Force-directed layouts

Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm

Matt Piekenbrock

Force-directed layout

Force directed (FD) layouts are motivated by the need for:

1. Even spacing between nodes
2. Minimal number of edge crossings
3. Uniform placement of nodes around the screen
4. Revealing symmetry in tangled graphs

The *force directed philosophy*:

- Use simple forces to implicitly handle issues (1-4)
- Empirical performance >> Analytical tractability

In terms of bounds, guarantees, etc. FD offers no advantages.

That said, in practice, the heuristic work quite well!

Spring Embedders

The first attempt at force-directed layout was the through the use of *spring embedders*

> The basic idea is as follows. To embed [lay out] 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.

Foundational thought: to layout a graph $G = (V, E)$, treat is as a *physical system*, i.e. subject it to certain forces and let it reach a "steady state"

Eades, Peter. "A heuristic for graph drawing." Congressus numerantium 42 (1984): 149-160.

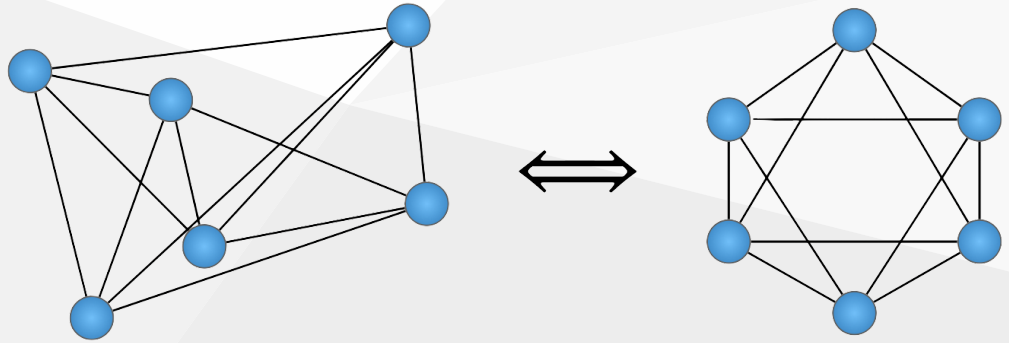
Force-directed layout

Fruchterman Reingold: Insight

(1) *Connected* vertices should be close

(2) ... but no vertices should be *too* close

> At a distance of about 1 fm [femto-meter] the strong nuclear force is attractive and about 10 times the electric force between two protons. The force decreases rapidly with increasing distance... when two nucleons are within about 0-4 fm of each other, the strong nuclear forces become repulsive.



Fruchterman, Thomas MJ, and Edward M. Reingold. "Graph drawing by force-directed placement." Software: Practice and experience 21.11 (1991): 1129-1164.

Fruchterman Reingold: Insight

What are we trying to achieve? Let $n = |V|$ and $d_{ij} = \|v_i - v_j\|$. Want an optimal layout P^* satisfying:

$$\underset{P \subset \mathbb{R}^{n \times 2}}{\arg\min} \sum_{i<j} c_{ij}(d_{ij} - l_{ij})^2$$

where l_{ij} is the "ideal distance" between v_i and v_j and c_{ij} is a scaling constant

- Note: This is actually solved by Multi-dimensional Scaling (MDS) if l_{ij} is known!

Fruchterman Reingold: Insight

Optimal distance between vertices ought to be:

$$k = C\sqrt{A/|V|}$$

where:

- $k \approx$ radius of empty area around a vertex
- A = area of screen
- C = constant scaling factor

How can we optimize the layout such that every vertex is at distance k from each other?

Fruchterman, Thomas MJ, and Edward M. Reingold. "Graph drawing by force-directed placement." Software: Practice and experience 21.11 (1991): 1129-1164.

Fruchterman Reingold: Insight

$$\text{Objective:} \quad \underset{P \subset \mathbb{R}^{n \times 2}}{\arg\min} \sum_{i<j} c_{ij}(d_{ij} - l_{ij})^2$$

Fruchterman's idea:

(1) Connected vertices should be close (*attraction*)

(2) ... but no vertices should be *too* close (*repulsion*)

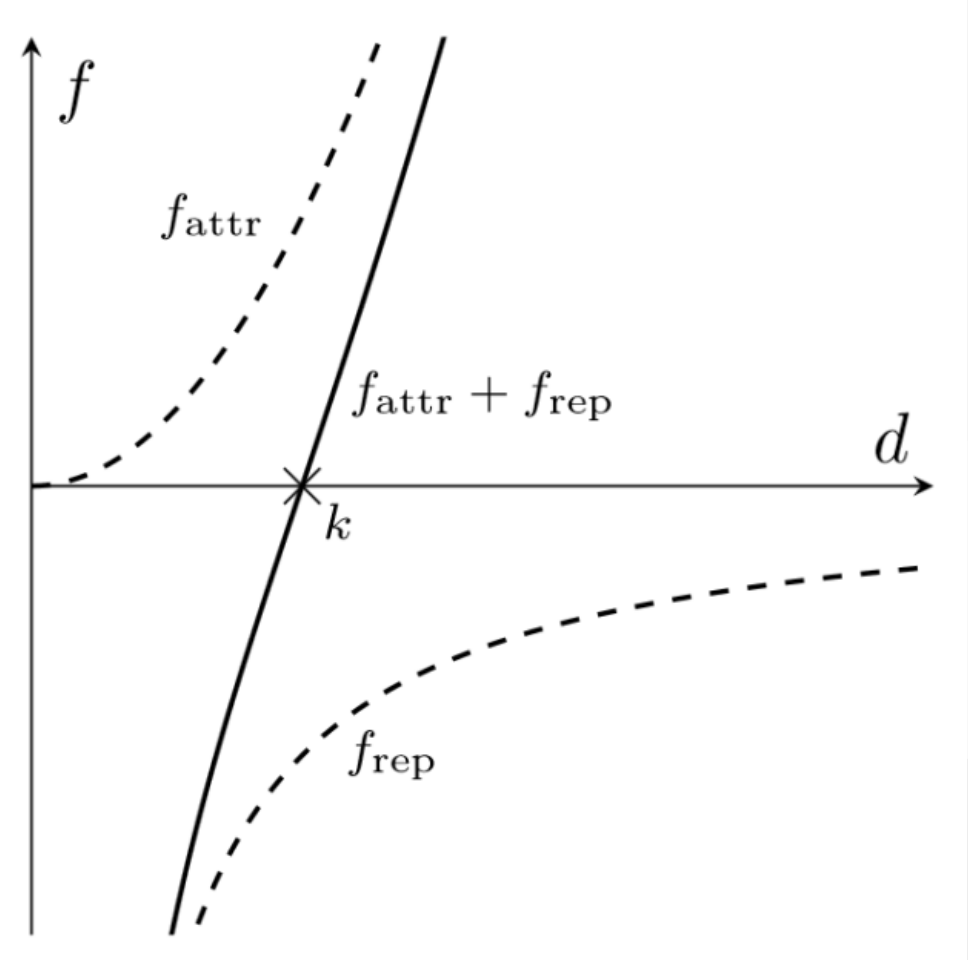
We can encapsulate these heuristics as functions:

