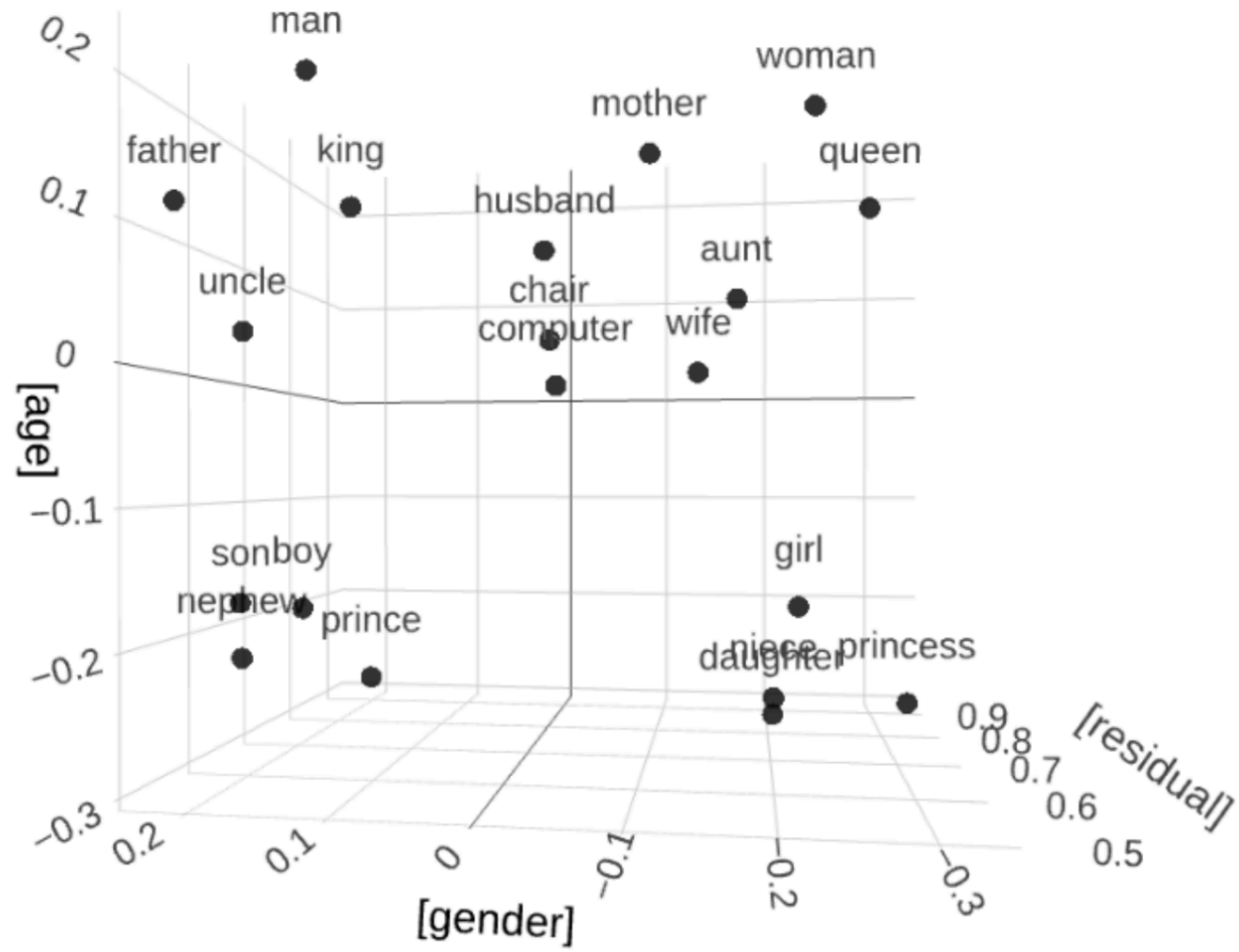
Nearest Neighbour Search: Methods and Theoretical Foundations

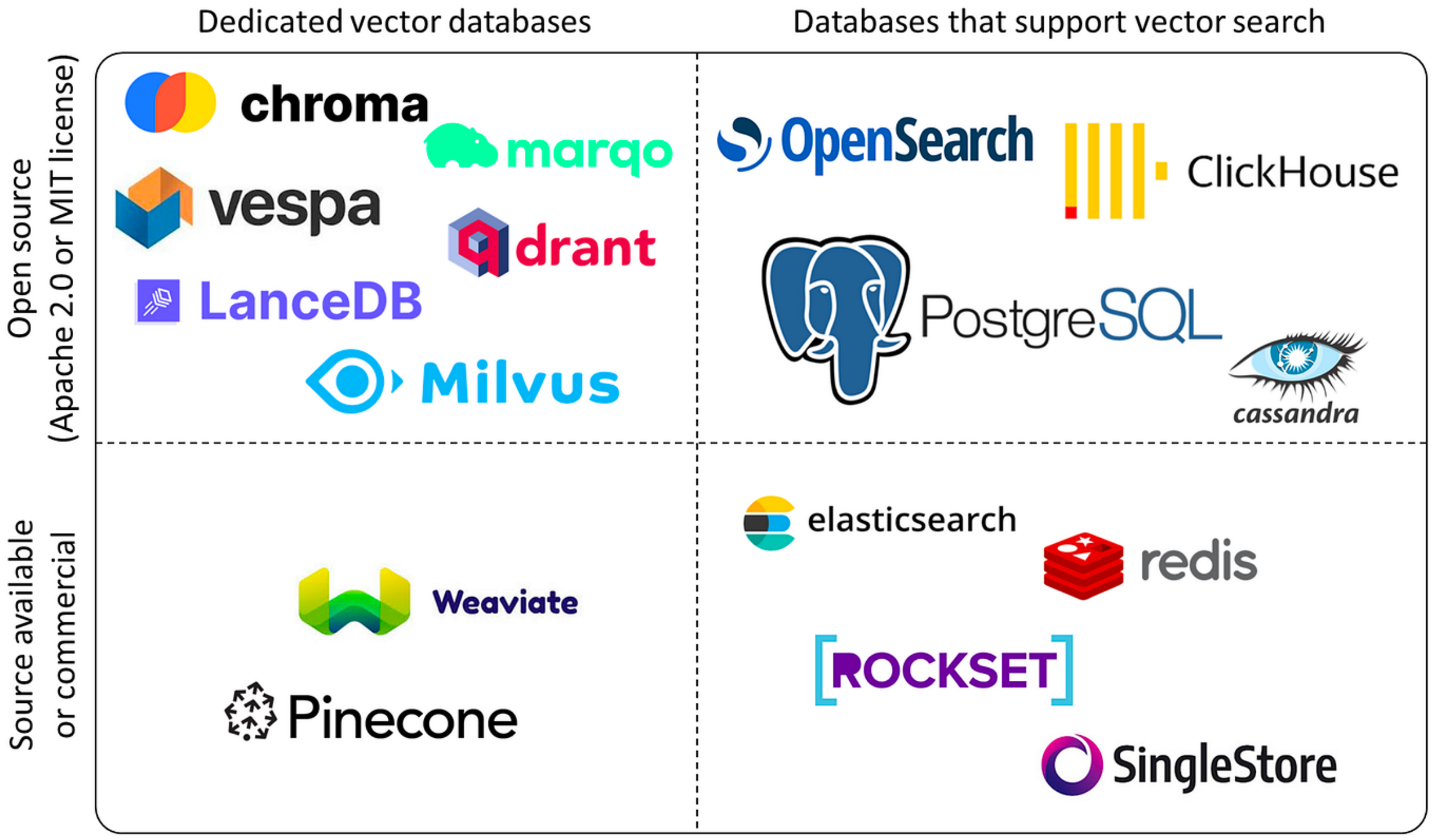
Alexander Ponomarenko
Laboratory of Algorithms and Technologies for Network Analysis, HSE

Nizhny Novgorod
2025

Applications

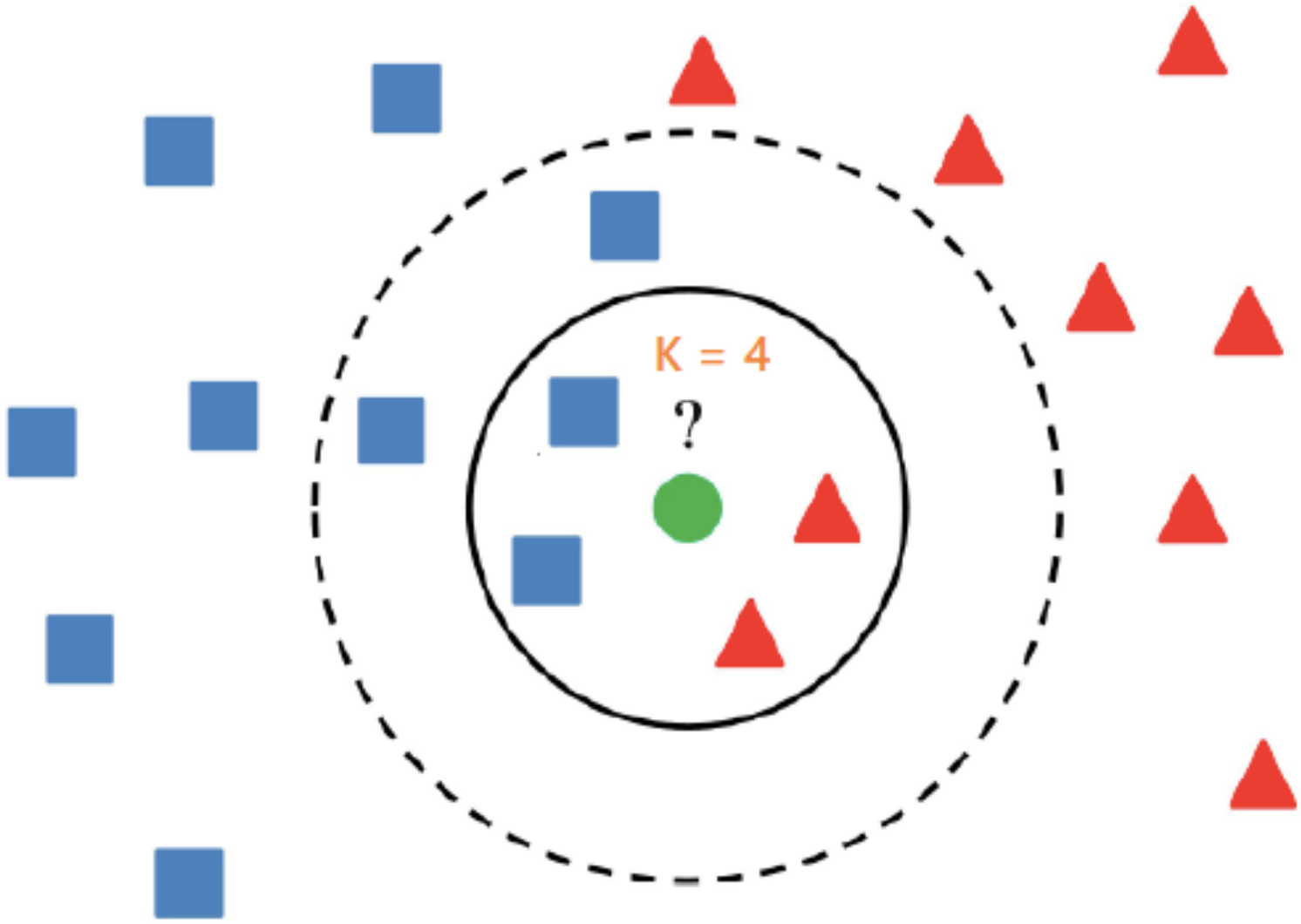
- распознание паттернов и классификация
- системы поиска по содержимому
- рекомендательные системы
- генерация дополненная поиском (RAG)



