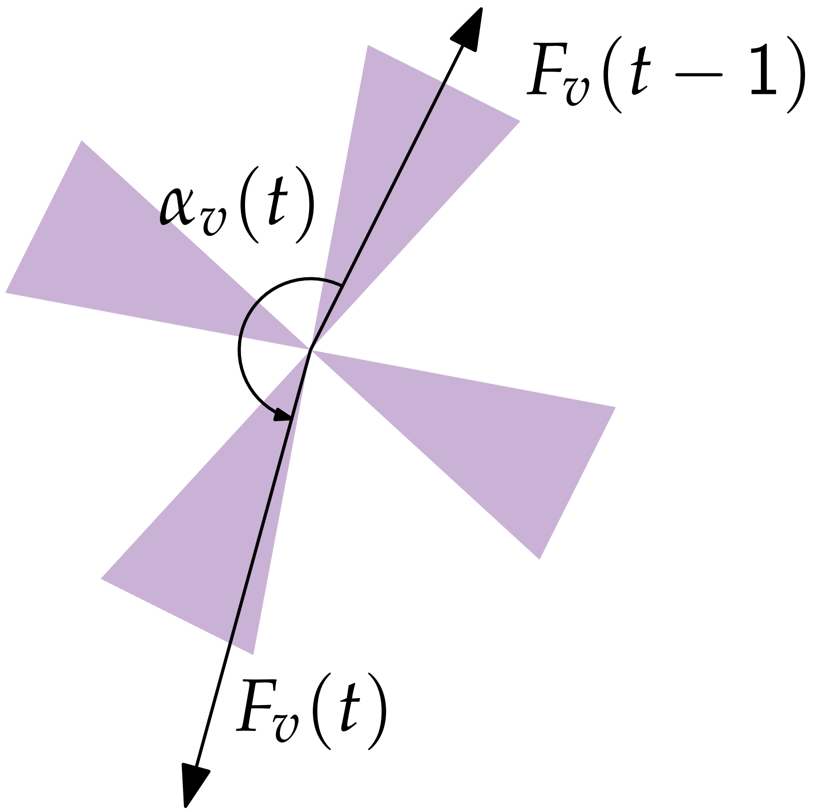


Same direction.

$\rightarrow$ increase temperature $\delta_v(t)$

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]

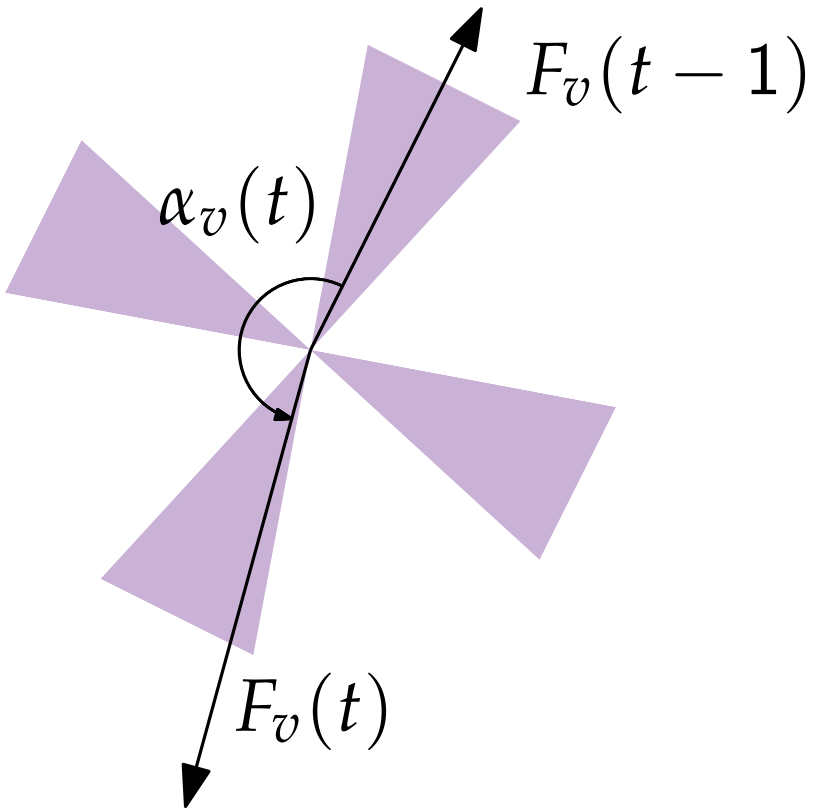


Same direction.

$\rightarrow$ increase temperature $\delta_v(t)$

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]

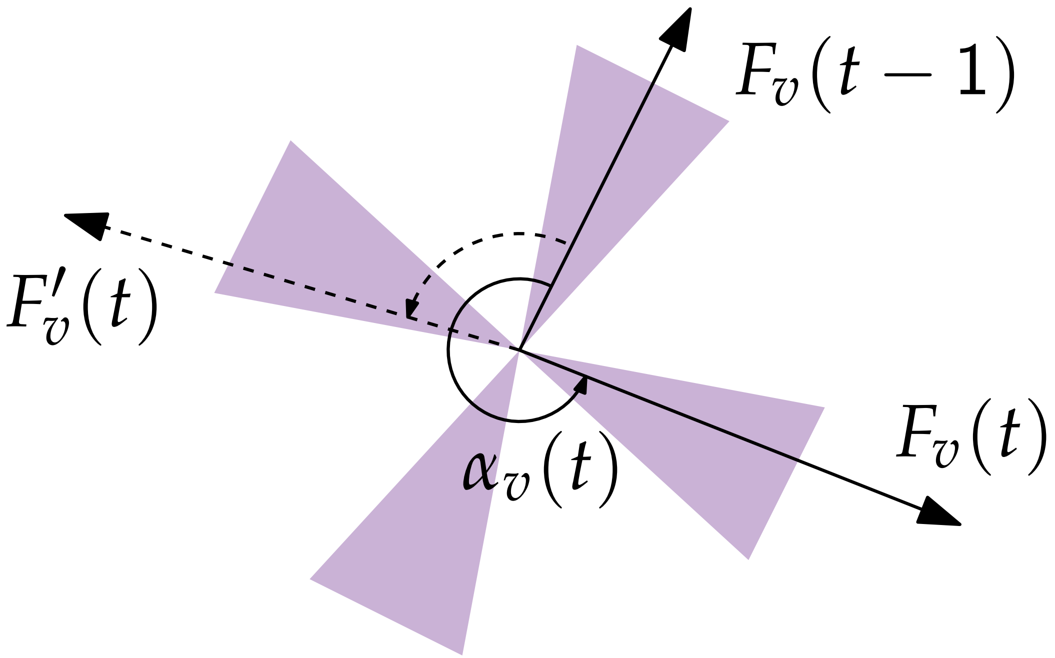


Same direction.
$\rightarrow$ increase temperature $\delta_v(t)$

Oszillation.
$\rightarrow$ decrease temperature $\delta_v(t)$

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]

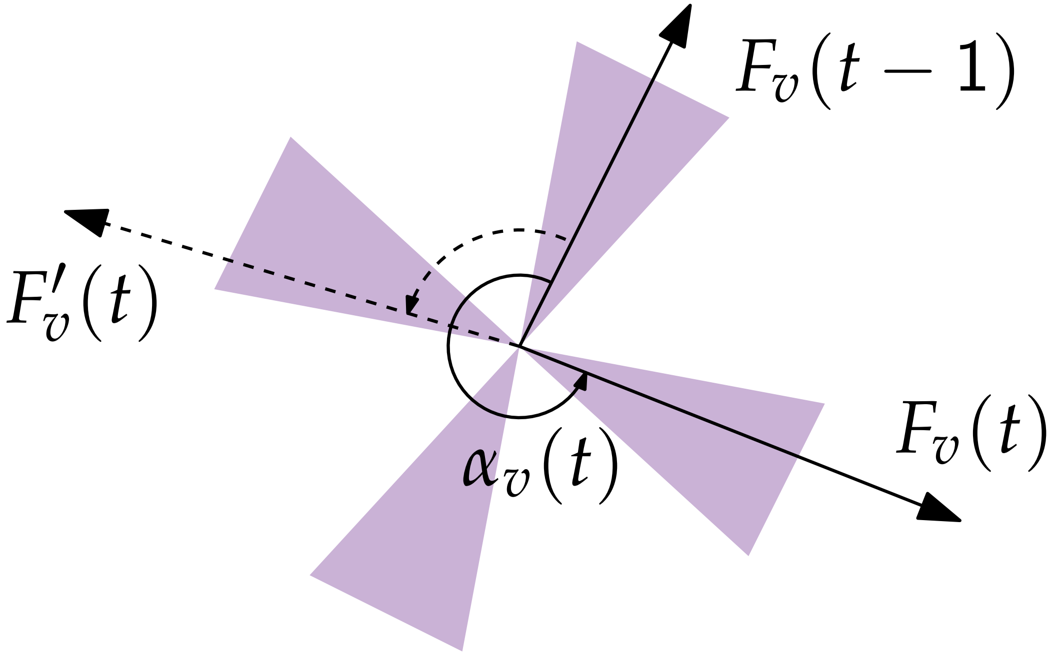


Same direction.
$\rightarrow$ increase temperature $\delta_v(t)$

Oszillation.
$\rightarrow$ decrease temperature $\delta_v(t)$

Speeding up "convergence" by adaptive displacement $\delta_v(t)$

[Frick, Ludwig, Mehldau '95]



Same direction.
$\rightarrow$ increase temperature $\delta_v(t)$

Oszillation.
$\rightarrow$ decrease temperature $\delta_v(t)$

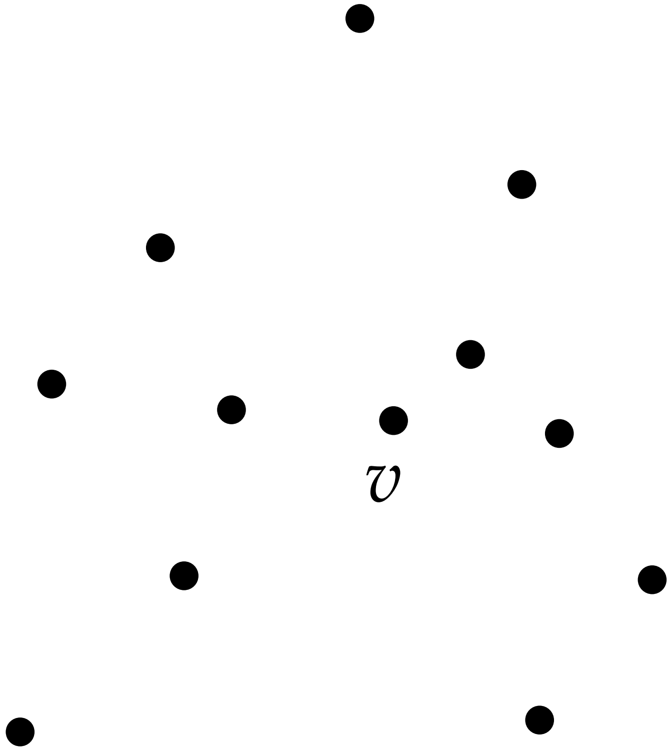
Rotation.