$$f_a(d) = d^2/k$$

$$f_r(d) = -k^2/d$$

Fruchterman, Thomas MJ, and Edward M. Reingold. "Graph drawing by force-directed placement." Software: Practice and experience 21.11 (1991): 1129-1164.

$$\text{Objective} \approx \sum_{i<j} f_a(d_{ij}) + f_r(d_{ij})$$



From https://schneide.blog/tag/fruchterman-reingold/

Force directed issues

- Problem: If G is complex (or large), the objective $\hat{f}$ may yield *many* local minima
- Solution: Throw computation at it!
  - Idea => iteratively adjust positions over time (example)

```
procedure Layout(G = (V, E))
  for i = 1 to n do
    for each v in V do
      disp(v) = 0
      for each u in V do
        if u ≠ v then
          Δ = pos(v) − pos(u)
          disp(v) = disp(v) + Δ/|Δ| · f_rep(u, v)
    for each (u, v) in E do
      Δ = pos(v) − pos(u)
      disp(v) = disp(v) − Δ/|Δ| · f_attr(u, v)
      disp(u) = disp(u) + Δ/|Δ| · f_attr(u, v)
    for each v in V do
      pos(v) = pos(v) + disp(v)/|disp(v)| · |disp(v)|
```

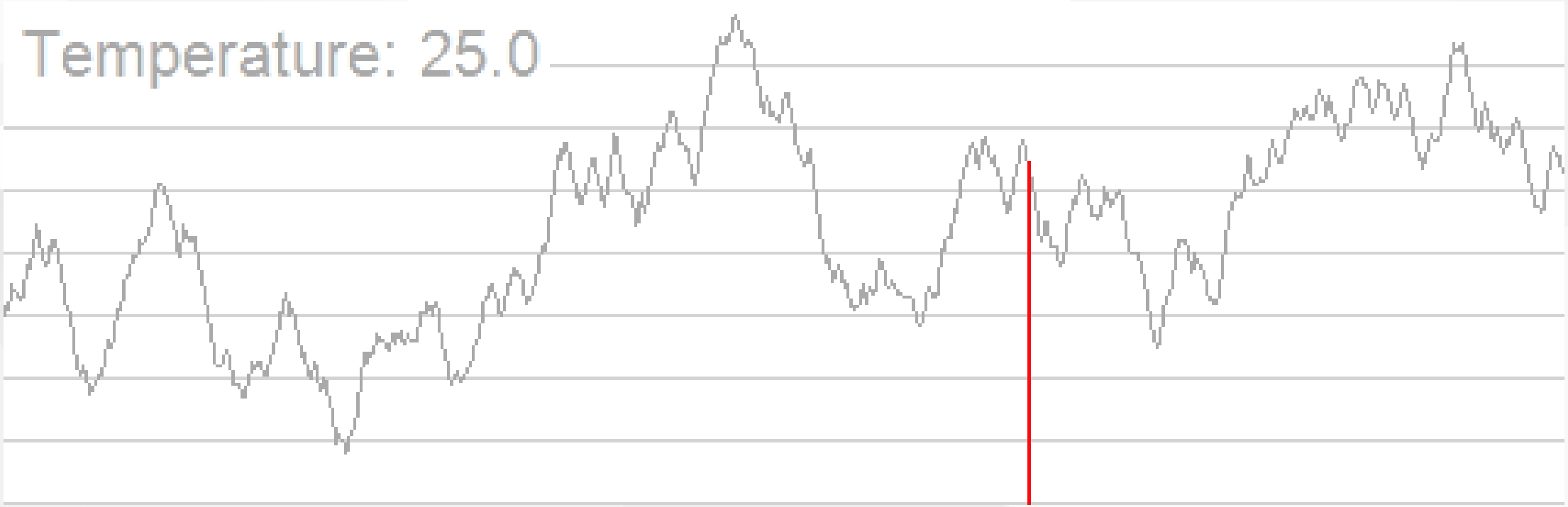
Does anyone see anything wrong with this?

From https://schneide.blog/tag/fruchterman-reingold/

Force directed issues

- Problem #1: If G is complex (or large), the objective $\hat{f}$ may yield *many* local minima
- Solution #1: Throw computation at it!
  - Idea => iteratively adjust positions *over time*
  - Problem #2: minima may never actually be reached!
- Solution #2: Add *temperature* and *cooling* to the system (treat it as as physical system!)