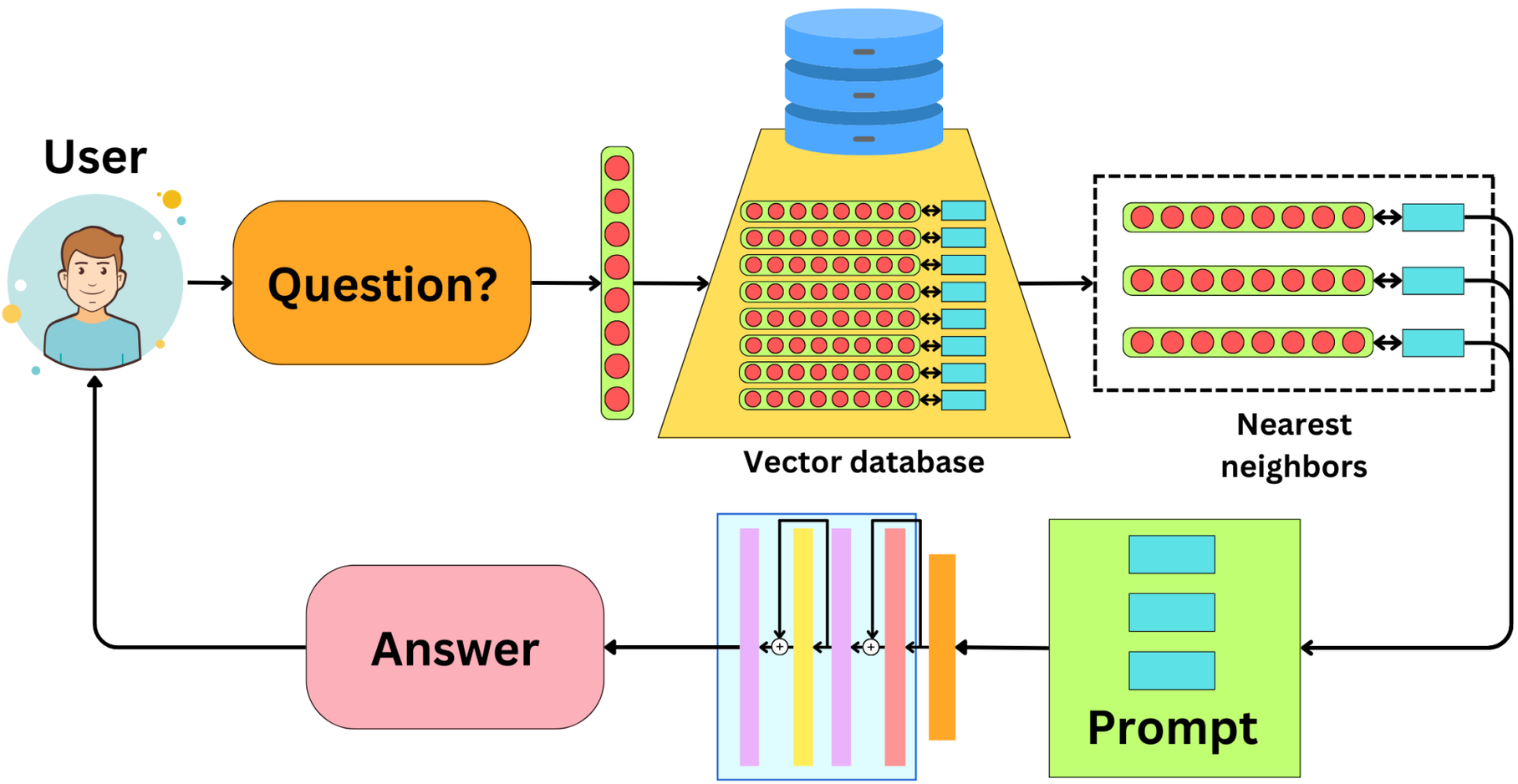


https://www.datacamp.com/blog/the-top-5-vector-databases

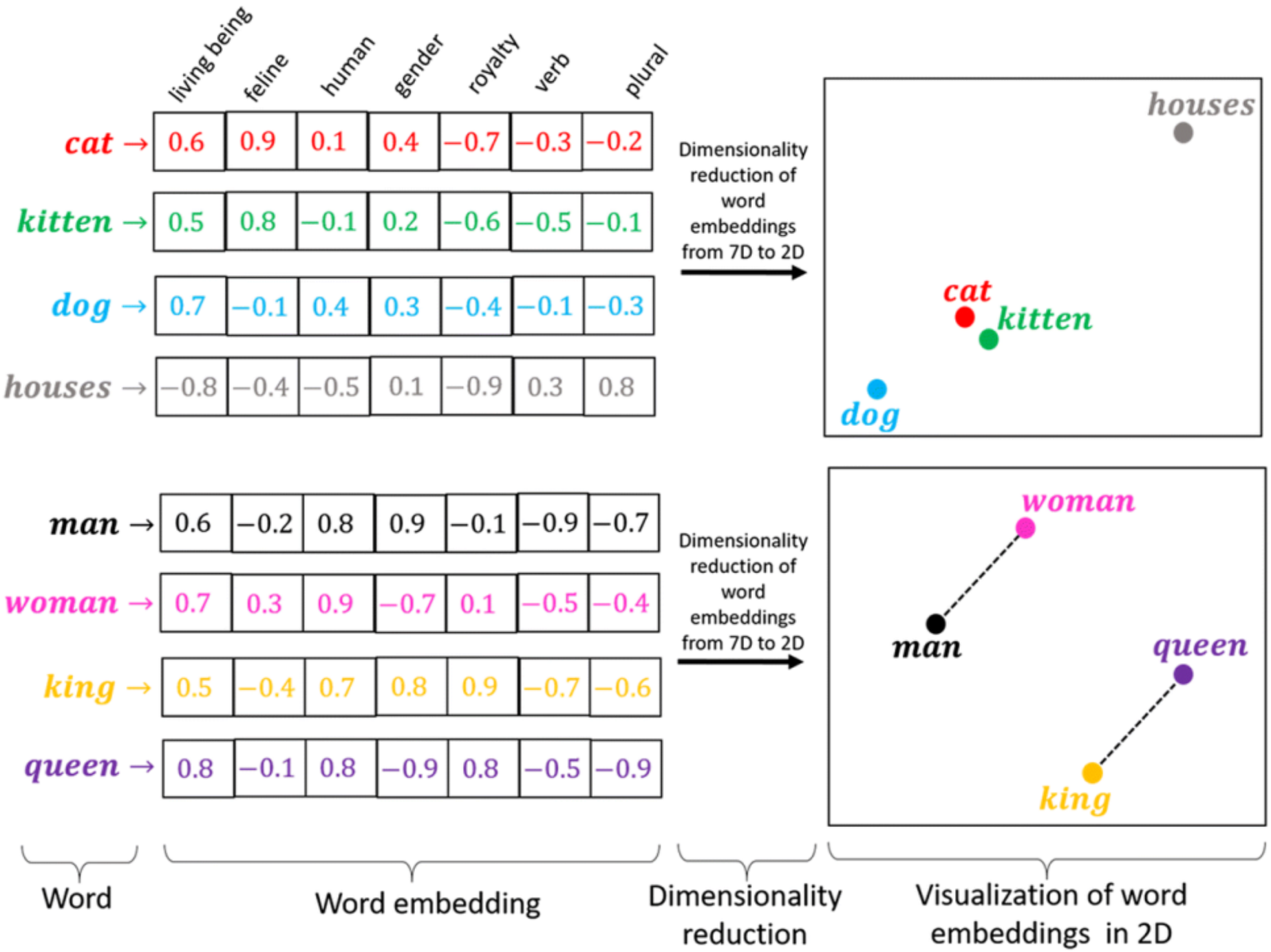
K-Nearest Neighbour Classifier


K = 4
?

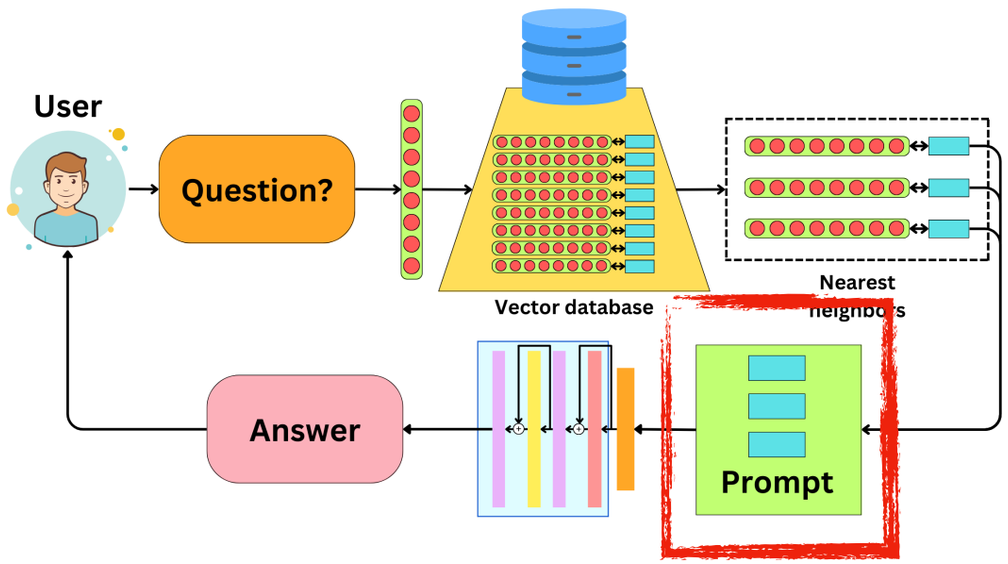
Retrieval Augmented Generation (RAG)



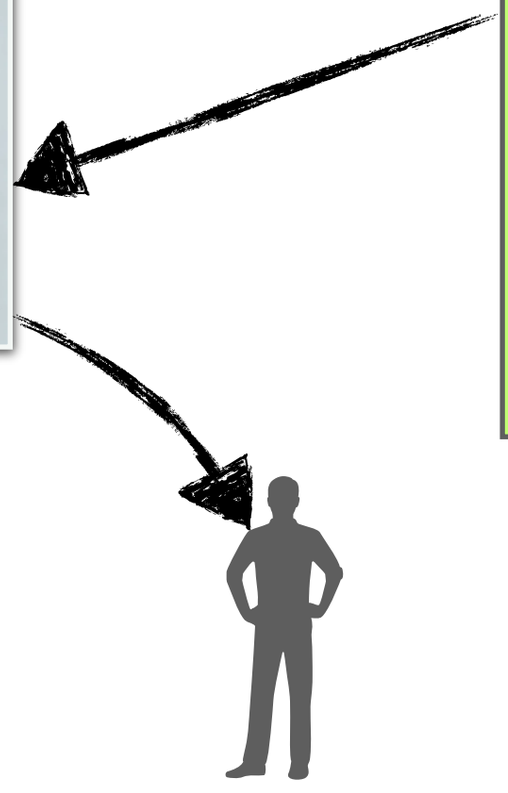
[https://newsletter.theaiedge.io/p/how-to-optimize-your-rag-pipelines]



https://www.linkedin.com/pulse/optimize-llm-enterprise-applications-through-embeddings-chunking/



Large Language Model (LLM)



Answer the question: "**How to return a travel ticket?**"
Consider this information:

1. Refund of one seat out of two issued in a compartment of a luxury class carriage (business) on international (far abroad) is not made. Refund of the cost of unused electronic tickets issued in a luxury class carriage (business) is made only in case of simultaneous refund of electronic tickets issued in one compartment.

2. How to return an electronic ticket on the website? Go to the "My orders" section in your personal account. Click "Request ticket status". Click "Make a refund".

3. …

4. …

Example of a prompt

Problem Statement

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix}_{n \times d}$$

$$x_i \in \mathbb{R}^d$$

$$X \subset \mathbb{R}^d$$

Dataset of n objects (d-dimensional vectors (points))

$\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \geq 0$ is a distance function

$$\rho(\cdot , \cdot)$$

For give $q \in \mathbb{R}^d$ need to calculate

$$x^* = \arg\min_{x_i \in X} \rho(x_i, q)$$

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix}_{n \times d}$$

$$x_i \in \mathbb{R}^d$$

$$X \subset \mathbb{R}^d$$

Dataset of n objects (d-dimensional vectors (points))

$\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \geq 0$ is a distance function $\rho(\cdot, \cdot)$

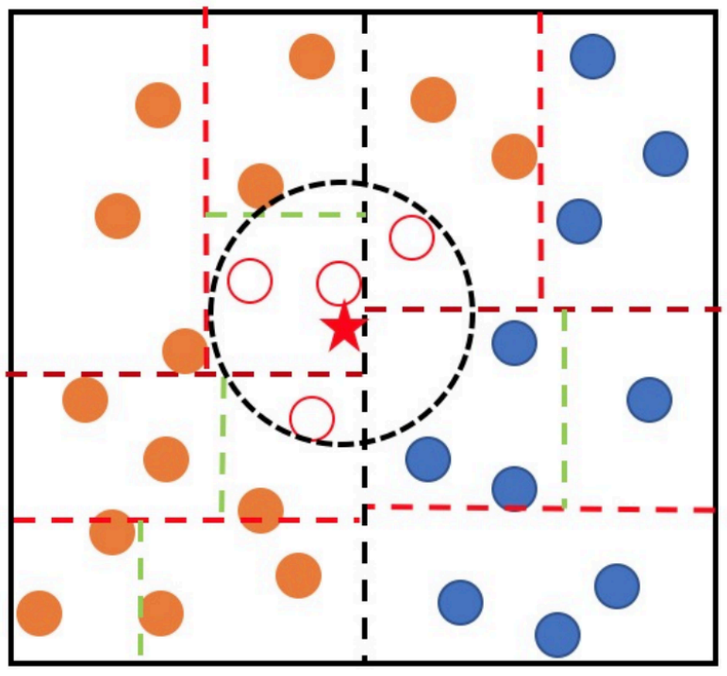
Need to build a structure S over X, so for any given $q \in \mathbb{R}$ operation $x^* = \arg\min_{x_i \in X} \rho(x_i, q)$ can be calculated as fast as it possible.