- count rotations
- if applicable

$\rightarrow$ decrease temperature $\delta_v(t)$

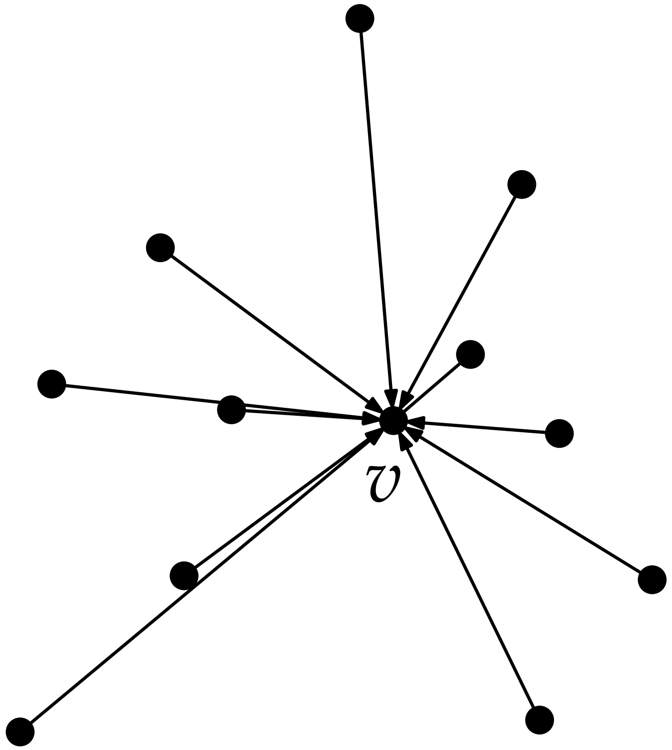
Speeding up "convergence" via grids

[Fruchterman & Reingold '91]



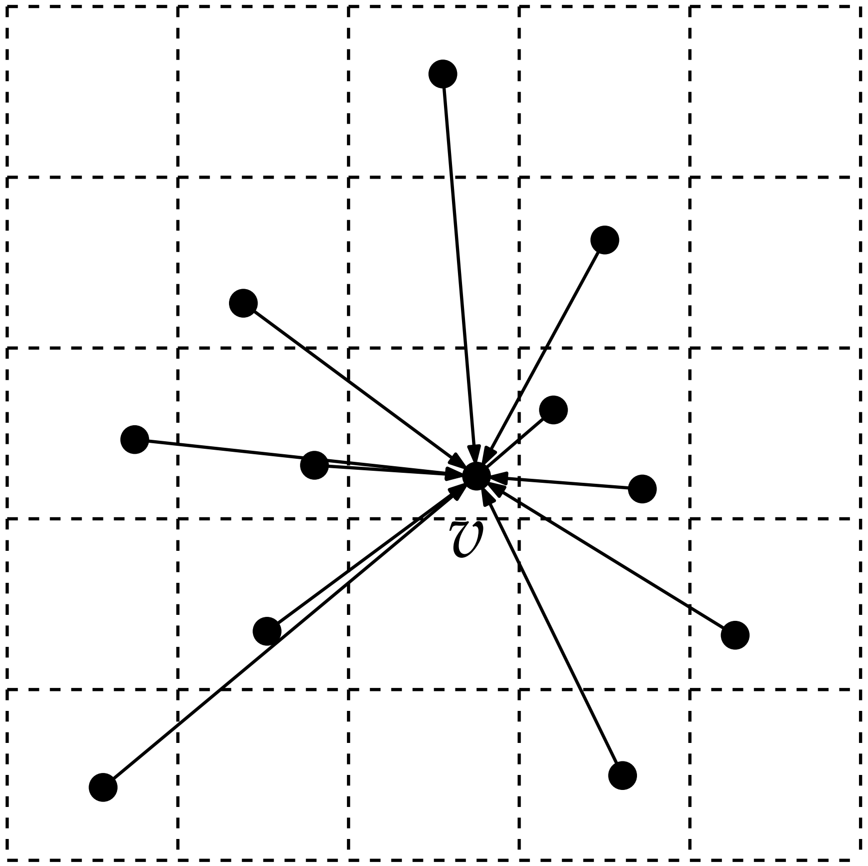
Speeding up “convergence” via grids

[Fruchterman & Reingold ’91]



Speeding up "convergence" via grids

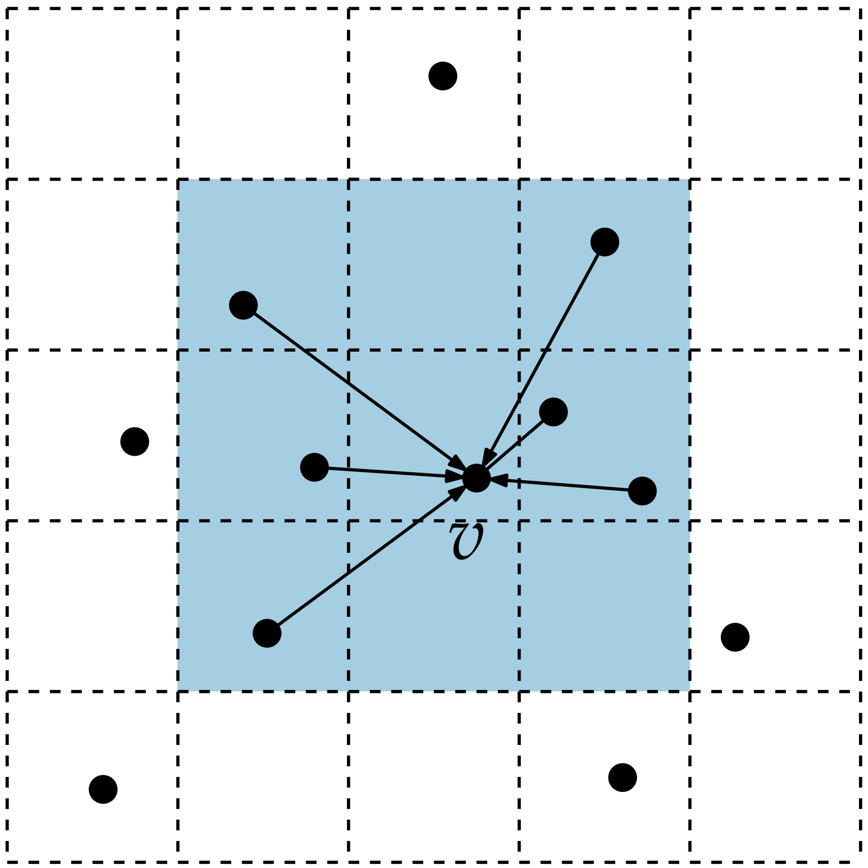
[Fruchterman & Reingold '91]



- divide plane into grid

Speeding up "convergence" via grids

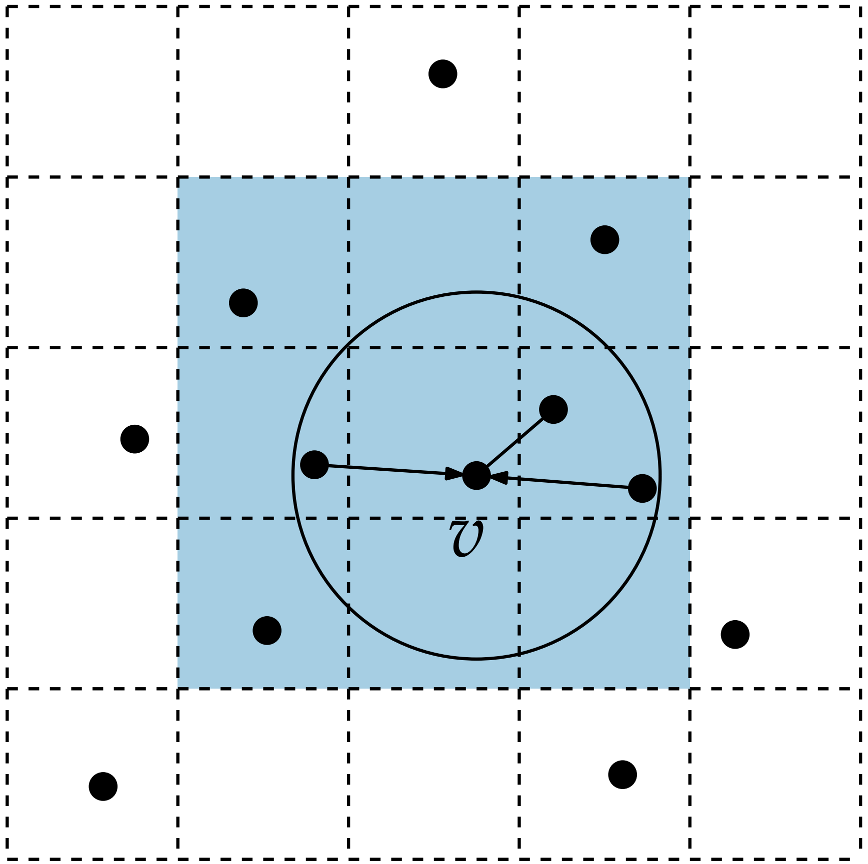
[Fruchterman & Reingold '91]



- divide plane into grid
- consider repelling forces only to vertices in neighboring cells

Speeding up "convergence" via grids

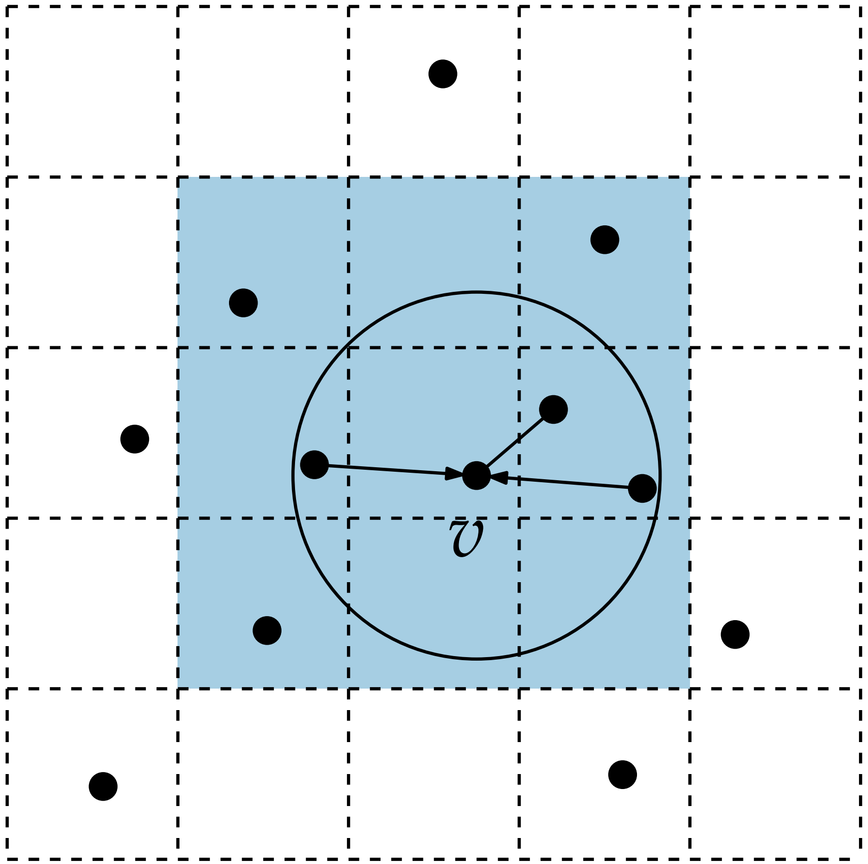
[Fruchterman & Reingold '91]