Simulated Annealing



From https://en.wikipedia.org/wiki/Simulated_annealing/

```
procedure Layout(G = (V, E))
  initialize temperature t
  for i = 1 to n do
    for each v in V do
      disp(v) = 0
      for each u in V do
        if u ≠ v then
          Δ = pos(v) − pos(u)
          disp(v) = disp(v) + Δ/|Δ| · f_rep(u, v)
    for each (u, v) in E do
      Δ = pos(v) − pos(u)
      disp(v) = disp(v) − Δ/|Δ| · f_attr(u, v)
      disp(u) = disp(u) + Δ/|Δ| · f_attr(u, v)
    for each v in V do
      pos(v) = pos(v) + disp(v)/|disp(v)| · min(|disp(v)|, t)
    t = cool(t)
```

From https://schneide.blog/tag/fruchterman-reingold/

Reminder to stop and show iterations of FR

Force directed issues

- Problem #1: If G is complex (or large), the objective $\hat{f}$ may yield *many* local minima
- Solution #1: Throw computation at it!
  - Idea => iteratively adjust positions *over time*
  - Problem #2: minima may never actually be reached!
- Solution #2: Add *temperature* to the system
  - Treat it as as physical system!
  - Problem #3: This is **slow!** ($O(|V|^2)$ *per iteration*)

Opinion: It's not immediately clear how to fix Problem #3

Detour: Solar Systems

Suppose we interpret G as a *galaxy* (set of *solar systems*)

Recall: a *partition* of a set $\mathcal{X}$ is a set of non-empty subsets $\{X_1, \ldots, X_k\}$ such that:

$$\mathcal{X} = \bigcup_{i=1}^{k} X_k, \quad X_i \neq X_j \implies X_i \cap X_j = \emptyset$$

Partition V such that each $v \in V$ is a *sun*, a *planet*, or a *moon*

- Every planet node is the neighbor of some sun node
- Every moon node is the neighbor of some planet node
- Every sun node is part of a *solar system*

Let's make this a bit more exact...

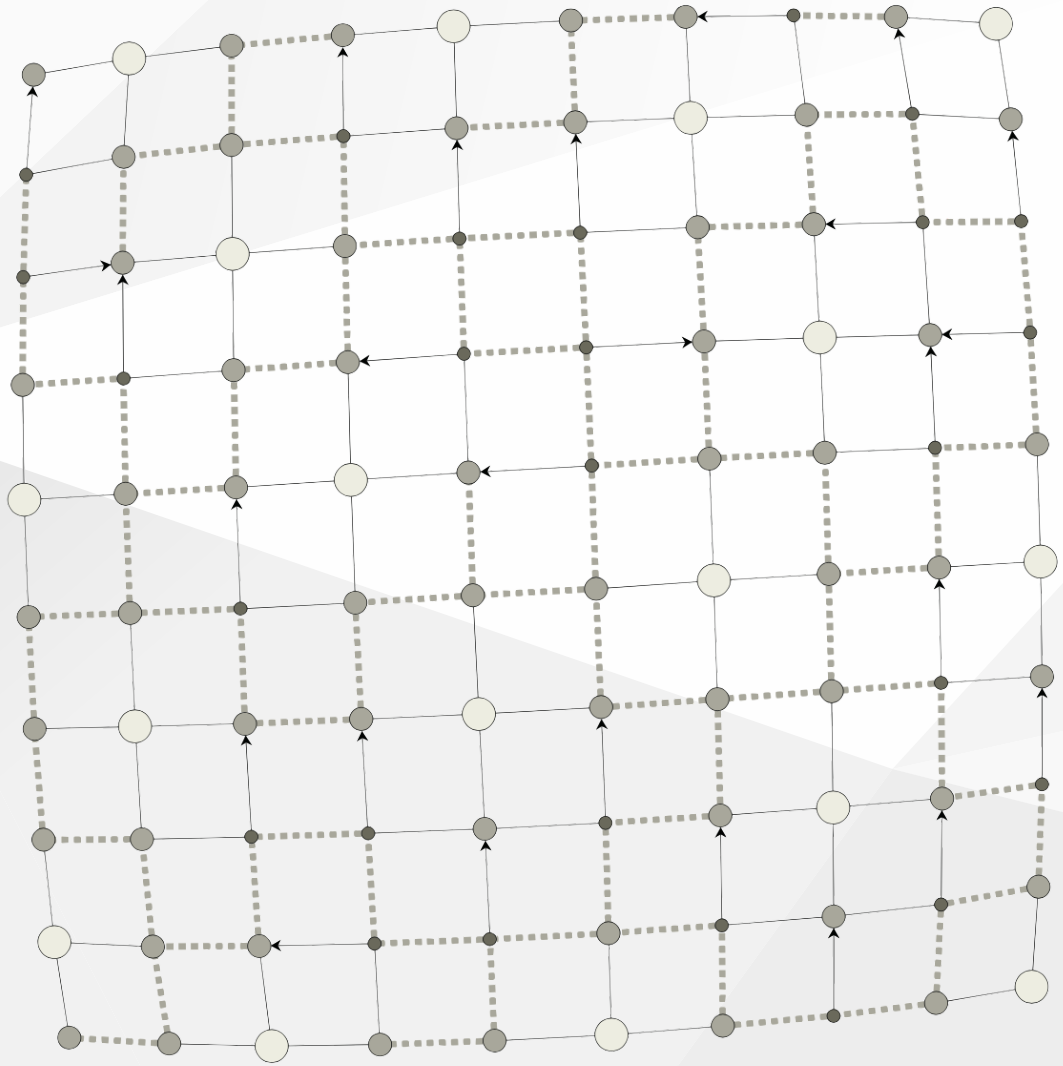
Detour: Solar Systems

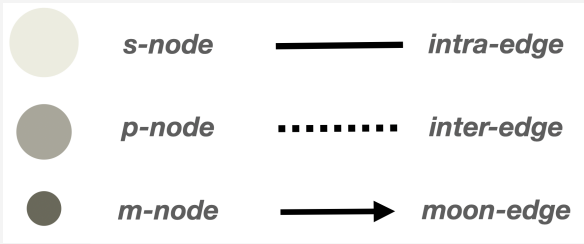
Partition $G = (V, E)$ into solar systems $\mathcal{S}$:

$$\mathcal{S} = \{S^0, S^1, \ldots, S^k\}$$

such that:

- Each $v \in V$ is a (s, p, m)-node (sun, planet, moon)
- Each $e \in E$ is a $(\phi, \psi, \rightarrow)$-edge (intra, inter, directed)
- $S^i \cap S^j = \emptyset \quad \forall i, j \in [n]$ (partition)
- $S^i := G[U]$ is an induced subgraph of s_i where each $u \in U \subseteq V$ satisfies $d_G(s_i, u) \leq 2$ (solar system)




s-node
intra-edge
p-node
inter-edge
m-node
moon-edge

Observations

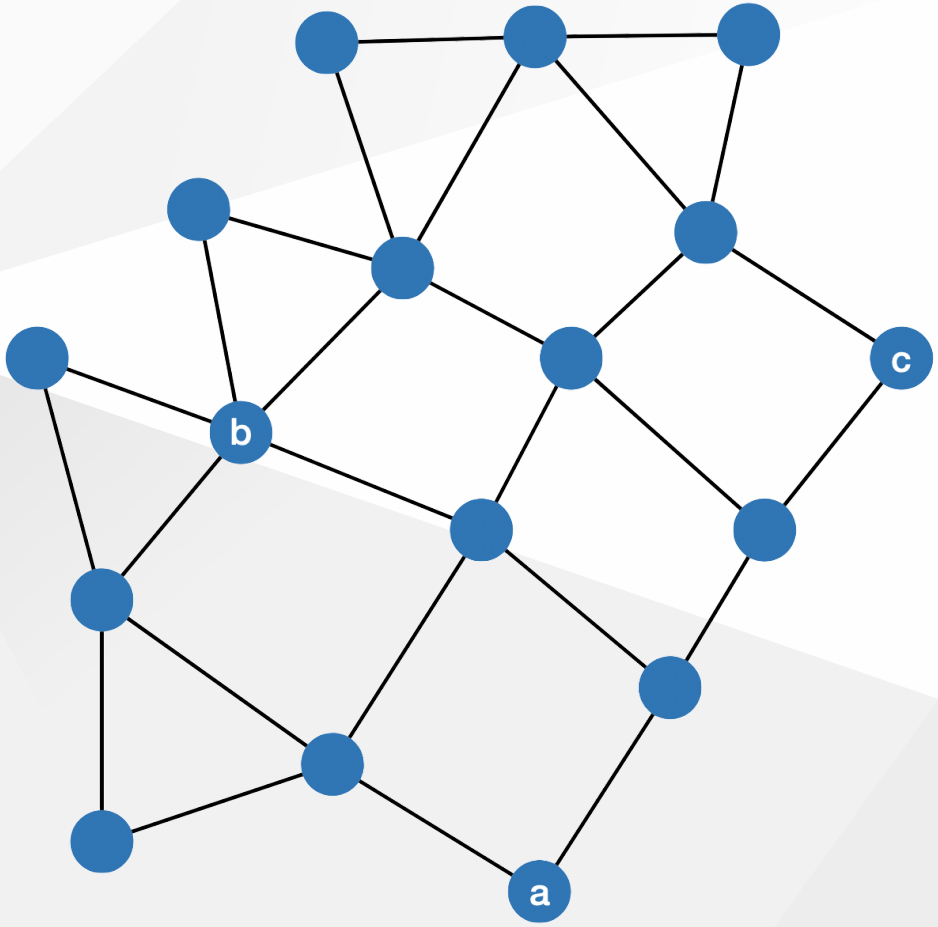
If $G = (V, E)$ is partitioned by $\mathcal{S}$ with $|\mathcal{S}| = k$, then:

- Each s-node is in exactly one solar system
- Each m-node has exactly one p-node as a neighbor
- A partitioning $\mathcal{S}$ can be obtained in $O(|V| + |E|)$ time

Greedy algorithm:

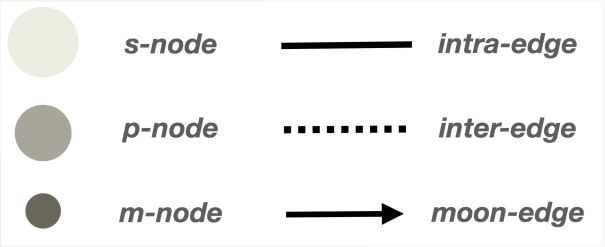
1. Set $V' = V$
2. Choose a s-node $s_i \in V'$ arbitrarily
3. $V' = V' \setminus S^i$ (S^i induced by s_i)
4. Repeat 2-3 until $V' = \emptyset$