ε-Nearest Neighbour Search

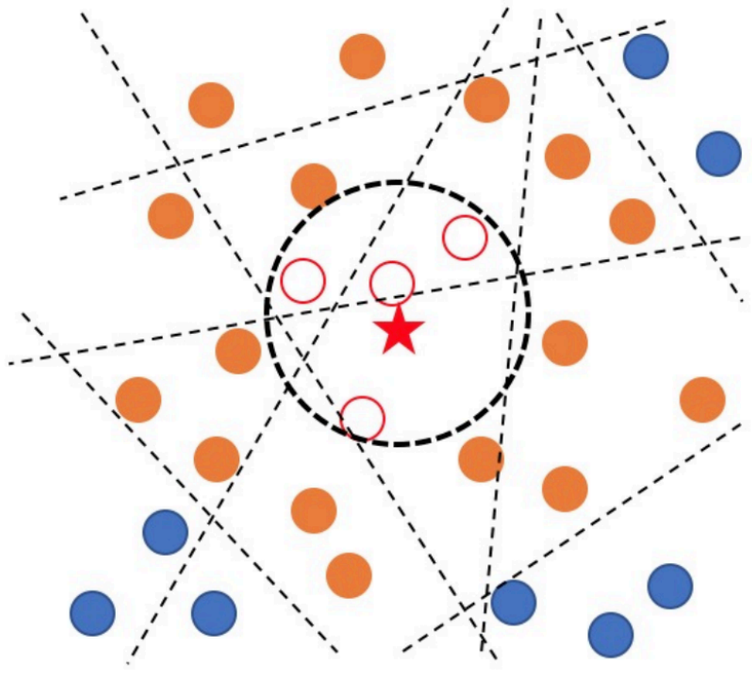
Given a query point q and an approximation ratio, a ε-NNS query returns a point $o \in D$ such that $\rho(o, q) \leq (1 + \varepsilon)\rho(o^*, q)$, where $o^* \in X$ is the exact nearest neighbour of q.

3 Ways to Attack The Problem

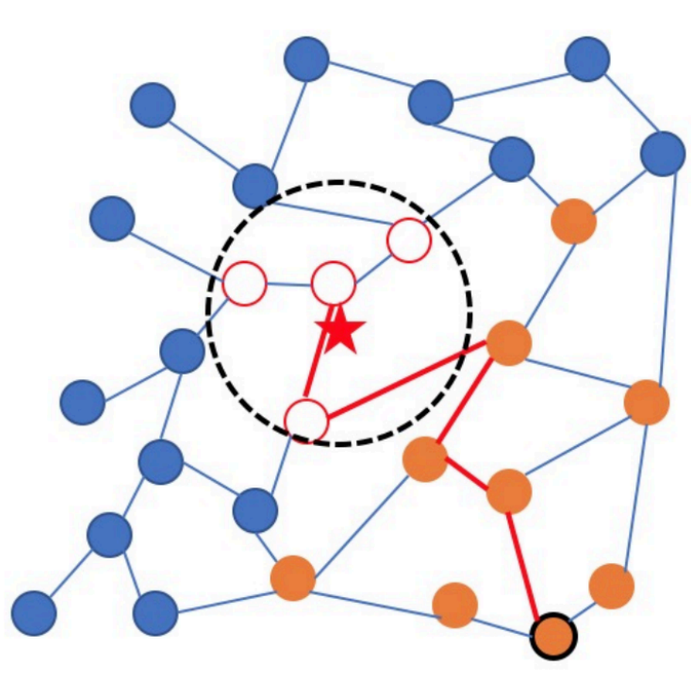
Trees, Mappings, Graphs



Partitioning: branch and bound



Mapping: LSH,
random projections

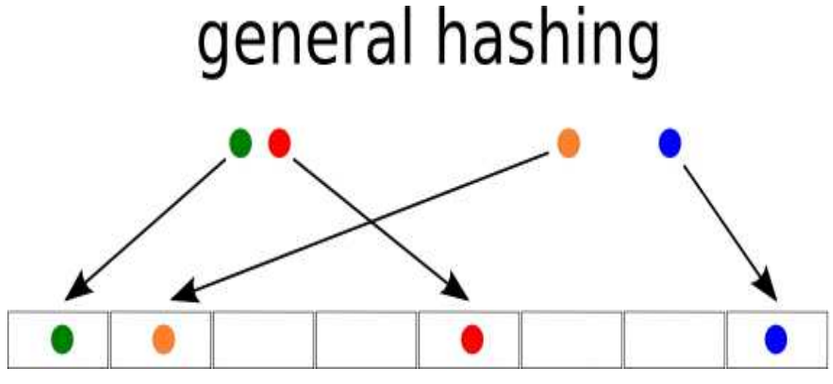


Graph based

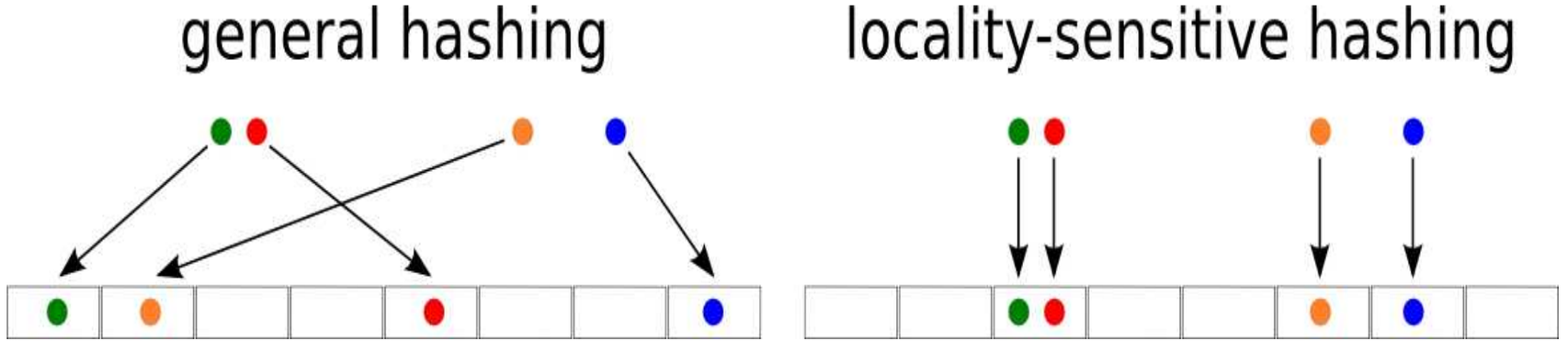
https://arxiv.org/pdf/1707.00143

Locality Sensitive Hashing (LSH)

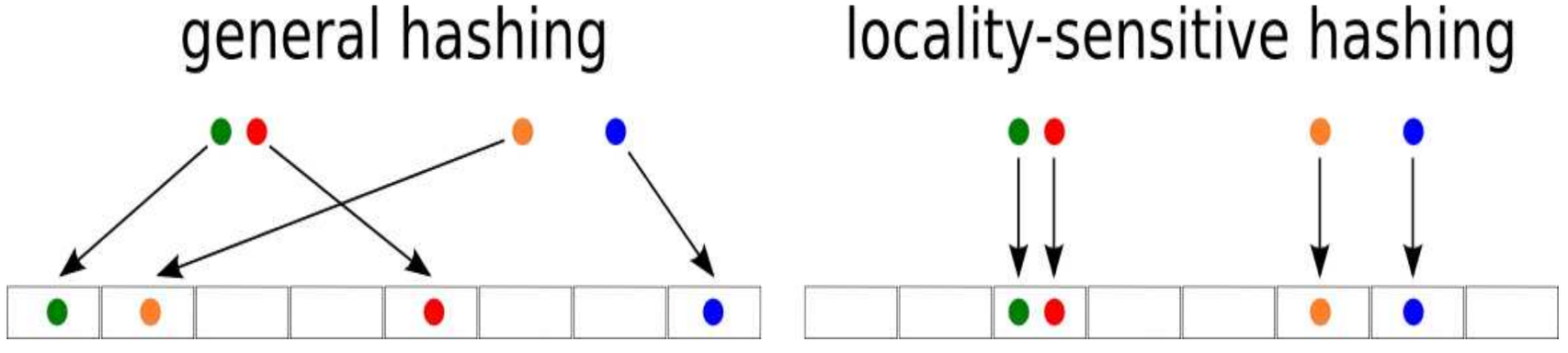
Locality Sensitive Hashing



Locality Sensitive Hashing

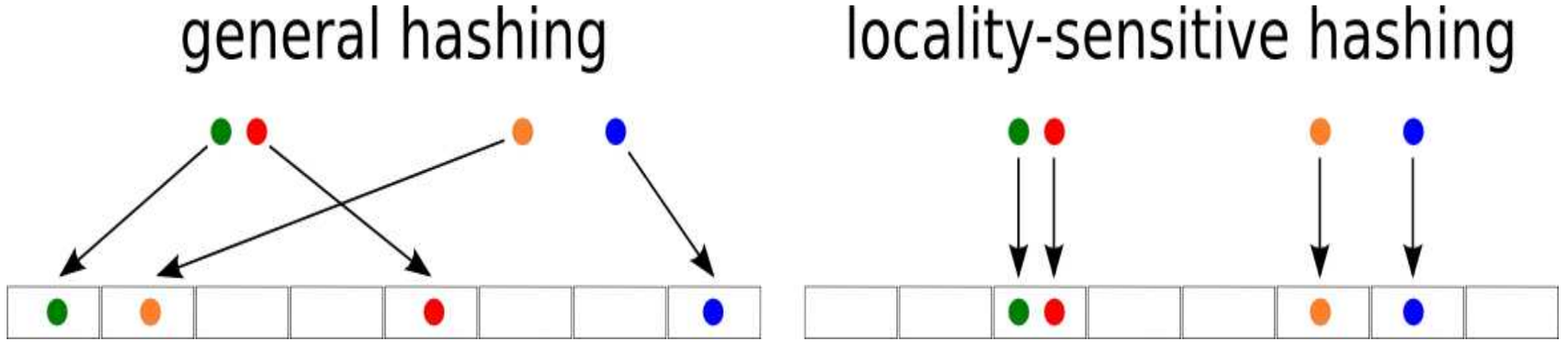


Locality Sensitive Hashing



The general idea of hashing is to avoid collisions. The idea of LSH is to exploit collisions for mapping points which are nearby (in geometrical sense) into the same bucket.