- divide plane into grid
- consider repelling forces only to vertices in neighboring cells
- and only if distance is less than some max distance

Speeding up "convergence" via grids

[Fruchterman & Reingold '91]



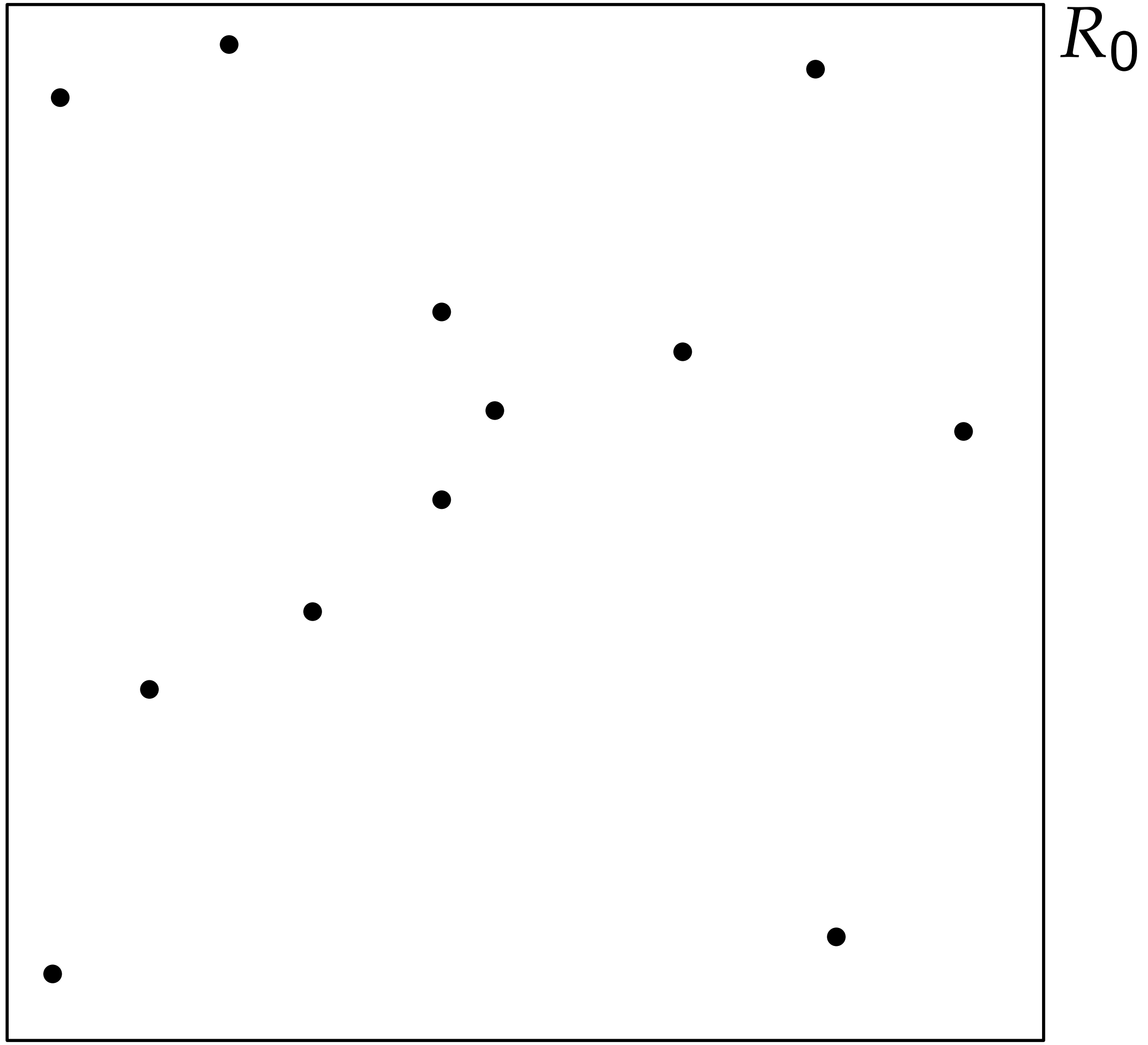
- divide plane into grid
- consider repelling forces only to vertices in neighboring cells
- and only if distance is less than some max distance

Discussion.

- good idea to improve runtime
- worst-case has not improved
- might introduce oszillation and thus a quality loss

Speeding up with quad trees

[Barnes, Hut '86]