Activity

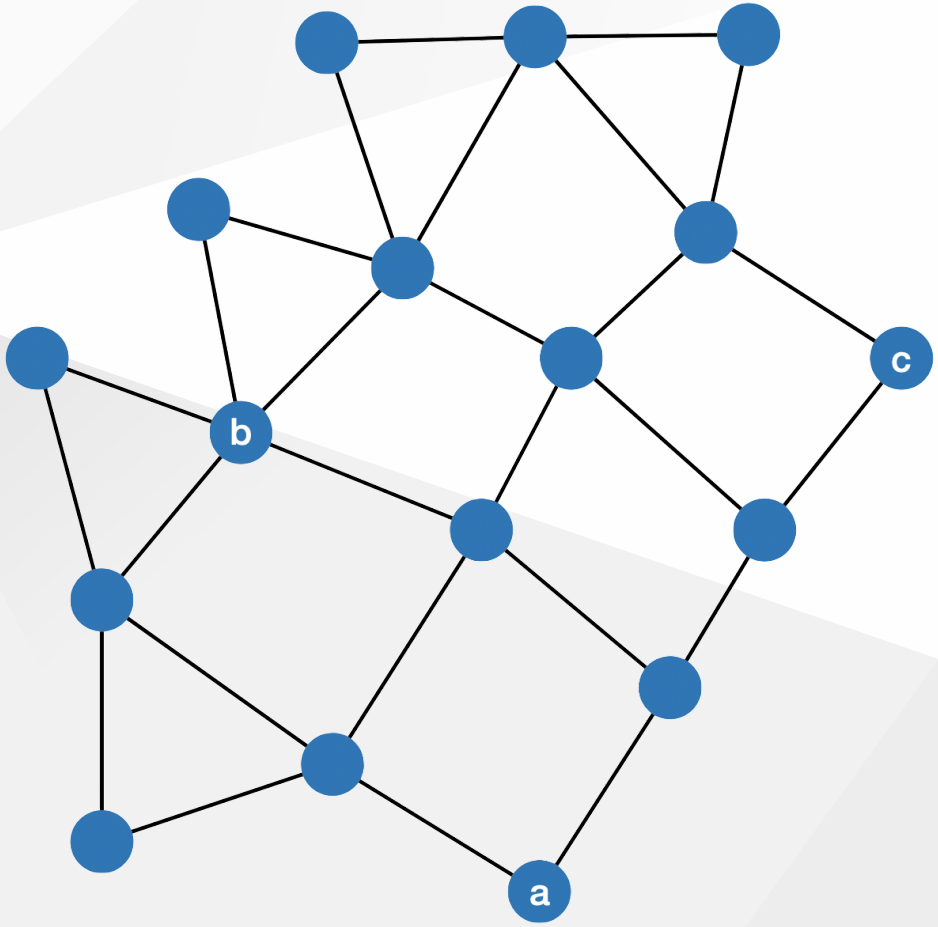


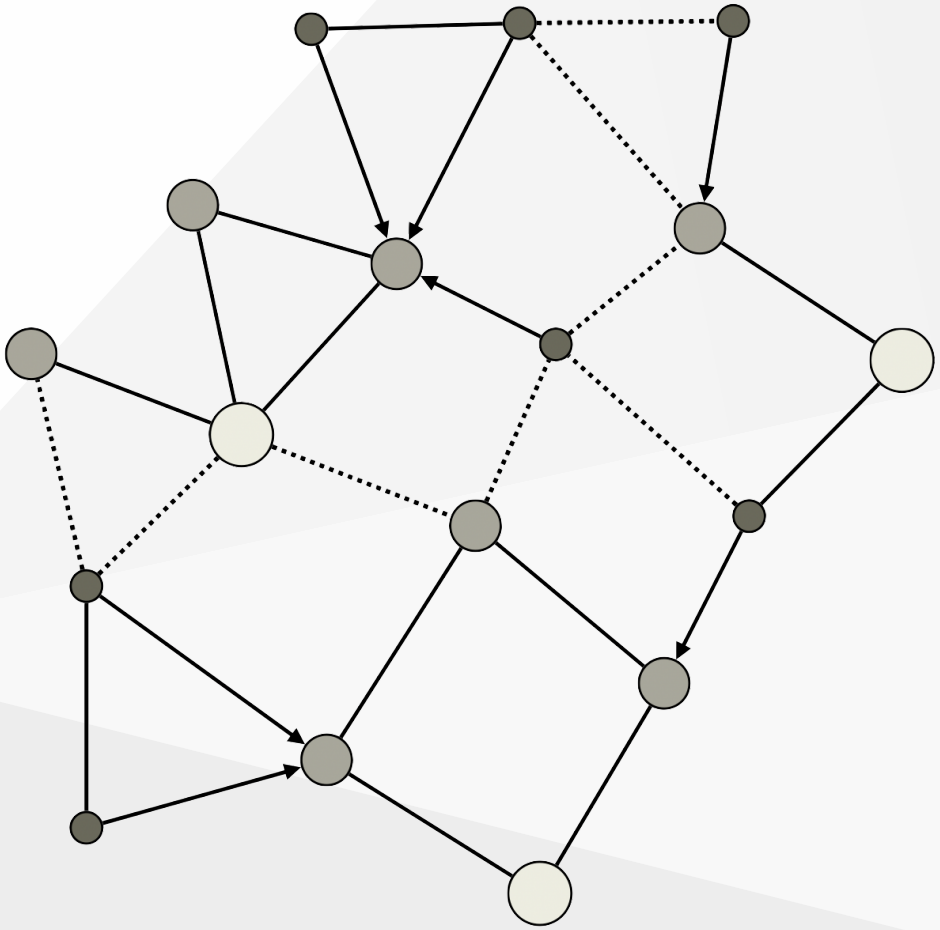
Greedy algorithm:

1. Set $V' = V$
2. Choose $s_i \in V'$ arbitrarily
3. $V' = V' \setminus S^i$
4. Repeat 2-3 until $V' = \emptyset$



Optional: Choose sun nodes in order a, b, c


c
b
a



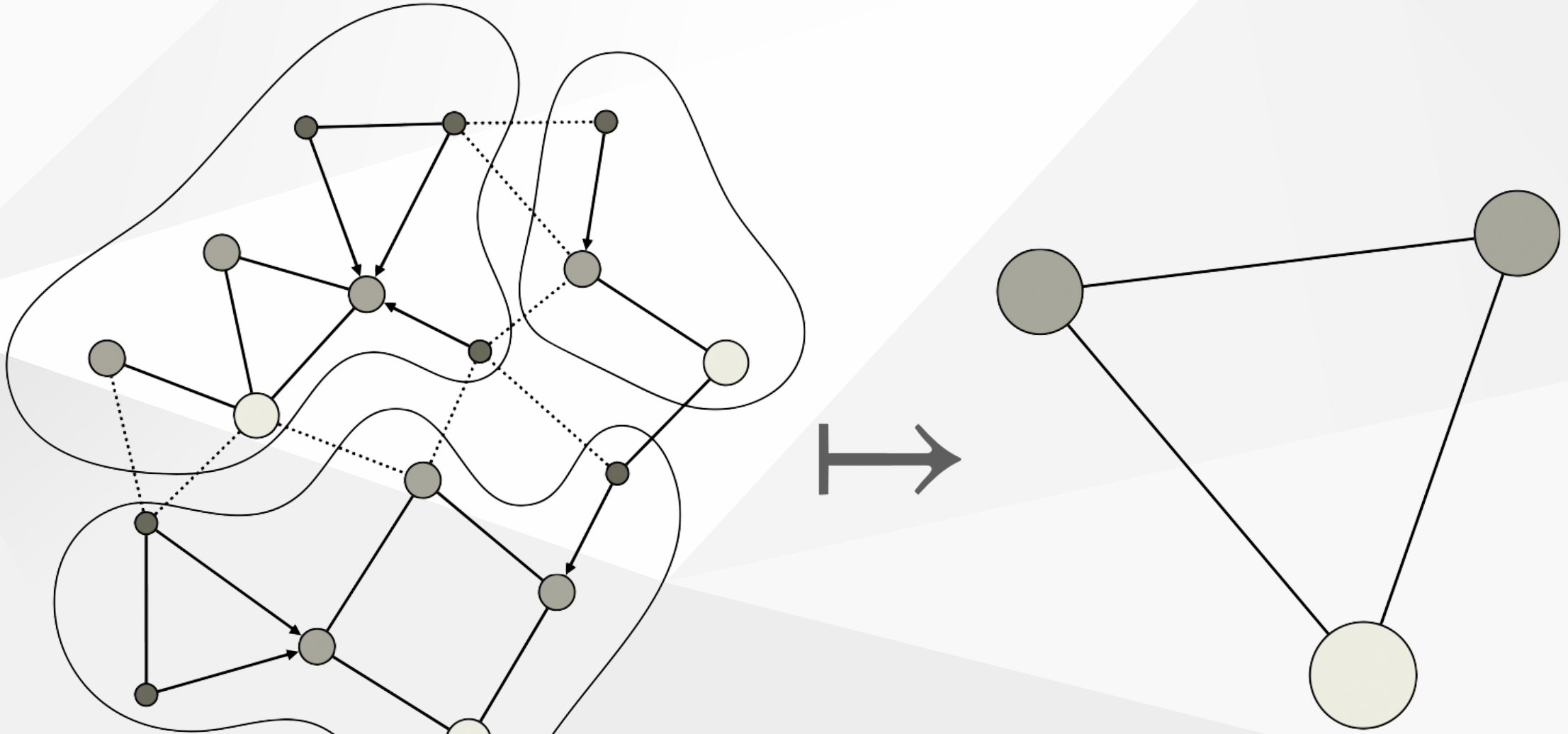
Coarsening

- Suppose each solar system S^i is *collapsed* to a vertex positioned its corresponding barycenter
- Similarly: collapse inter-system edges weighted with some heuristic (e.g. average edge weights)
- Observe this takes $O(|V| + |E|)$ time

Inductively: Let $G = G_0$, $G_i =$ *coarsened* graph of G_{i-1}, such that we have a *multilevel representation* of G:

$$G_0 \hookleftarrow G_1 \hookleftarrow \cdots \hookleftarrow G_k$$

such that $X \hookleftarrow Y$ indicates that Y is a coarsening of X

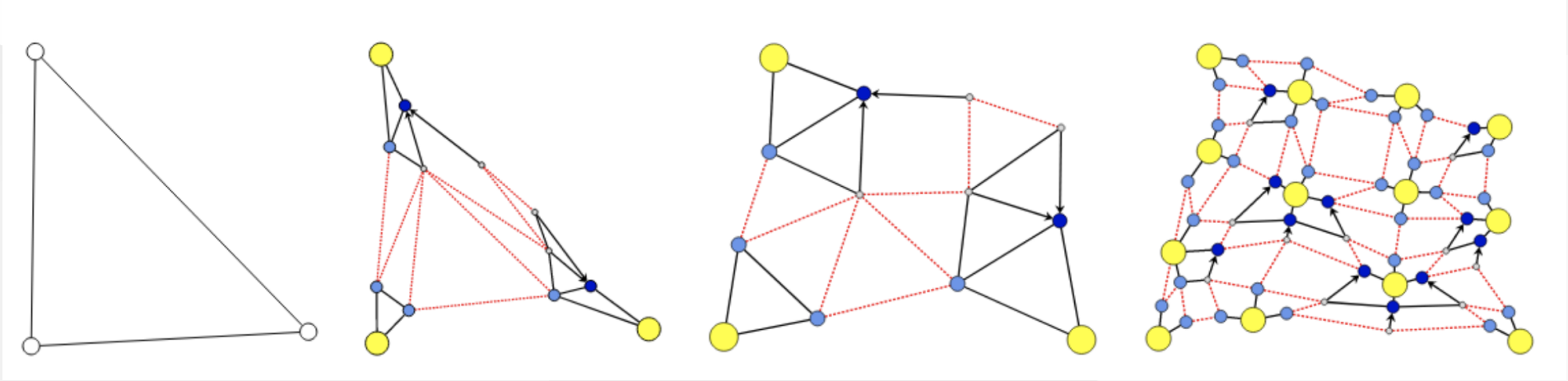


Analysis Part #1

- Every node v_i belongs to exactly one solar system
- Every solar system contains at least 2 nodes