Locality Sensitive Hashing



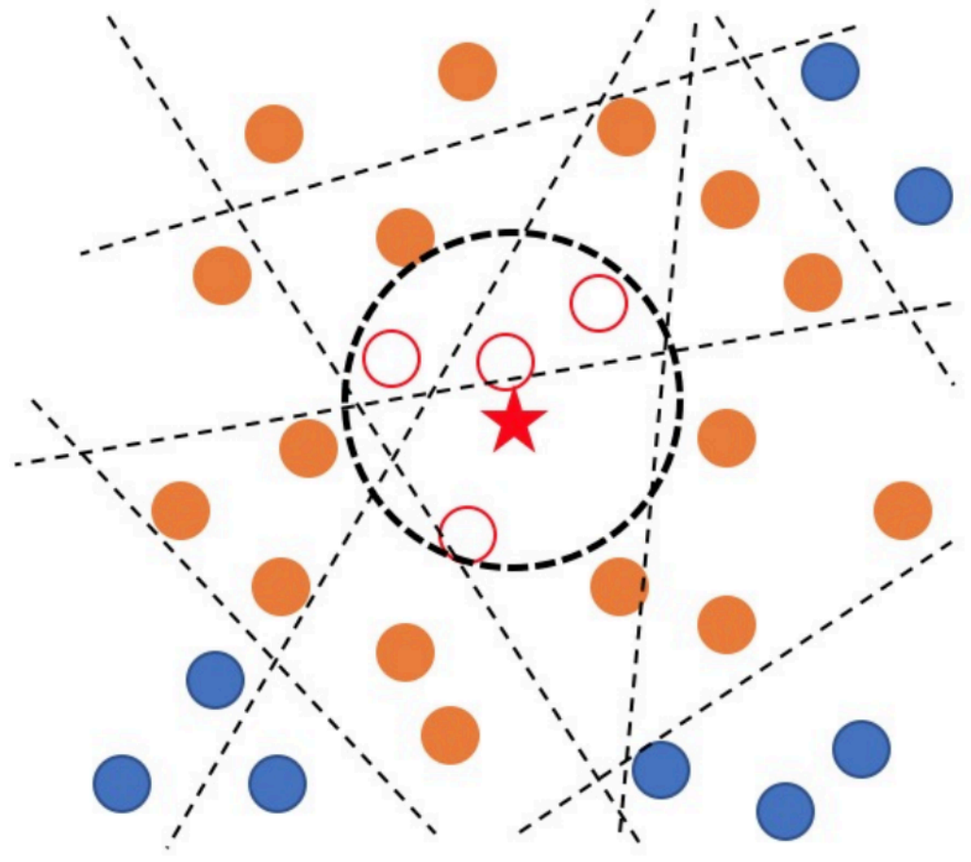
The general idea of hashing is to avoid collisions. The idea of LSH is to exploit collisions for mapping points which are nearby (in geometrical sense) into the same bucket.

[P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998, pp. 604–613.]

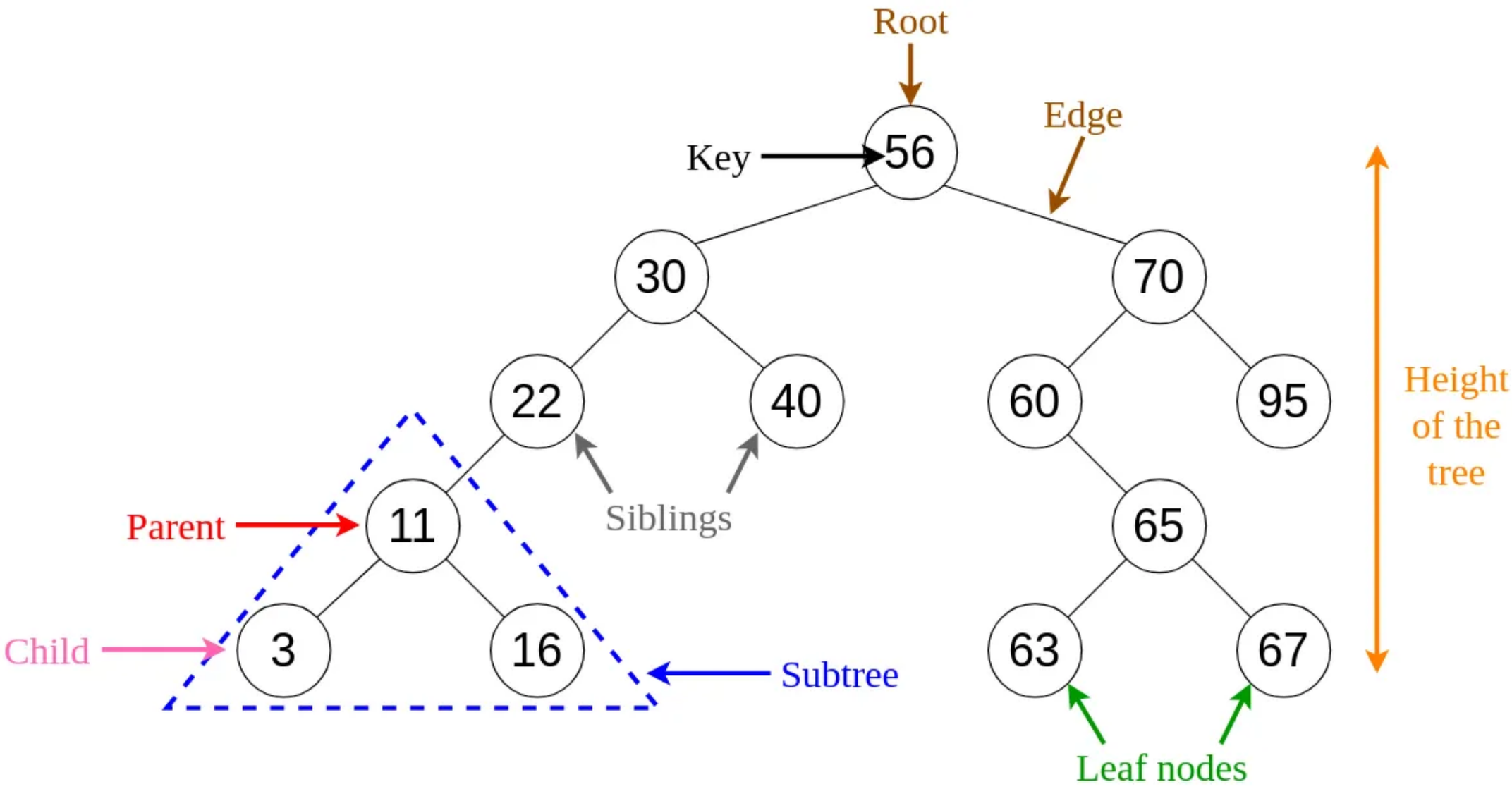
LSH: Random Projections

$$H(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \dots, h_M(\boldsymbol{x})]^T$$

$$h_m(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{a}^T \boldsymbol{x} + b}{W} \right\rfloor$$



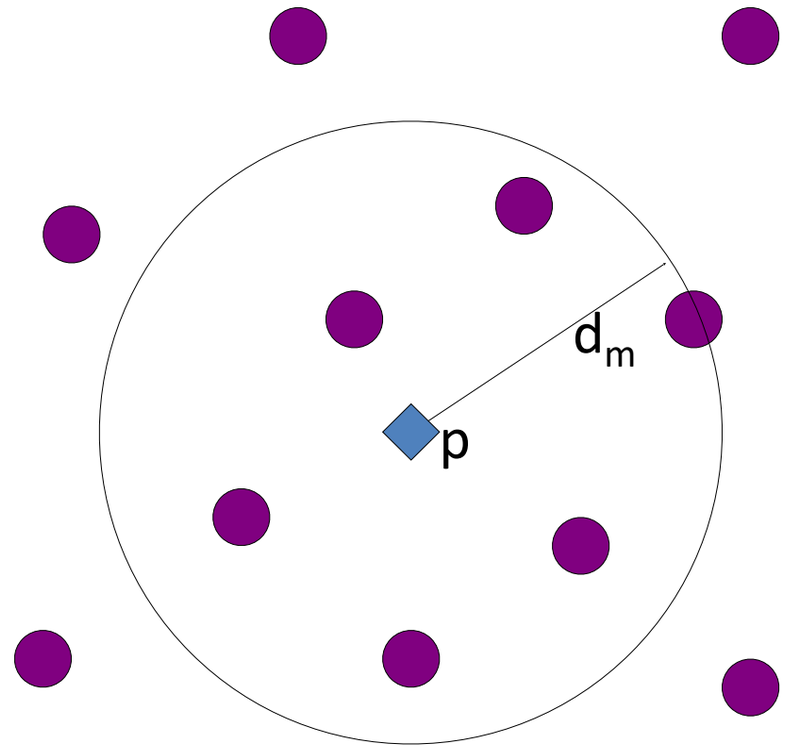
Trees: Hierarchical Space Partitioning



https://faun.pub/introduction-to-binary-search-trees-8e225aca7037

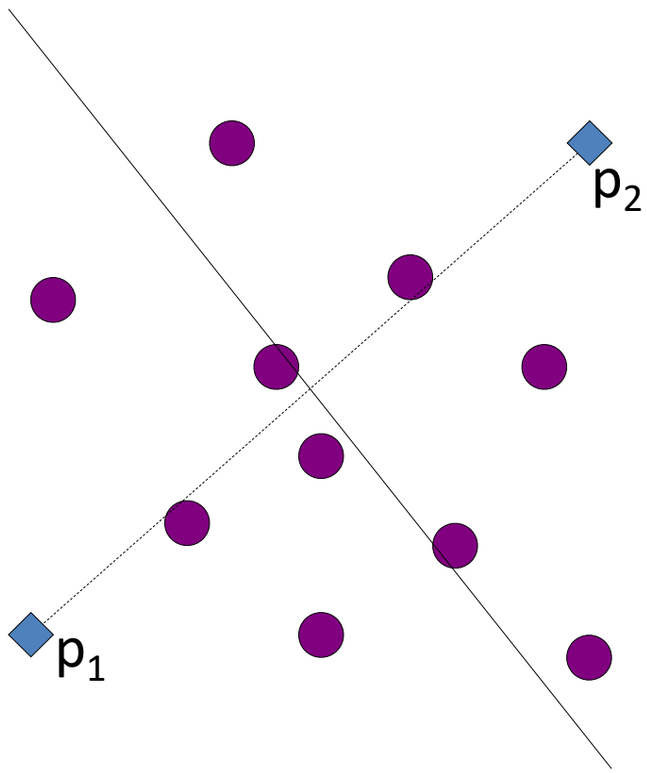
Ball Partitioning

- Inner set: $\{x \in X \mid \rho(p, x) \leq d_m\}$
- Outer set: $\{x \in X \mid \rho(p, x) > d_m\}$



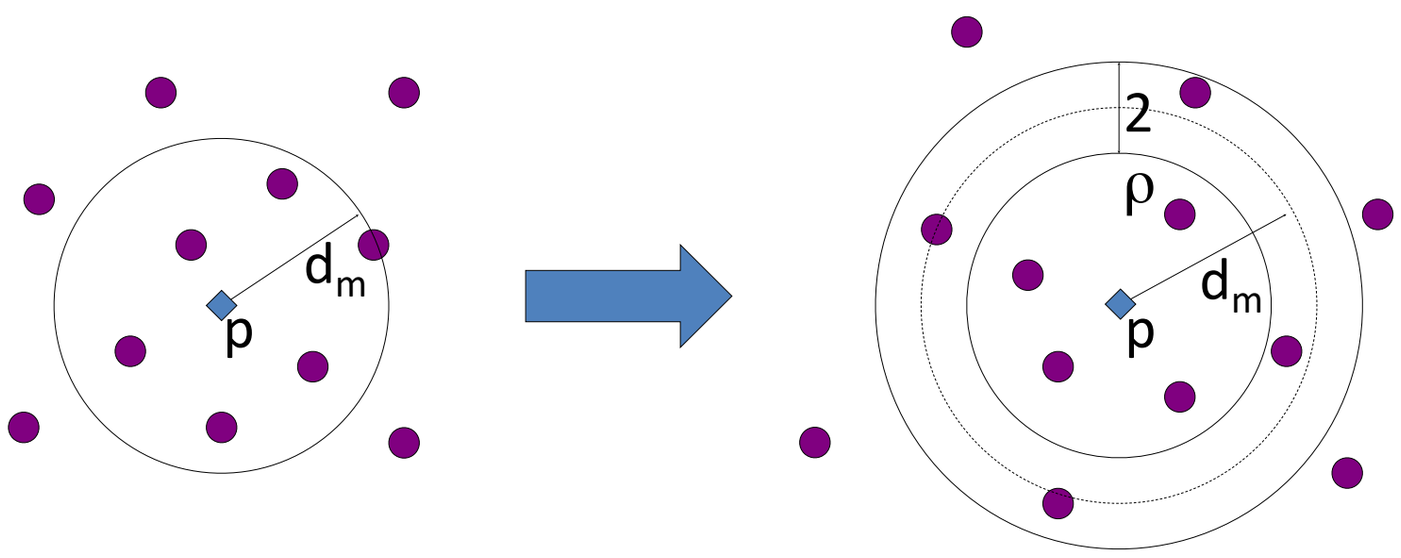
Generalized Hyper-plane

- $\{x \in X \mid \rho(p_1, x) \leq d(p_2, x)\}$
- $\{x \in X \mid \rho(p_1, x) > d(p_2, x)\}$

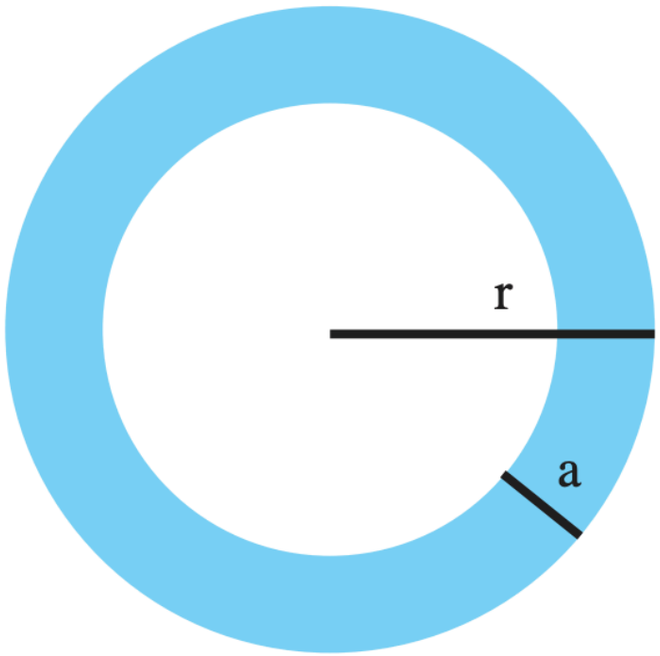


Excluded Middle Partitioning

- Inner set: $\{x \in X \mid \rho(p,x) \leq d_m - \rho\}$
- Outer set: $\{x \in X \mid \rho(p,x) > d_m + \rho\}$
- Excluded set: otherwise



What fraction of the total volume of the ball $B^n(r)$ is contained in the shell of radius r and thickness a ?