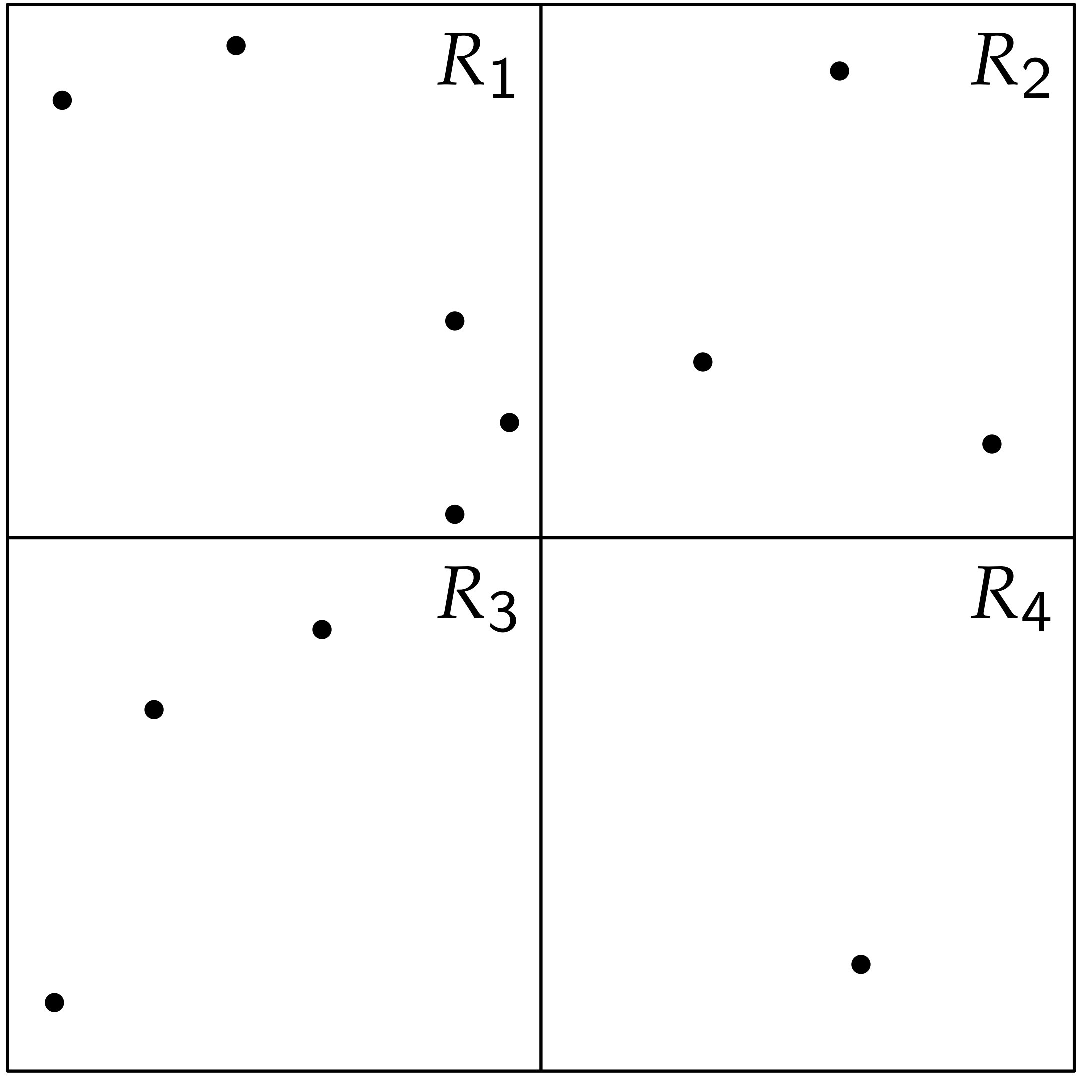


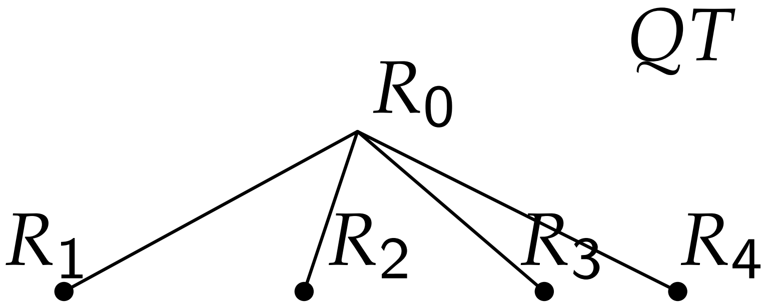
QT

R_0

Speeding up with quad trees

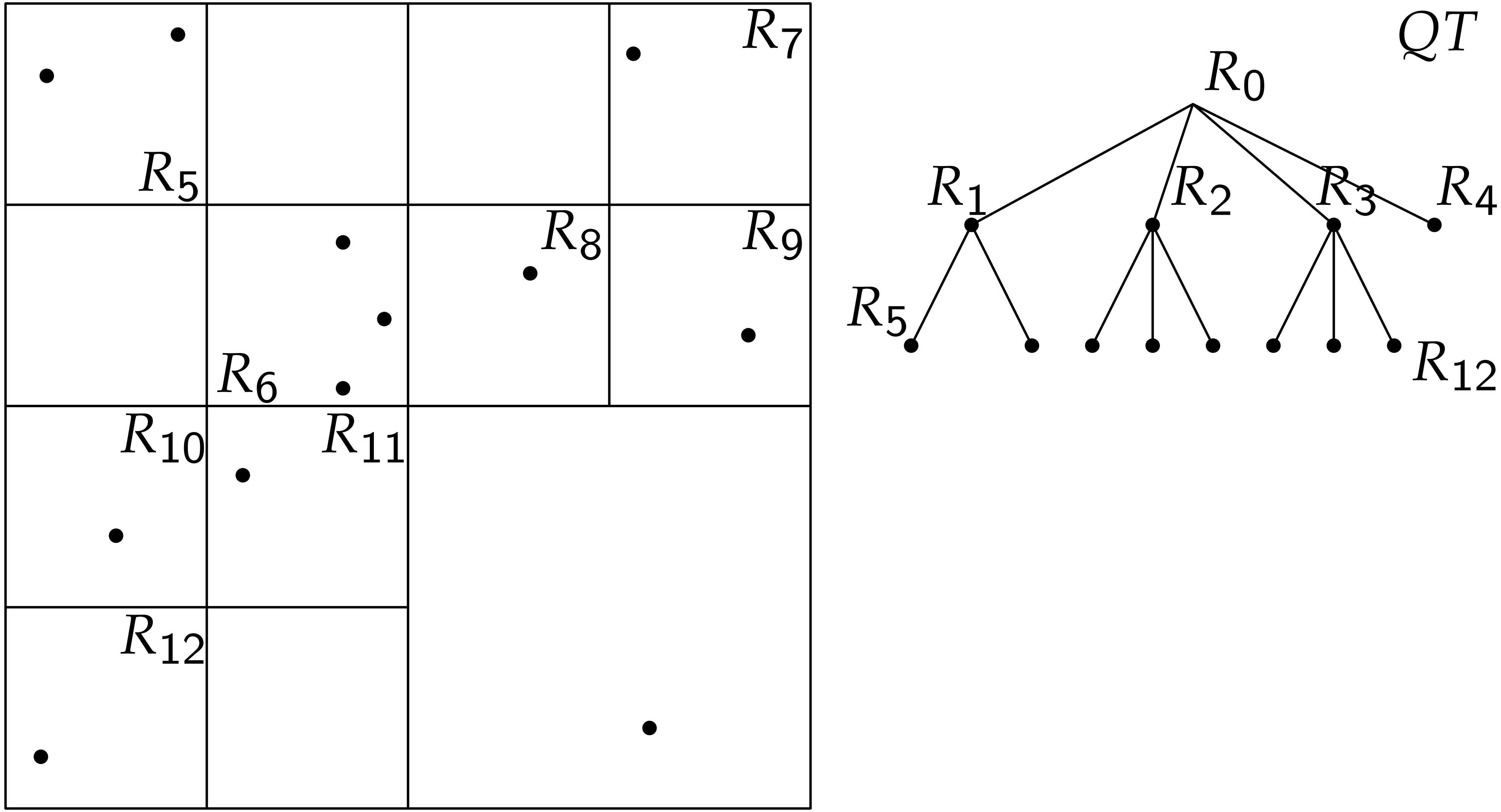
[Barnes, Hut '86]





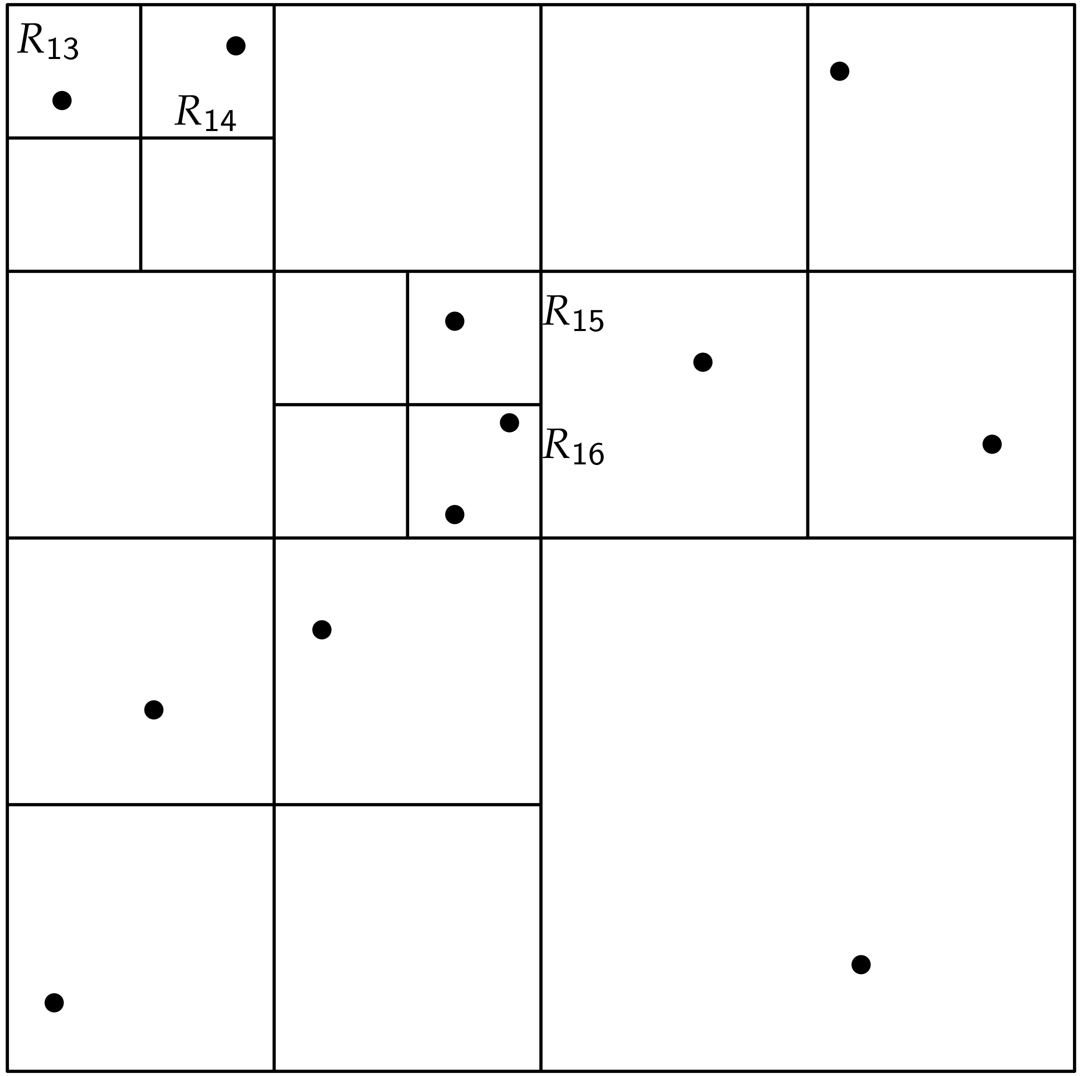
Speeding up with quad trees

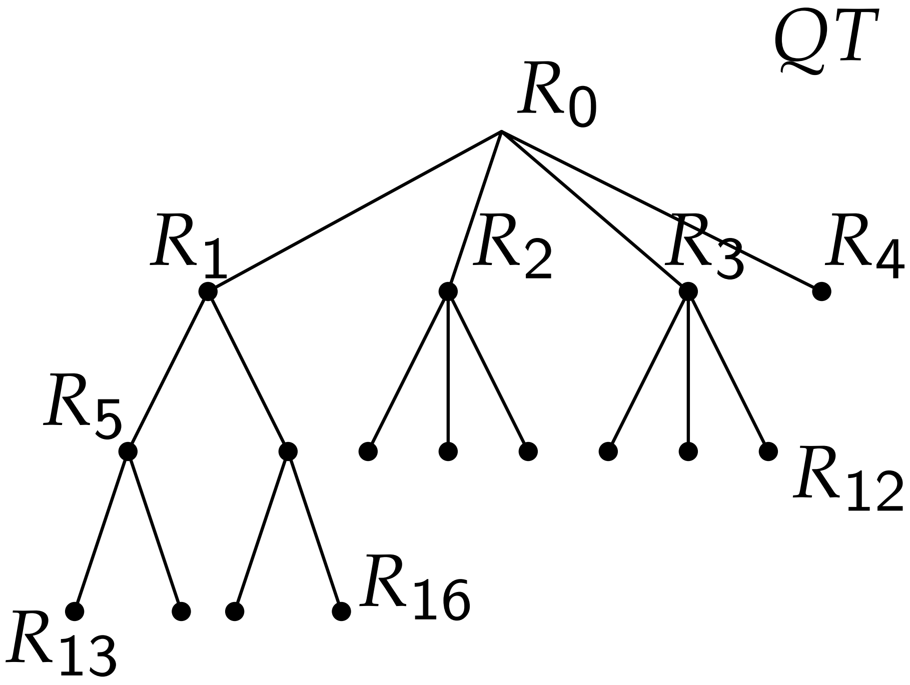
[Barnes, Hut ’86]



Speeding up with quad trees

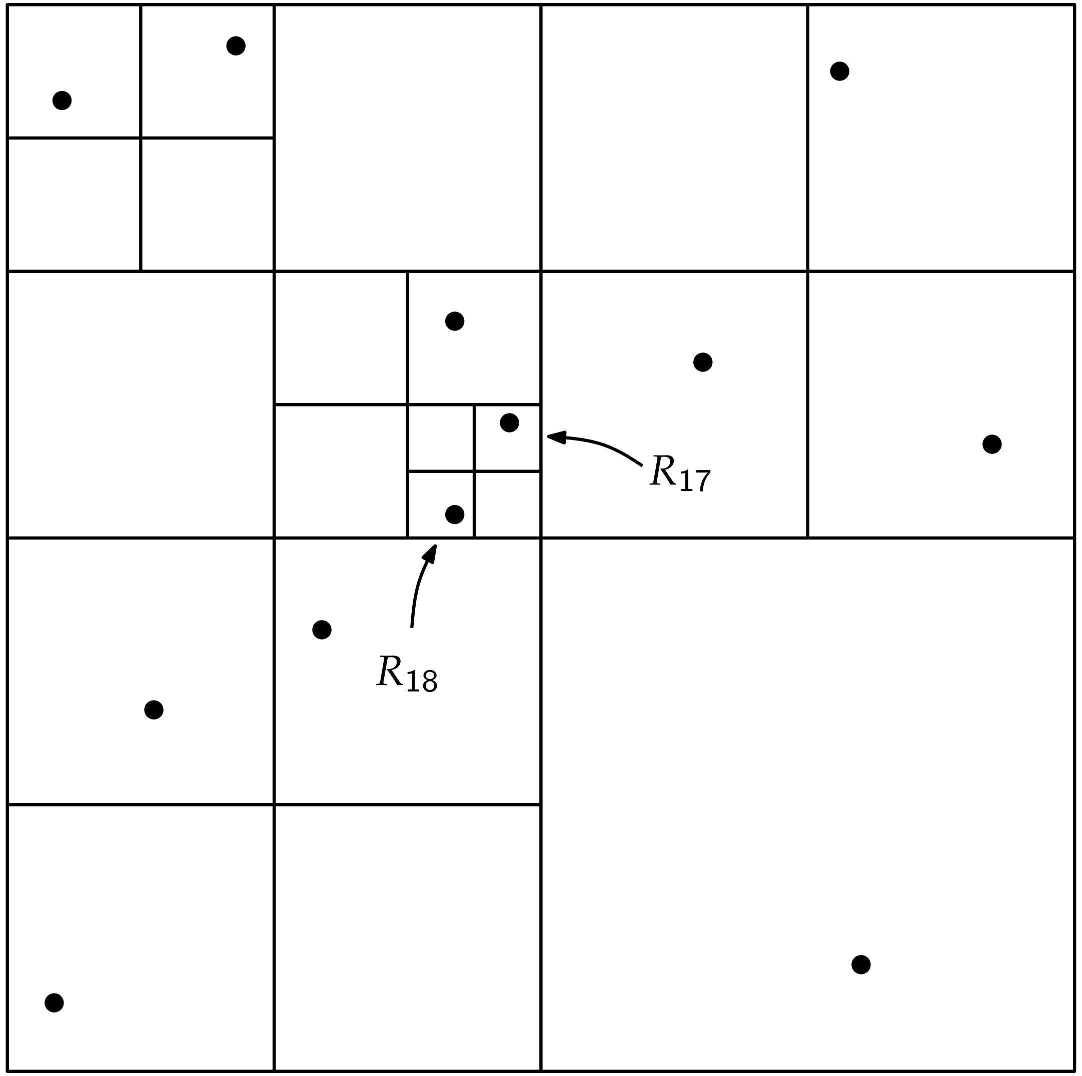
[Barnes, Hut '86]