  $\Longrightarrow$ G_{i+1} contains *at most* $\frac{1}{2}|V_i|$ nodes

  $\Longrightarrow$ there are *at most* $k \leq \log(|V|)$ collapsed graphs



Picture from (1) showing 3 coarsenings

1. Hachul, Stefan, and Michael Jünger. "Large-graph layout with the fast multipole multilevel method." Spring, V (December) (2005)

a Multi-level Layout Algorithm

End Detour: *Why did we just do all of that?*

Recall we want to accelerate:

$$\sum_{i<j} f_a(d_{ij}) + f_r(d_{ij})$$

The multi-layer representation motivates the following:

1. Create $G_0 \hookleftarrow G_1 \hookleftarrow \cdots \hookleftarrow G_k$
2. Apply forces to G_k
3. Propagate information from (2) to G_{k-1}
4. Repeat steps 2-3 until reaching the original G_0

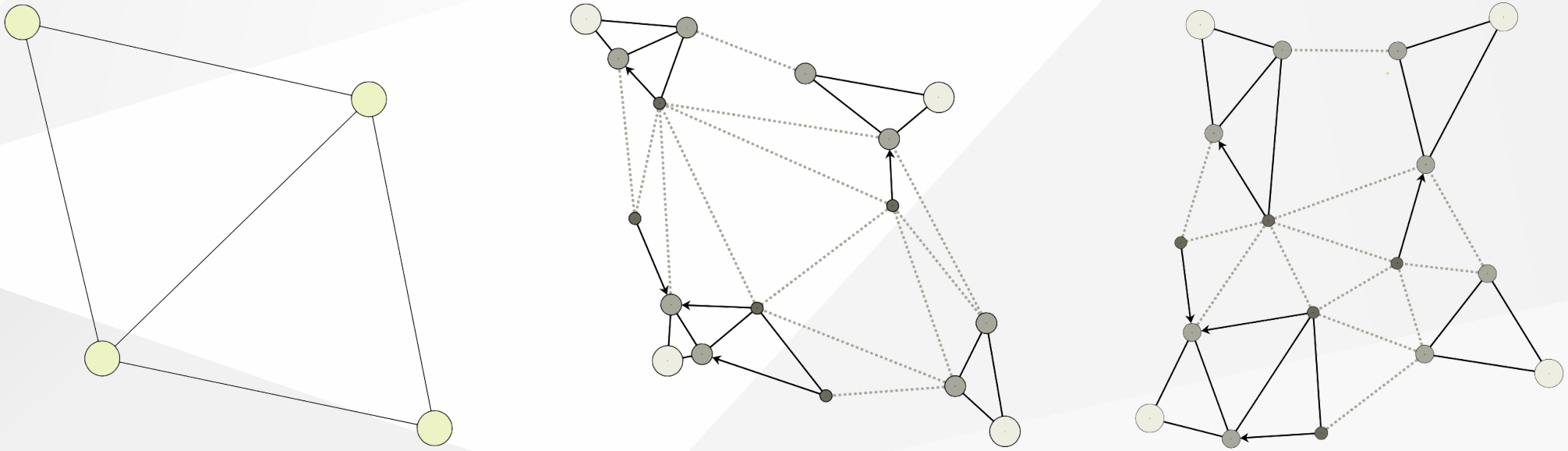


Fig. 2. (a) Drawing of G_2. (b) Initial placement of G_1. (c) Drawing of G_1.

1. Hachul, Stefan, and Michael Jünger. "Large-graph layout with the fast multipole multilevel method." Spring, V (December) (2005)

Analysis Part #2

Complexity of multi-layer layout:

$$T_{\text{multi}} = \sum_{i=0}^{k} \left(T_{\text{partition}}(G_i) + T_{\text{coarsen}}(G_i) + T_{\text{force}}(G_i) \right)$$

Note we've shown:

1. $\sum_{i=0}^{k} T_{\text{partition}}(G_i) \sim O(|V| + |E|)$
2. $\sum_{i=0}^{k} T_{\text{coarsening}}(G_i) \sim O(|V| + |E|)$

$$\implies T_{\text{multi}} \cong T_{\text{force}}$$

Analysis Part #2: continued

Assume $|E_{i+1}| \leq \frac{1}{2}|E_i|$ for all $i \in \{0, 1, \ldots, k\}$. Then:

$$\sum_{i=0}^{k} T_{\text{force}}(G_i) = \sum_{i=0}^{k} T(|V_i|, |E_i|)$$

$$\leq \sum_{i=0}^{k} T_{\text{force}}\left(\frac{|V|}{2^i}, \frac{|E|}{2^i}\right) \quad \text{(because } \frac{1}{2}|V_i| \leq |V_{i-1}|\text{)}$$

$$\leq T_{\text{force}}(|V|, |E|) \sum_{i=0}^{k} \frac{1}{2^i} \quad \text{(for large } |V| \text{ and } |E|\text{)}$$

$$\leq 2\, T_{\text{force}}(|V|, |E|) \quad \text{(by geometric series)}$$

$$\implies T_{\text{multi}} \sim \Omega(|V| + |E|)$$

Can we do better?