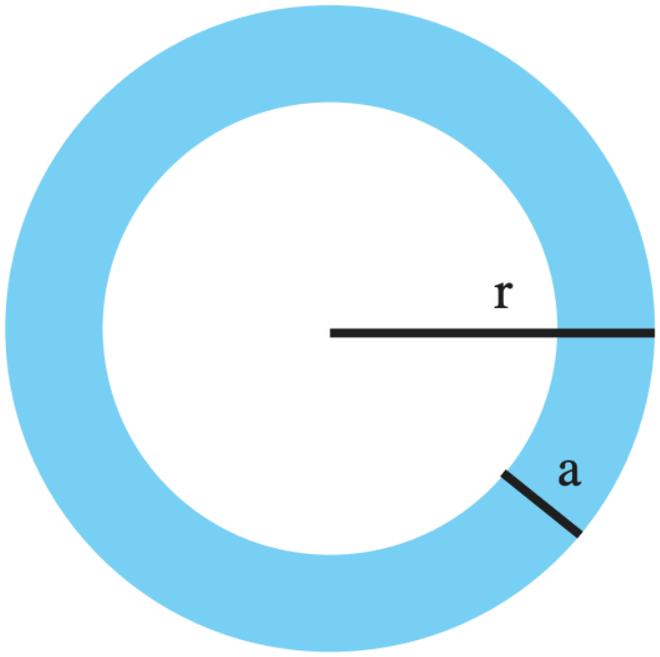


A shell of radius r and thickness a

$$\frac{V(B^n(r)) - V(B^n(r-a))}{V(B^n(r))} = \frac{V(B^n(1))(r^n - (r-a)^n)}{V(B^n(1))r^n}$$
$$= \frac{r^n - (r-a)^n}{r^n}$$
$$= 1 - (1 - a/r)^n.$$

[Feres, Renato. "Geometry in (Very) High Dimensions." (2006)]

What fraction of the total volume of the ball $B^n(r)$ is contained in the shell of radius r and thickness a ?



A shell of radius r and thickness *a*

if $n \geq 500$, more than 99% of the volume of $B^n(r)$ is contained in a shell of thickness corresponding to 1% of the radius!

[Feres, Renato. "Geometry in (Very) High Dimensions." (2006)]

Greedy Walk, Beam Search

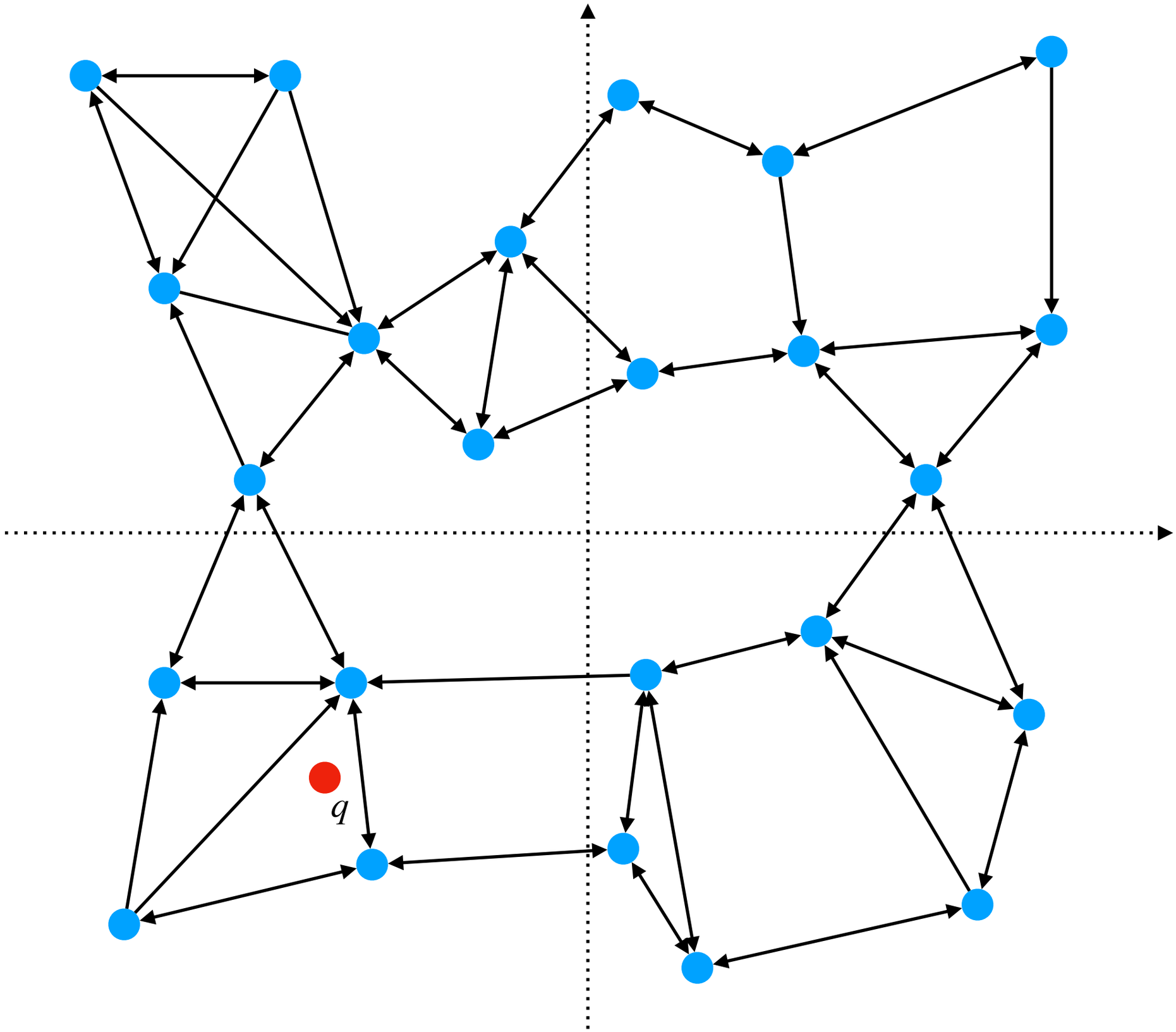
Local Search = Beam Search = Search with Backtracking

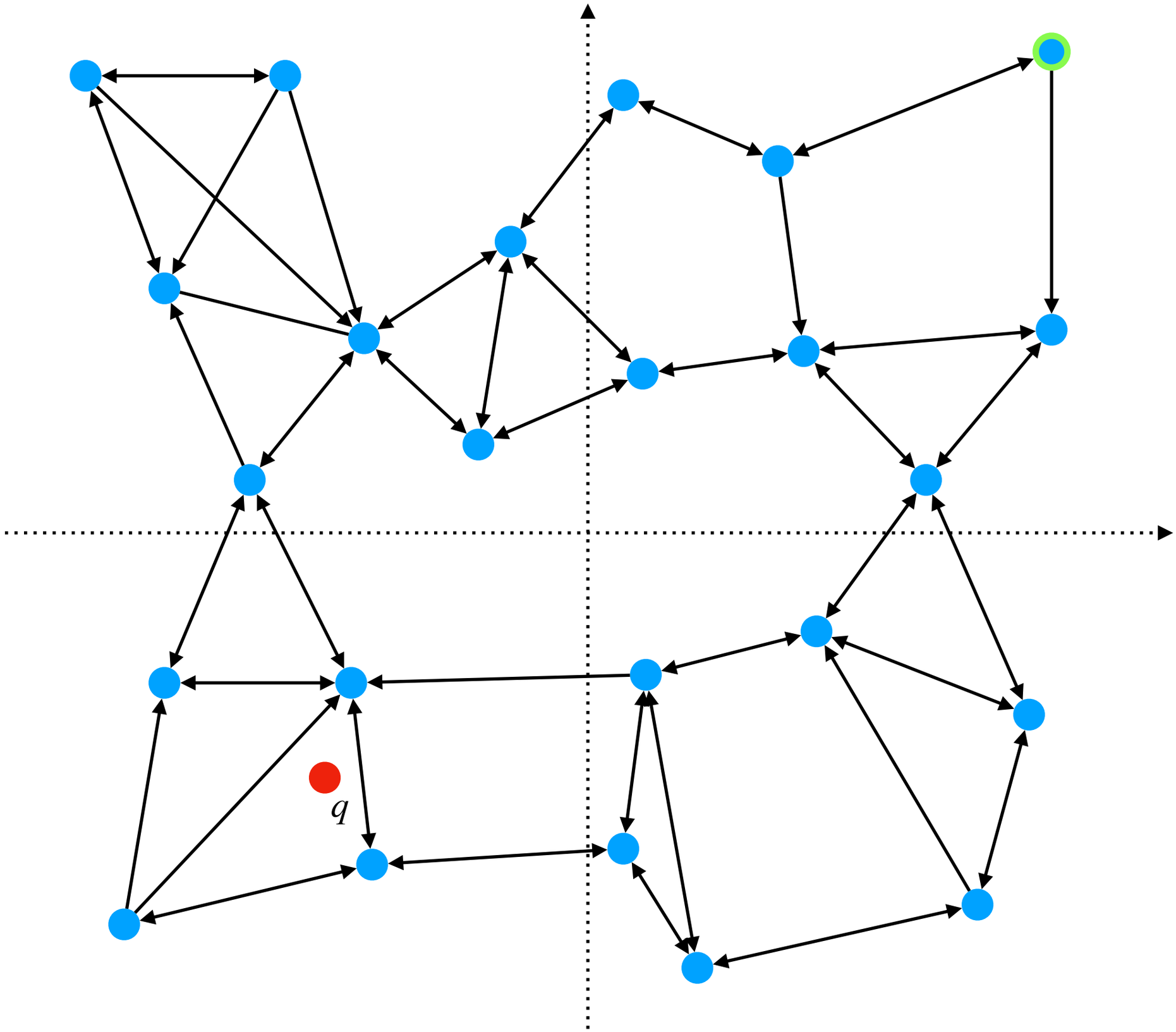
Algorithm 1 LocalSearch(G, q, C, k, L)