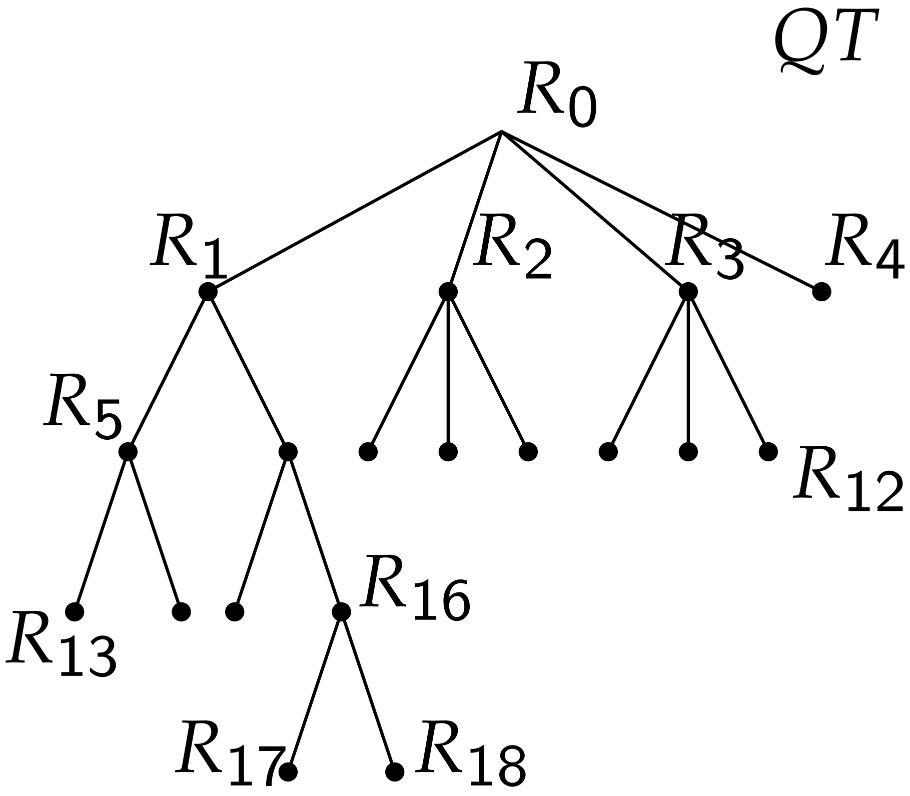




Speeding up with quad trees

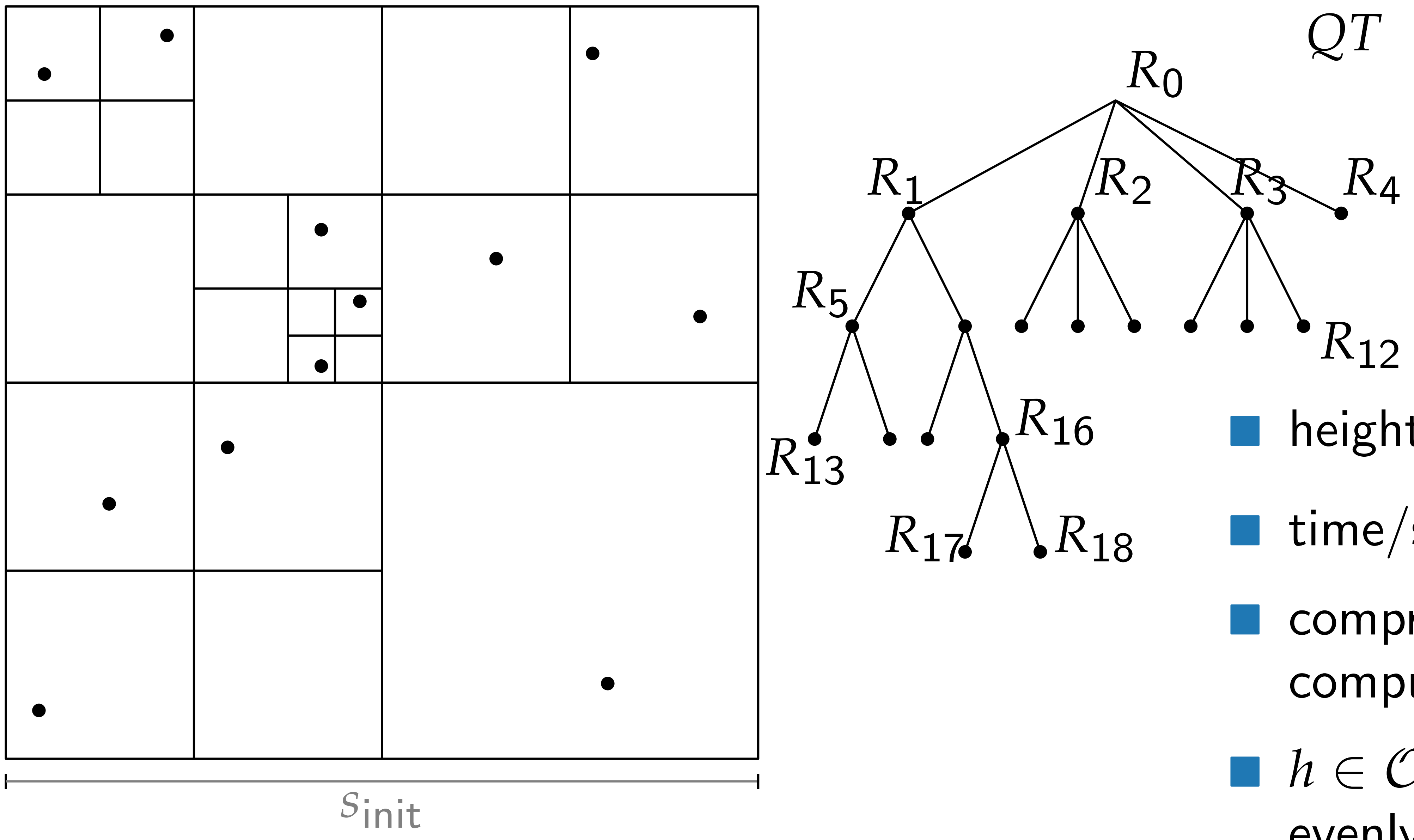
[Barnes, Hut '86]





Speeding up with quad trees

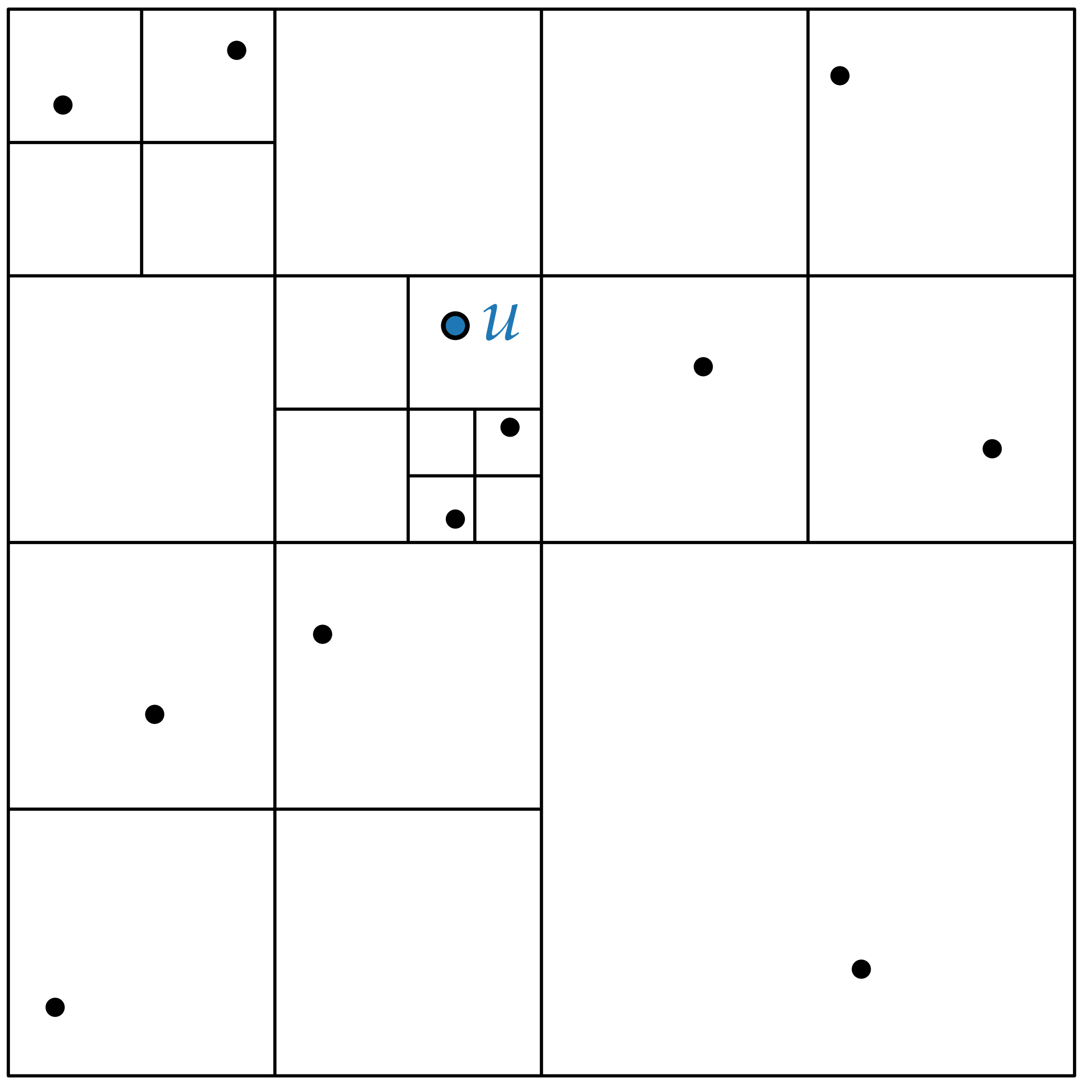
[Barnes, Hut '86]