$O(|V| + |E|)$ is much better than $O(|V|^2)$ if G is sparse

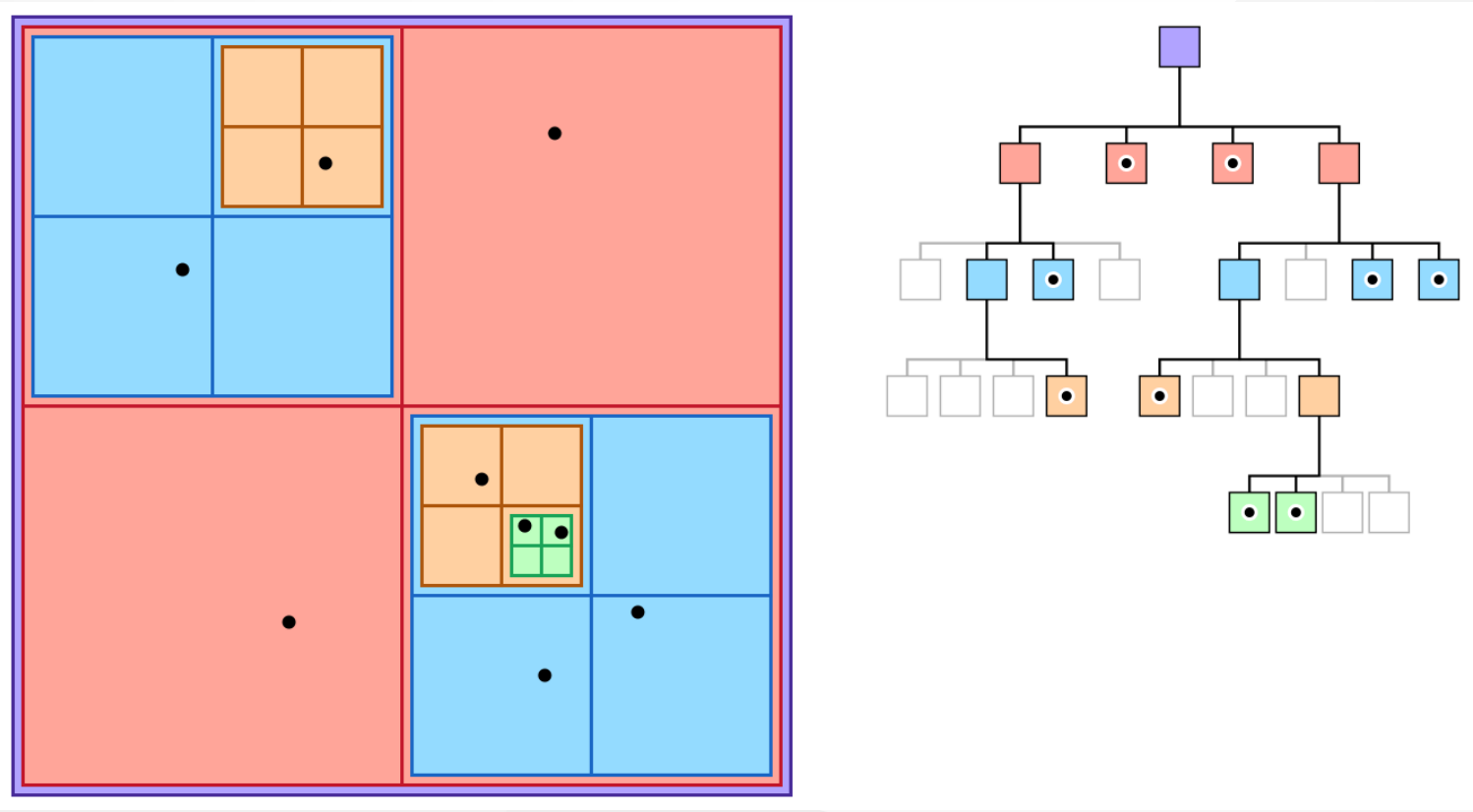
Can be improved to $O(|V| \log |V|)$ (!) if the *fast multipole method* (FMM) is employed

Listed by SIAM editors in top 10 algorithms of previous century[1]

FMM Perspective:
- solar system decomposition is just one type of decomposition
- there are plenty of others (e.g. quadtrees)!

1. https://archive.siam.org/news/news.php?id=637

What are Quad Trees?

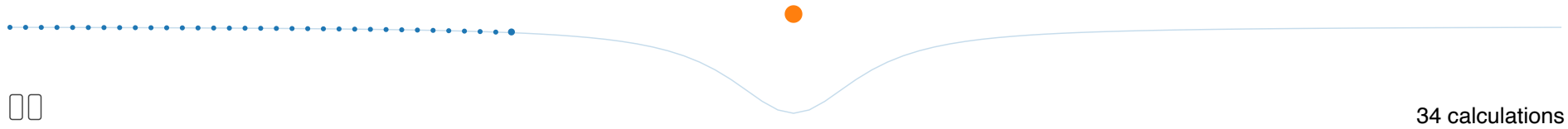


Quad tree $\Leftrightarrow$ hierarchical decomposition of points in $\mathbb{R}^2$ where each node has fixed degree 4.

From: https://developer.apple.com/documentation/gameplaykit/gkquadtree

What is the Fast Multipole Method?

What the field represents isn't so important for our purposes, because whatever it represents the way to calculate it is the same. To calculate the field, we look at each of a 100 equally-spaced points in turn and calculate the strength of the source's field at each one:

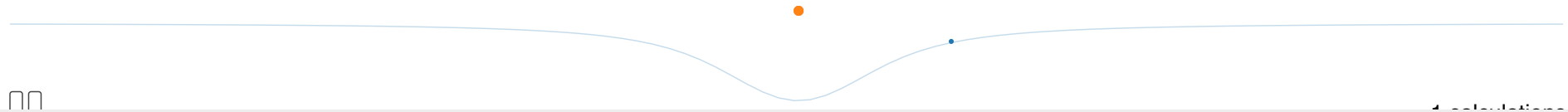


There are 100 points, so that took 100 calculations.

If there's more than one source, then their fields add together. To calculate the combined field at a point, the field emitted by each source needs to be taken into account. With 100 sources and 100 points, there are 100 × 100 = 10,000 calculations to do:

Groups

That's pretty wasteful though. If we've got a bunch of sources close to eachother like we do below, then they all make about the same contribution to the point. We could save a lot of work by using an approximation: move all the sources to the same spot, do *one* calculation from that spot, and then multiply the calculated field strength by the number of sources in the group:



Reminder to stop and show FMM benchmark

Thank you