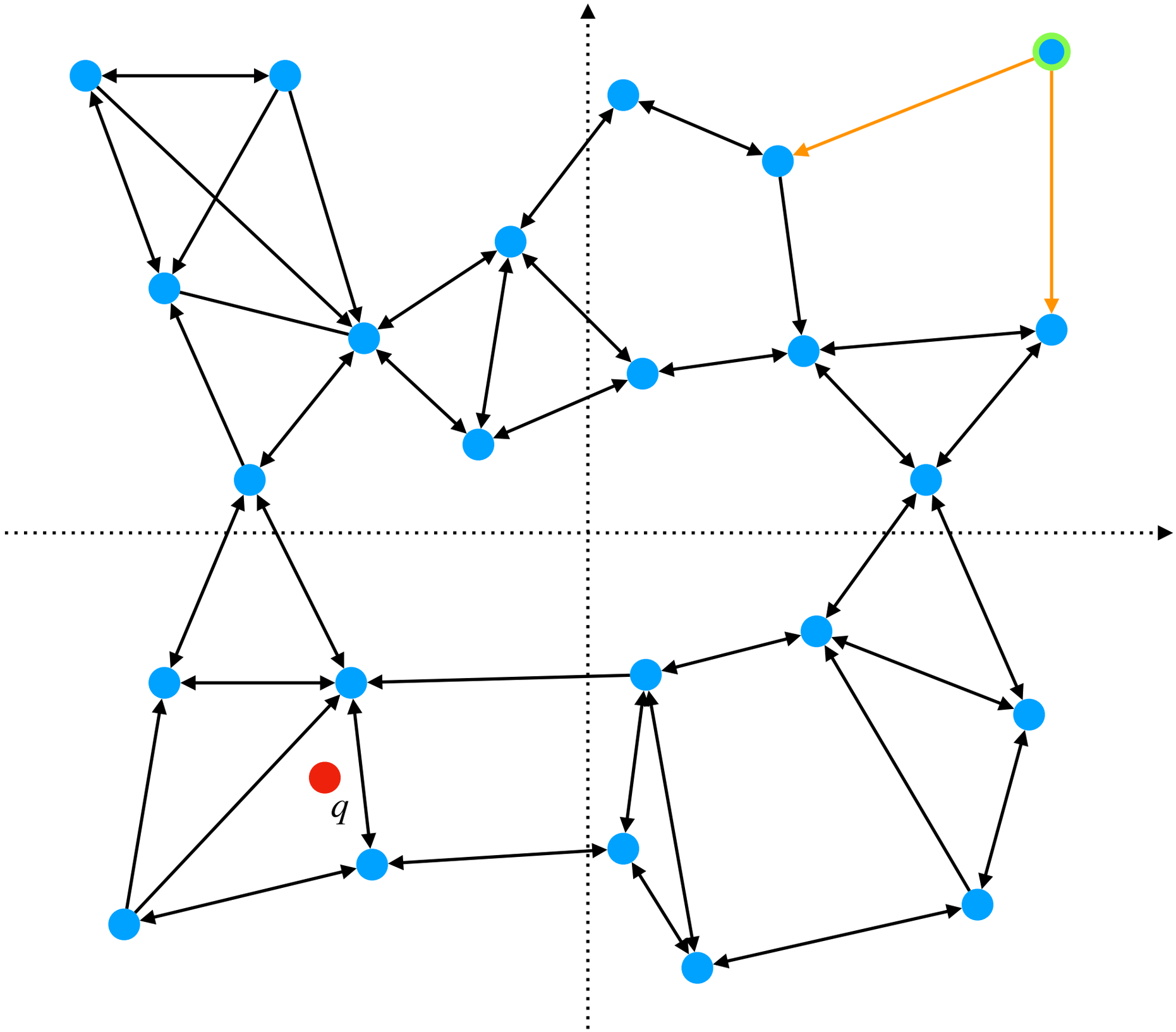
Require: Graph $G = (V, E)$, query $q \in \mathbb{R}^d$, initial set of candidate vertexes $C \subset V$, $k \in \mathbb{N}$, $L \in \mathbb{N}$

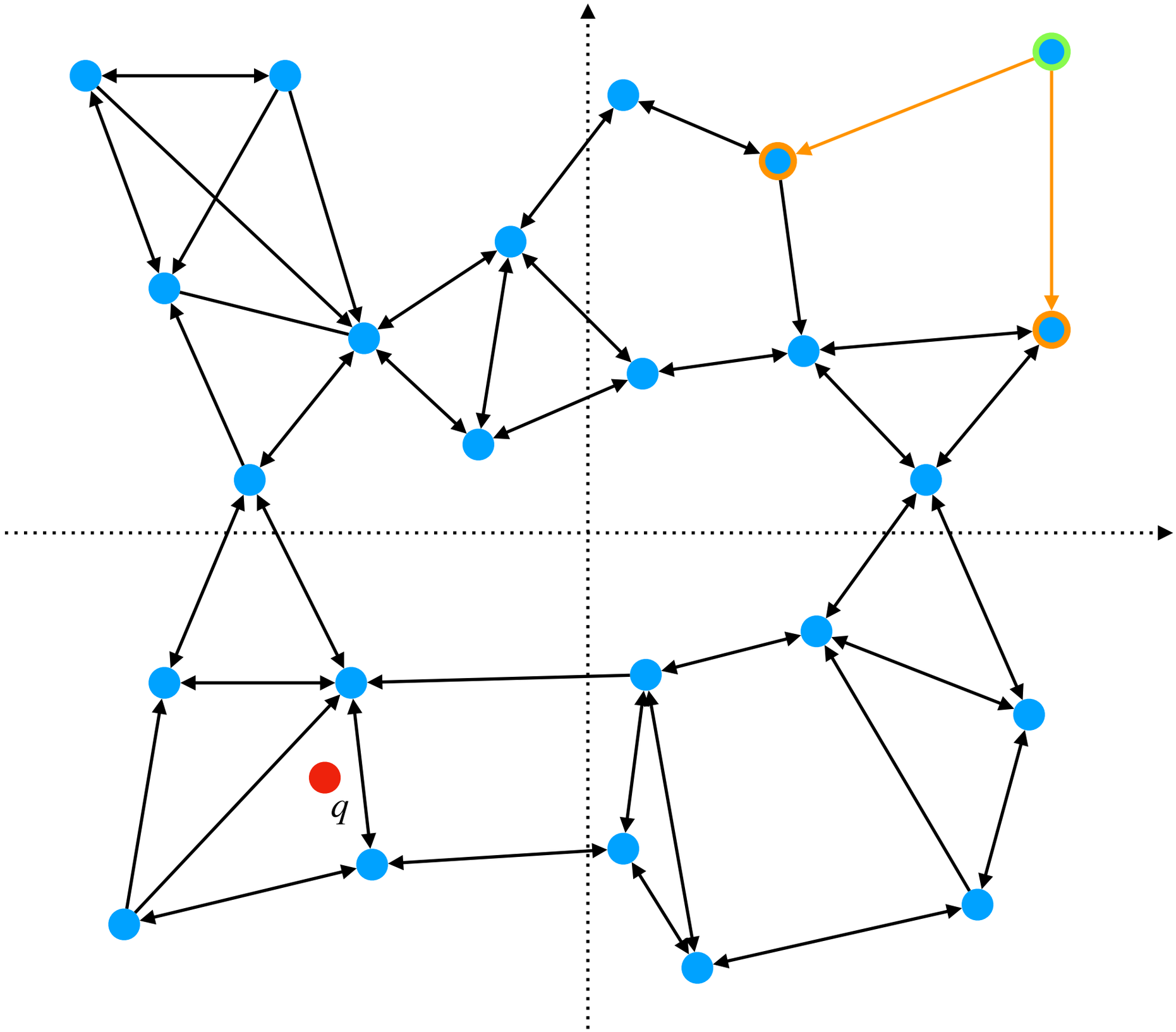
Ensure: approximate k-nearest neighbors $V^* \subset V$

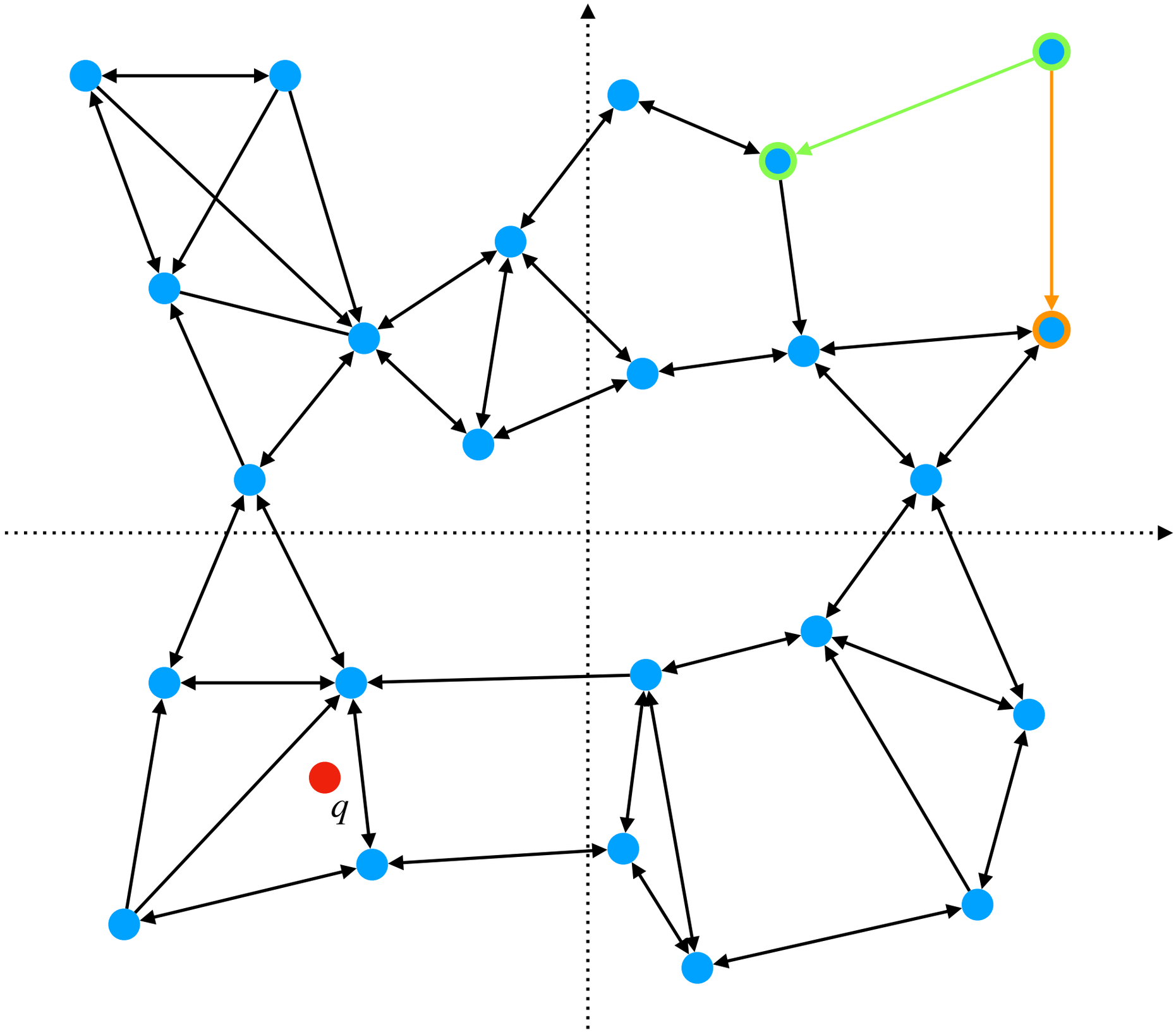
```
while True do
    u ← nearest unvisited point to q in C
    U ← {v | (u, v) ∈ E}
    for v ∈ U do
        if v is not visited then
            C ← C ∪ {v}
    if |C| > L then
        C ← top L nearest points to q in C
    if C is not updated then
        break
return nearest point to q in C
```