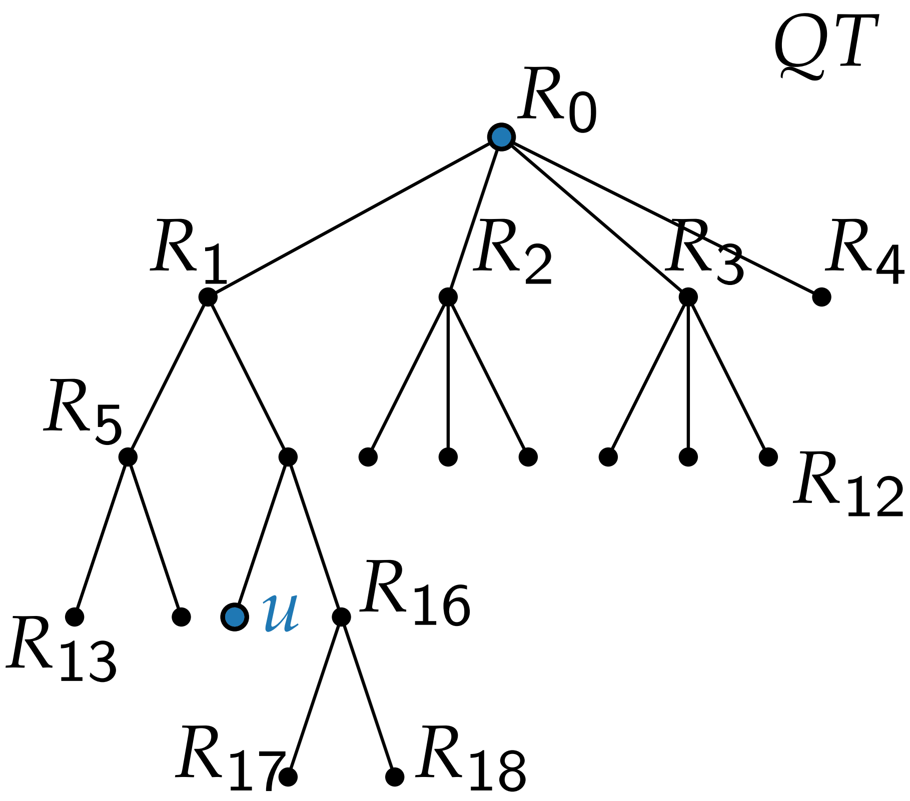


- height $h \leq \log \frac{s_{\text{init}}}{d_{\min}} + \frac{3}{2}$
- time/space in $\mathcal{O}(hn)$
- compressed quad tree can be computed in $\mathcal{O}(n \log n)$ time
- $h \in \mathcal{O}(\log n)$ if vertices evenly distriputed

Speeding up with quad trees

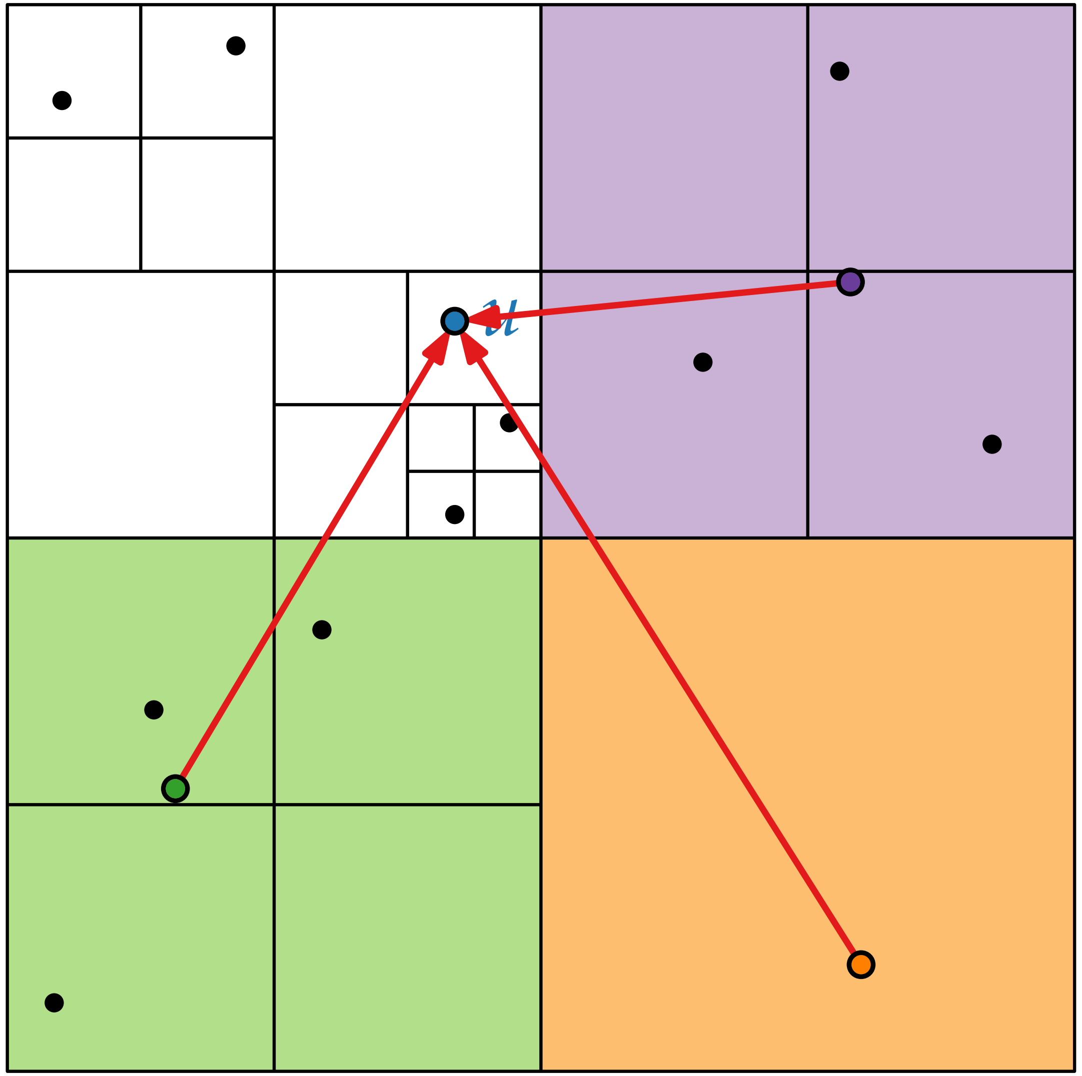
[Barnes, Hut '86]

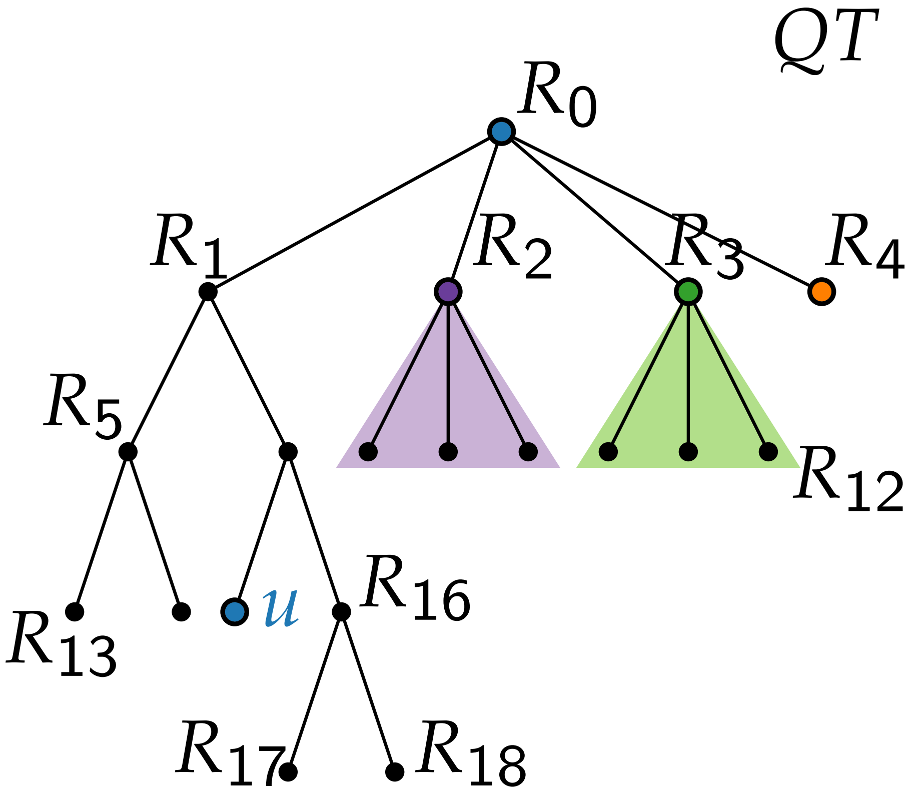




Speeding up with quad trees

[Barnes, Hut '86]