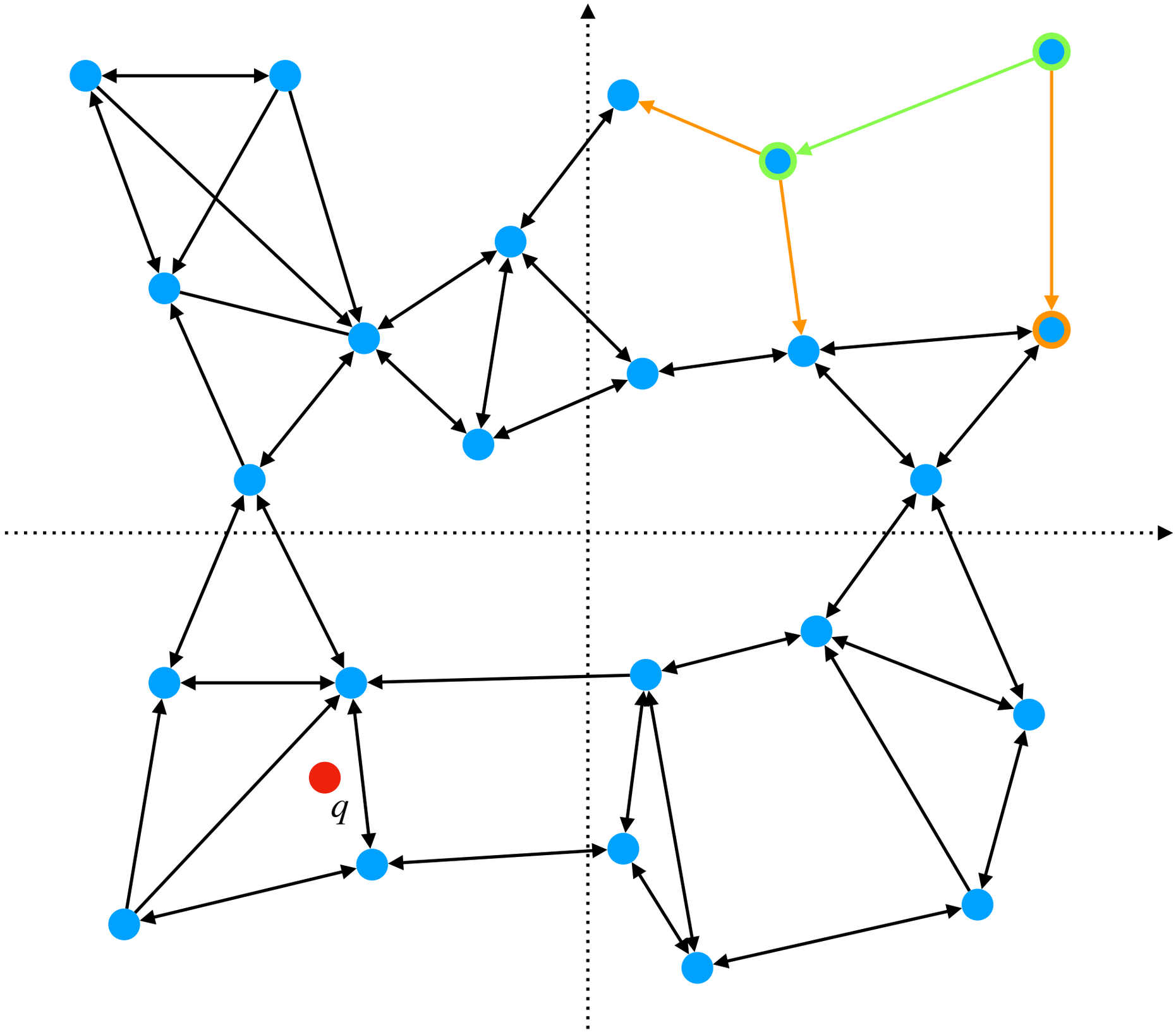

q

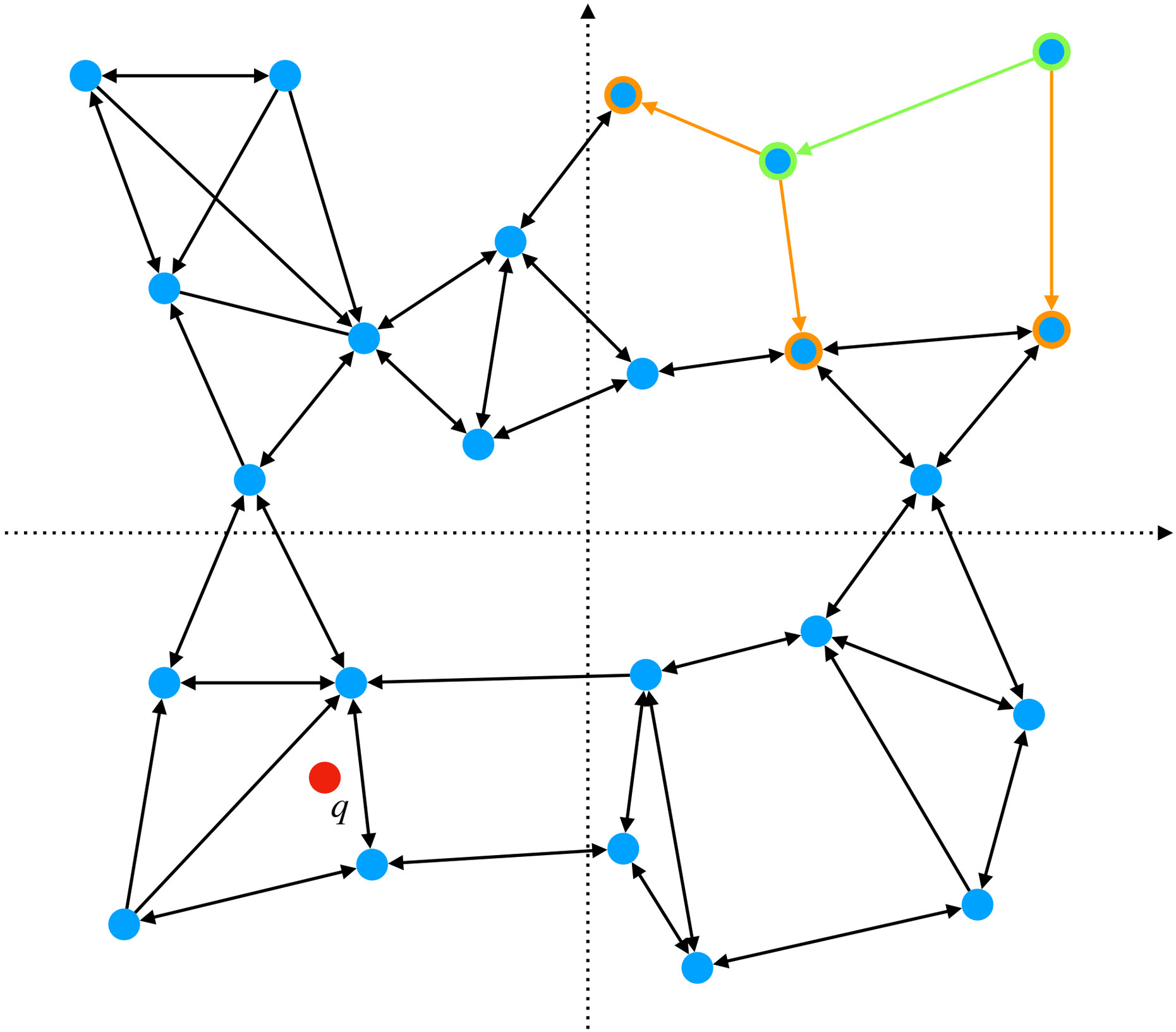

q

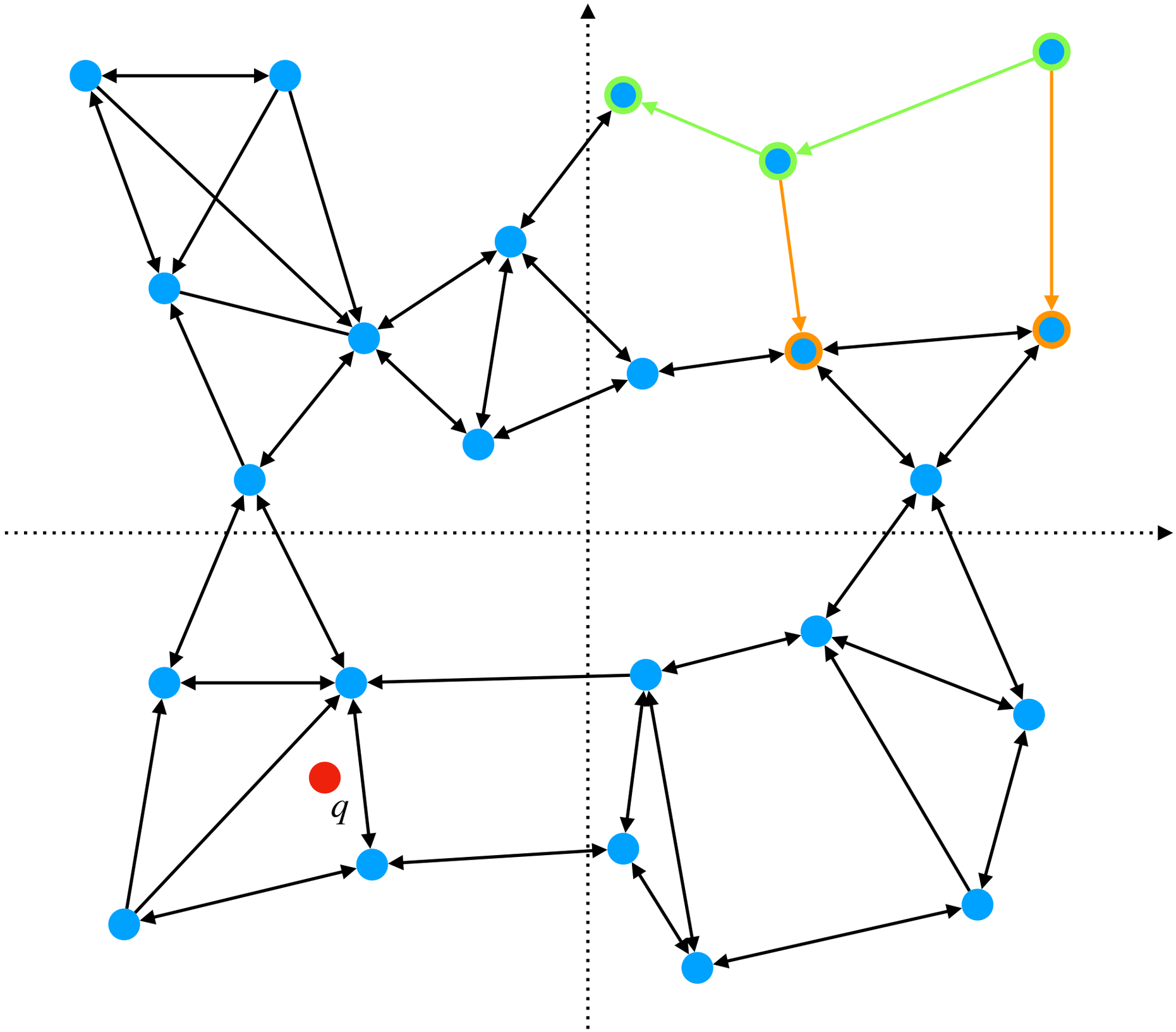

q

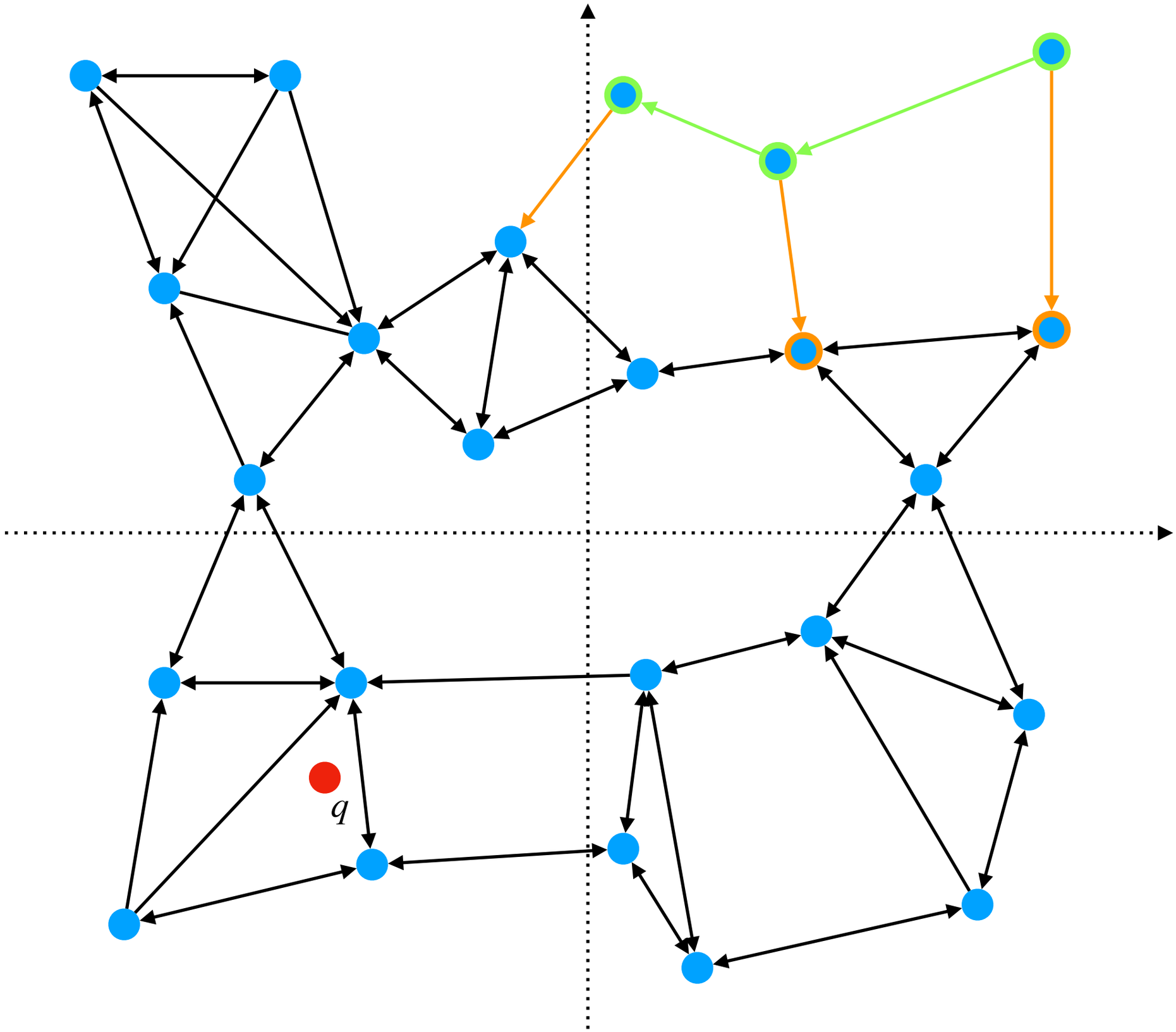