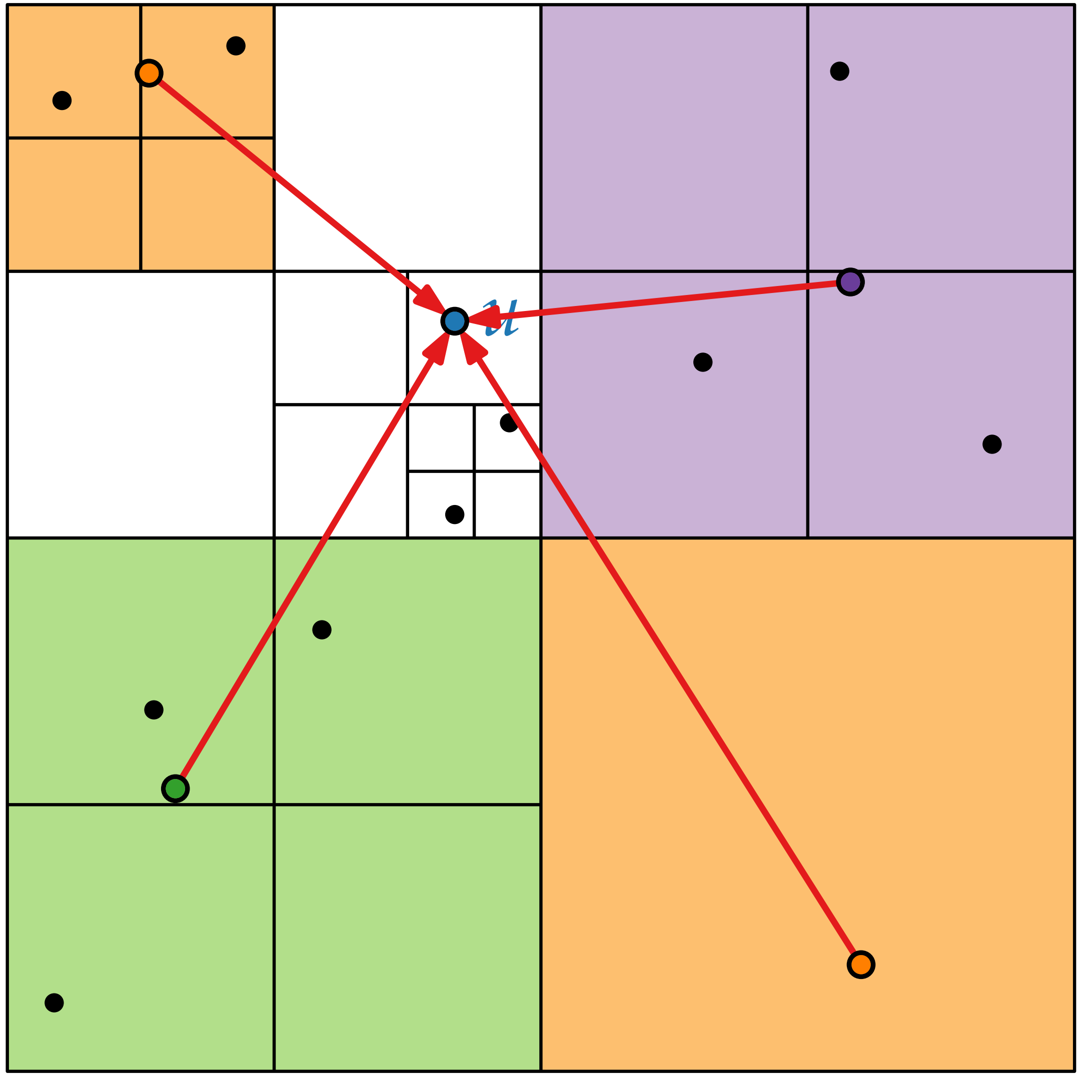


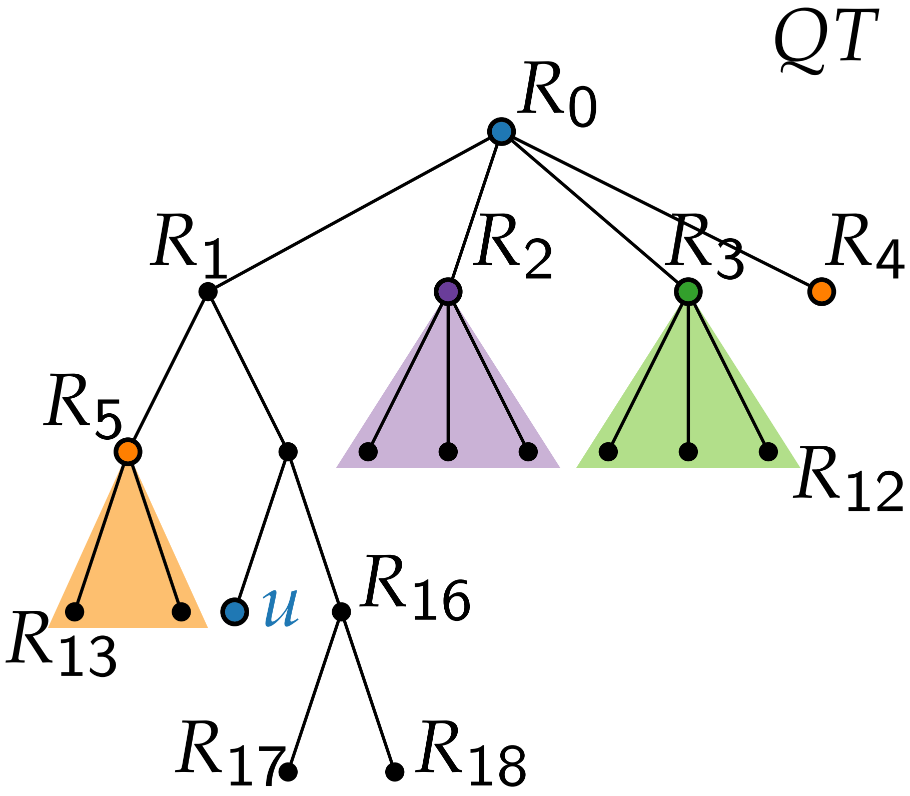


$$f_{\text{rep}}(R_i, p_u) = |R_i| \cdot f_{\text{rep}}(\sigma_{R_i}, p_u)$$

Speeding up with quad trees

[Barnes, Hut '86]

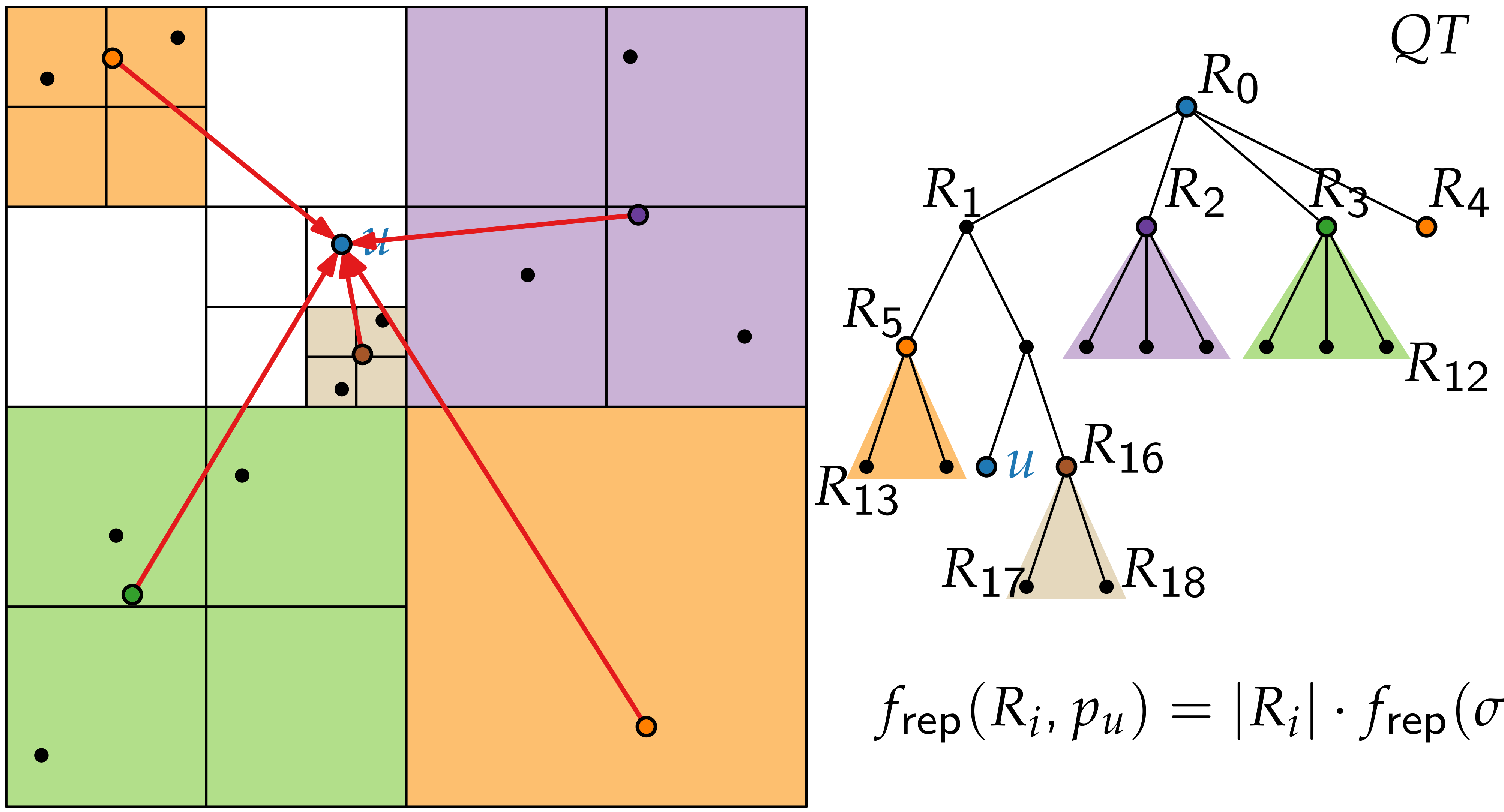




$$f_{\text{rep}}(R_i, p_u) = |R_i| \cdot f_{\text{rep}}(\sigma_{R_i}, p_u)$$

Speeding up with quad trees

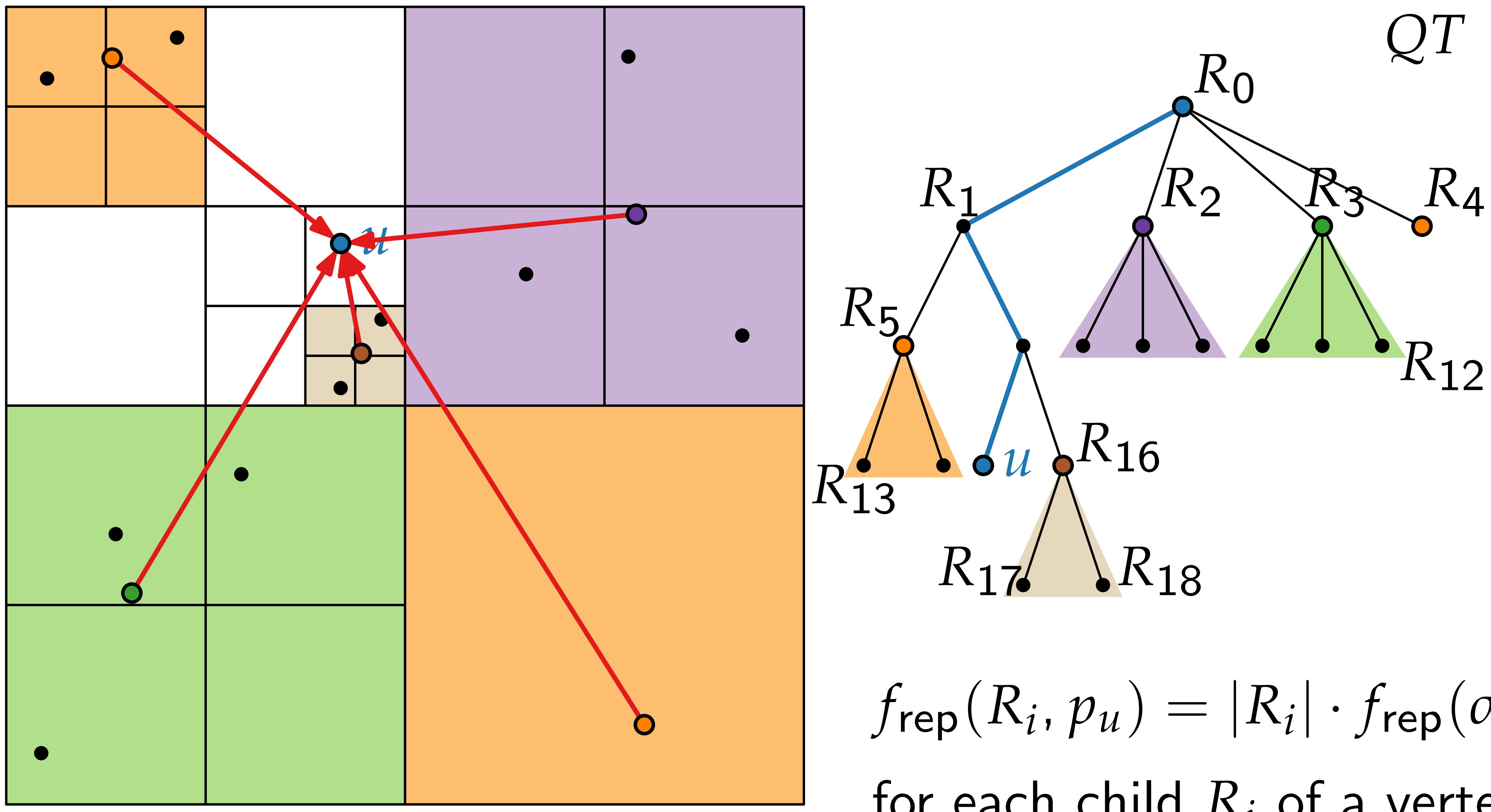
[Barnes, Hut ’86]



$$f_{\text{rep}}(R_i, p_u) = |R_i| \cdot f_{\text{rep}}(\sigma_{R_i}, p_u)$$

Speeding up with quad trees

[Barnes, Hut '86]



$$f_{\text{rep}}(R_i, p_u) = |R_i| \cdot f_{\text{rep}}(\sigma_{R_i}, p_u)$$

for each child R_i of a vertex on path from u to R_0

Multidimensional scaling

- Force-directed method reaches its limitations for large graphs

Multidimensional scaling

- Force-directed method reaches its limitations for large graphs

Idea.

Adapt the classical approach **multidimensional scaling (MDS)**:

- MDS is a technique to visualise similarity among a set of objects
- Input is a distance matric D with $d_{ij} \sim$ dissimilarity between objects i and j
- We search for points $x_1, \ldots, x_n \in \mathbb{R}^2$ such that

$$||x_i - x_j|| \approx d_{ij}$$

Multidimensional scaling

- Force-directed method reaches its limitations for large graphs

Idea.

Adapt the classical approach **multidimensional scaling (MDS)**:

- MDS is a technique to visualise similarity among a set of objects
- Input is a distance matric D with $d_{ij} \sim$ dissimilarity between objects i and j
- We search for points $x_1, \ldots, x_n \in \mathbb{R}^2$ such that

$$||x_i - x_j|| \approx d_{ij}$$

For our drawing, how do we define the dissimilarity between two vertices?

Multidimensional scaling

- Force-directed method reaches its limitations for large graphs

Idea.

Adapt the classical approach **multidimensional scaling (MDS)**:

- MDS is a technique to visualise similarity among a set of objects
- Input is a distance matric D with $d_{ij} \sim$ dissimilarity between objects i and j
- We search for points $x_1, \ldots, x_n \in \mathbb{R}^2$ such that

$$||x_i - x_j|| \approx d_{ij}$$

For our drawing, how do we define the dissimilarity between two vertices?

Multidimensional scaling

- Force-directed method reaches its limitations for large graphs

Idea.

Adapt the classical approach **multidimensional scaling (MDS)**:

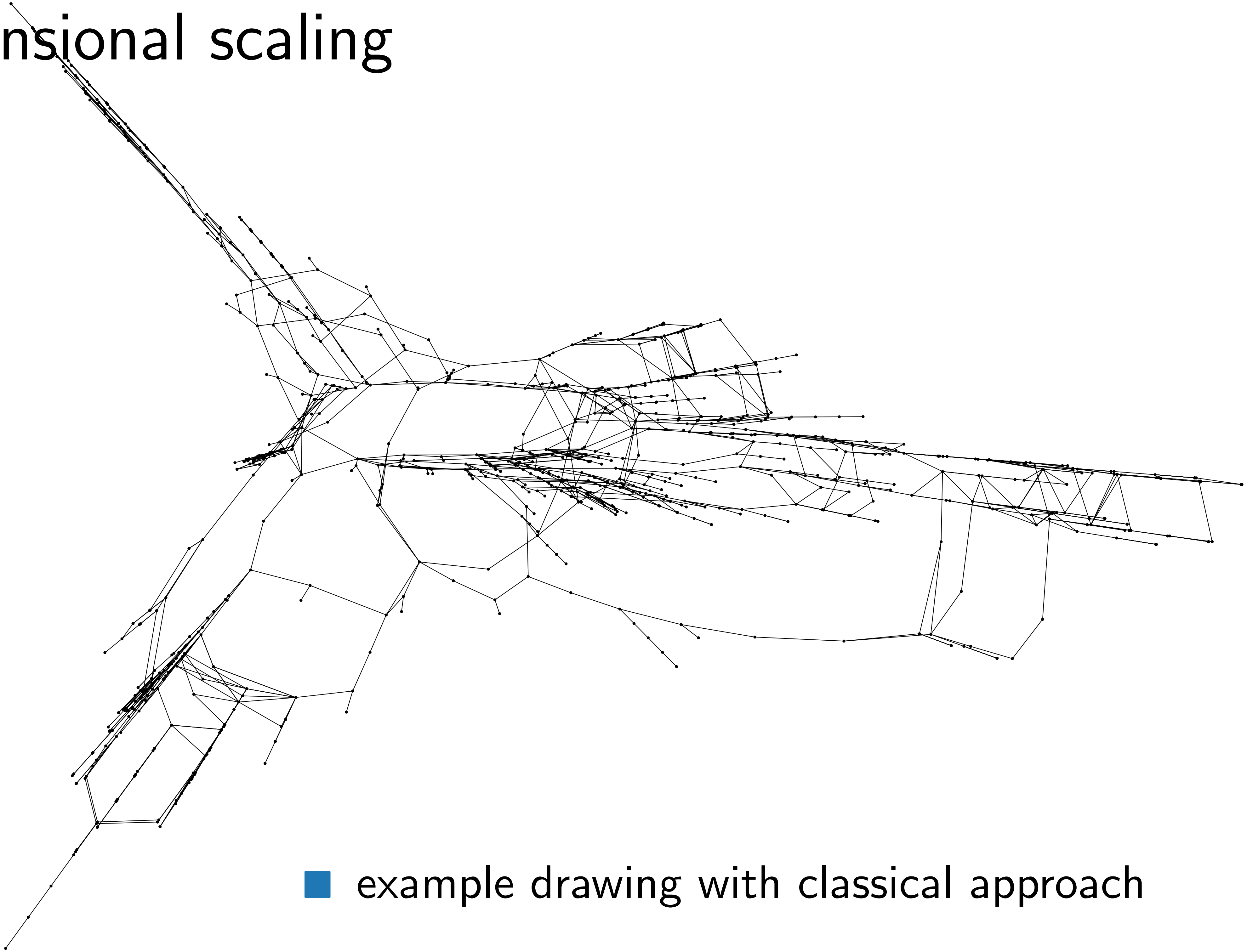
- MDS is a technique to visualise similarity among a set of objects
- Input is a distance matric D with $d_{ij} \sim$ dissimilarity between objects i and j
- We search for points $x_1, \ldots, x_n \in \mathbb{R}^2$ such that

$$||x_i - x_j|| \approx d_{ij}$$

For our drawing, how do we define the dissimilarity between two vertices?

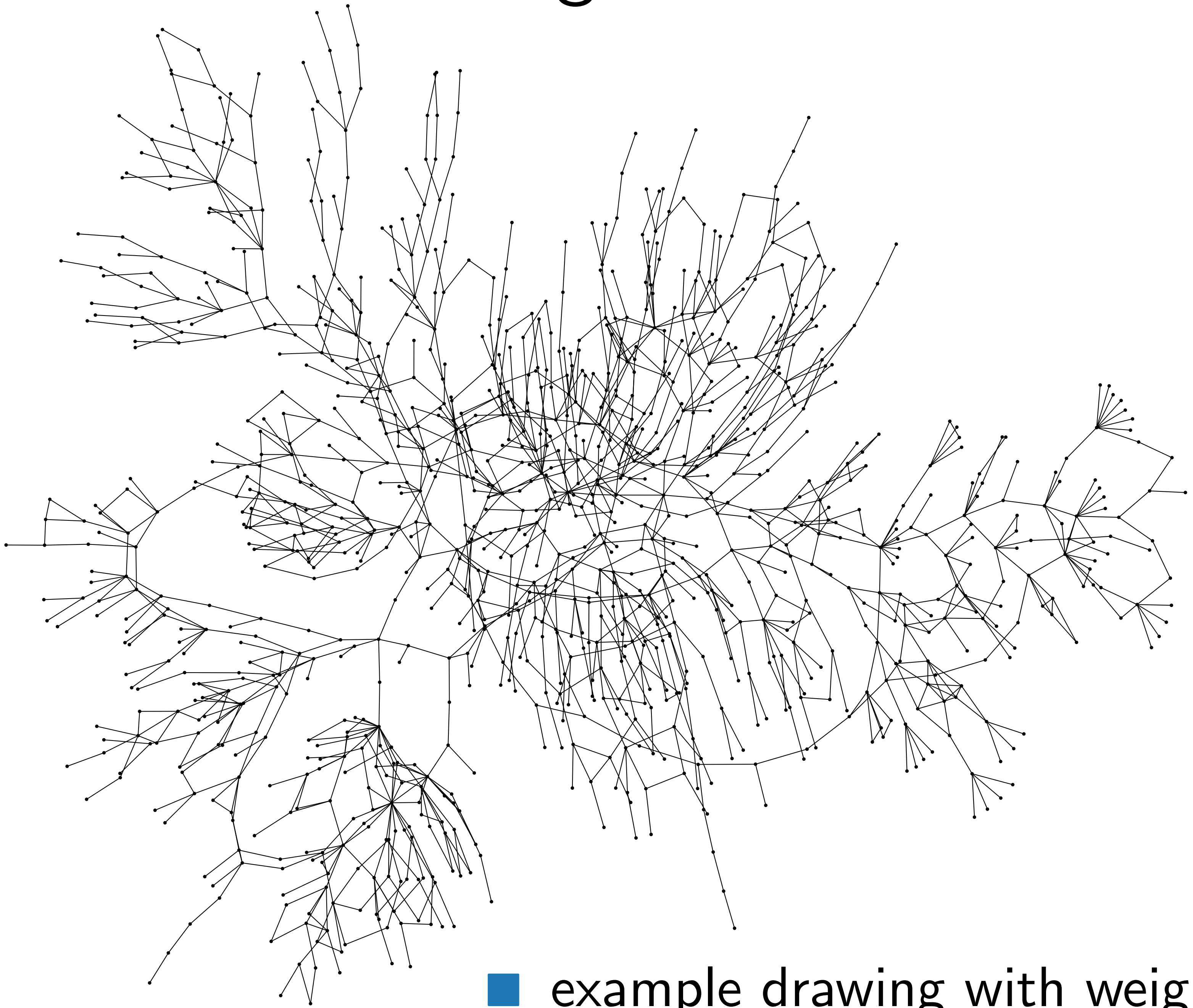
- Set d_{uv} as the distance of u and v in G in terms of a shortest path between them.

Multidimensional scaling



- example drawing with classical approach

Multidimensional scaling



- example drawing with weighted version

Literature

Main sources:

- [GD Ch. 10] Force-Directed Methods
- [DG Ch. 4] Drawing on Physical Analogies

Referenced papers:

- [Johnson 1982] The NP-completeness column: An ongoing guide
- [Eades, Wormald 1990] Fixed edge-length graph drawing is
- [Saxe 1980] Two papers on graph embedding problems NP-hard
- [Eades 1984] A heuristic for graph drawing
- [Fruchterman, Reingold 1991] Graph drawing by force-directed placement
- [Frick, Ludwig, Mehldau 1994] A fast adaptive layout algorithm for undirected graphs