q

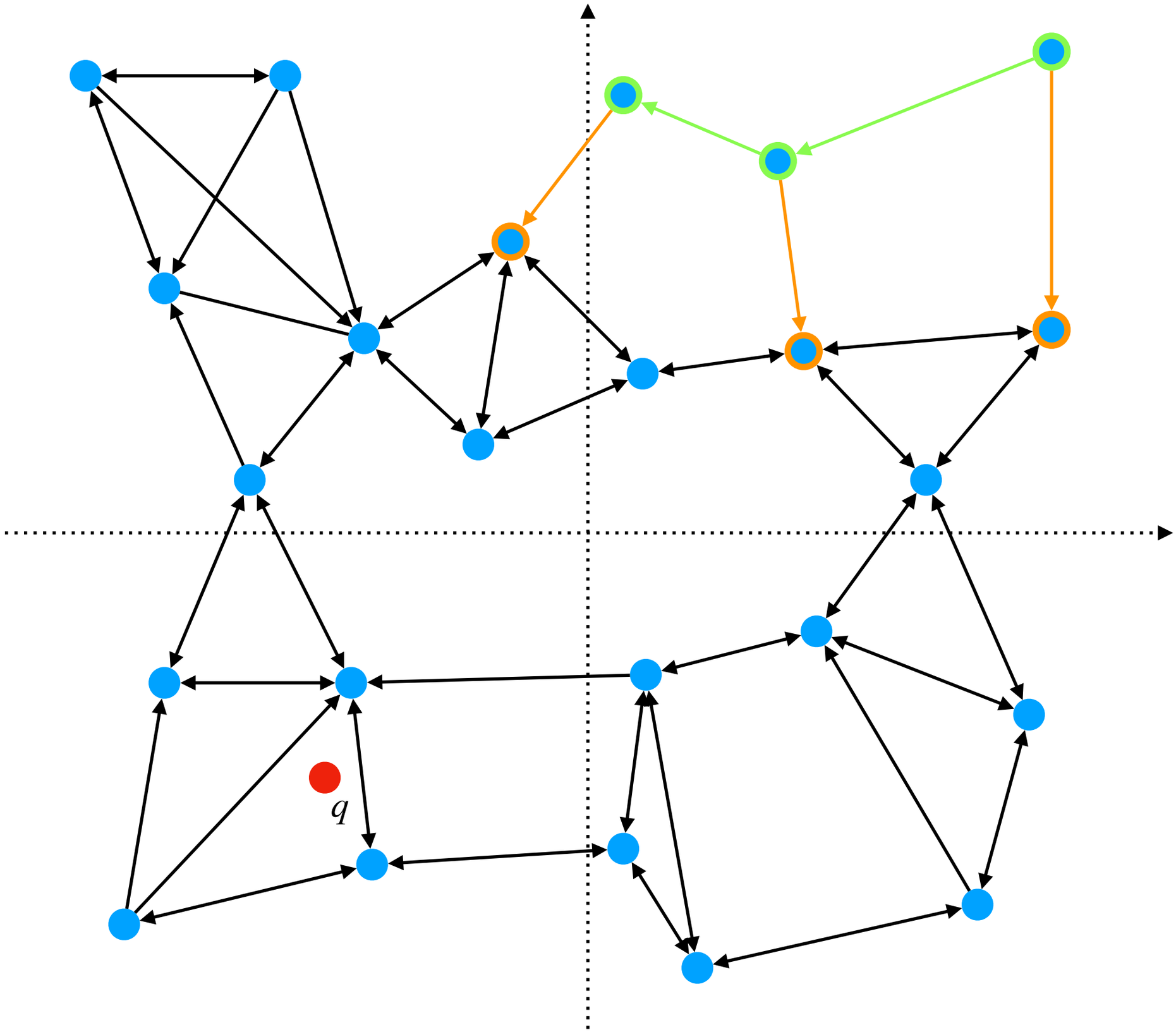

q

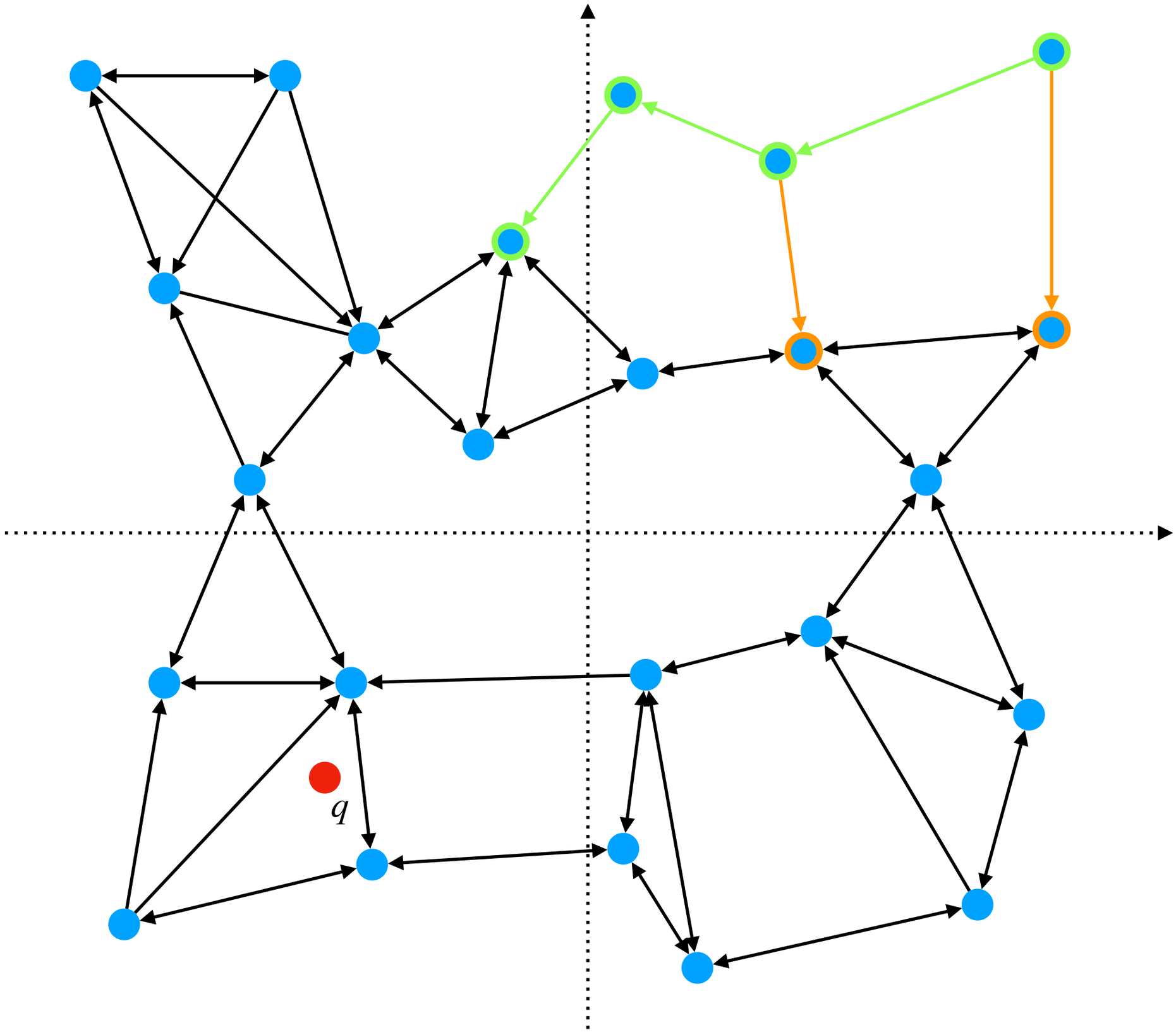

q

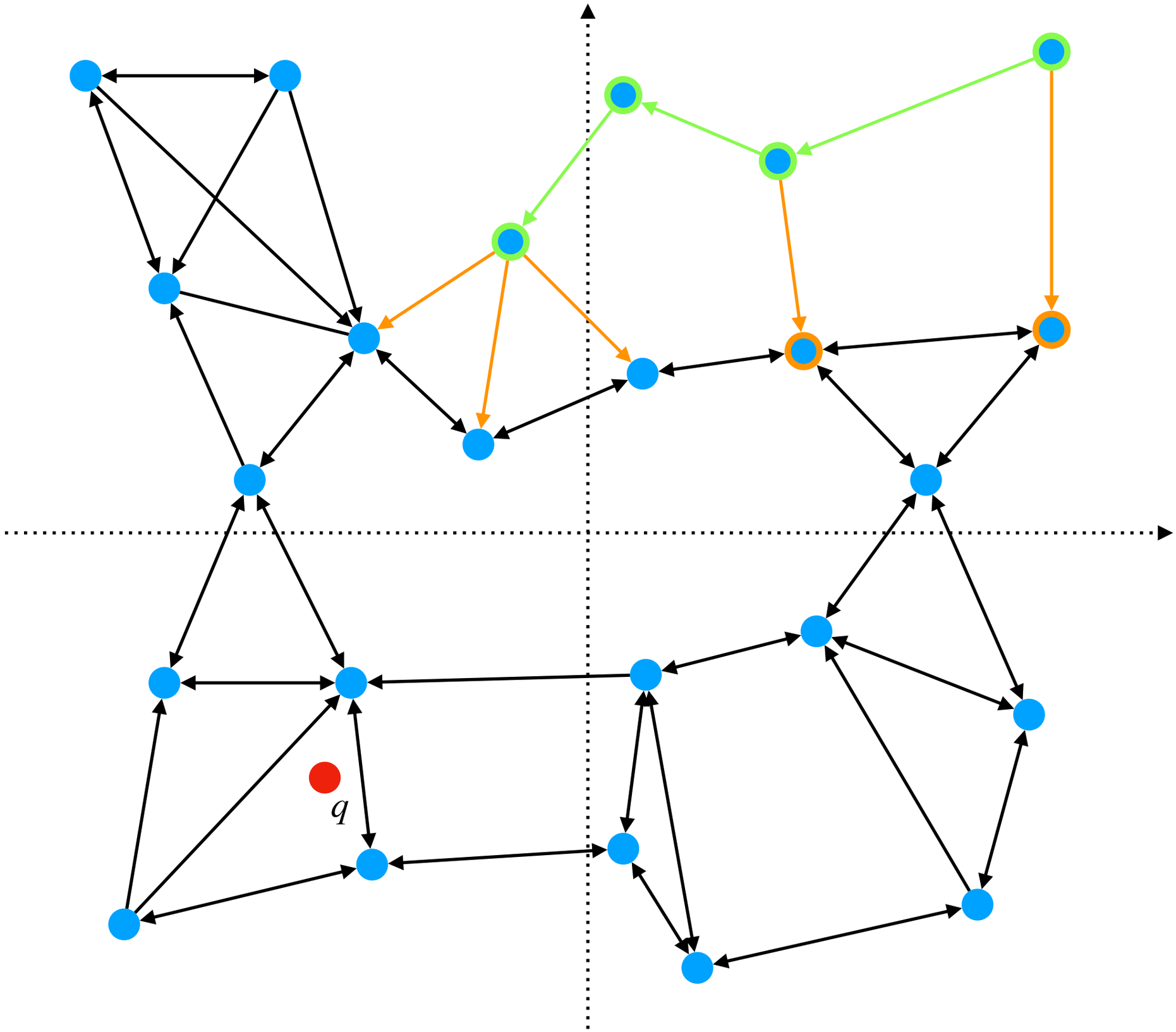

q

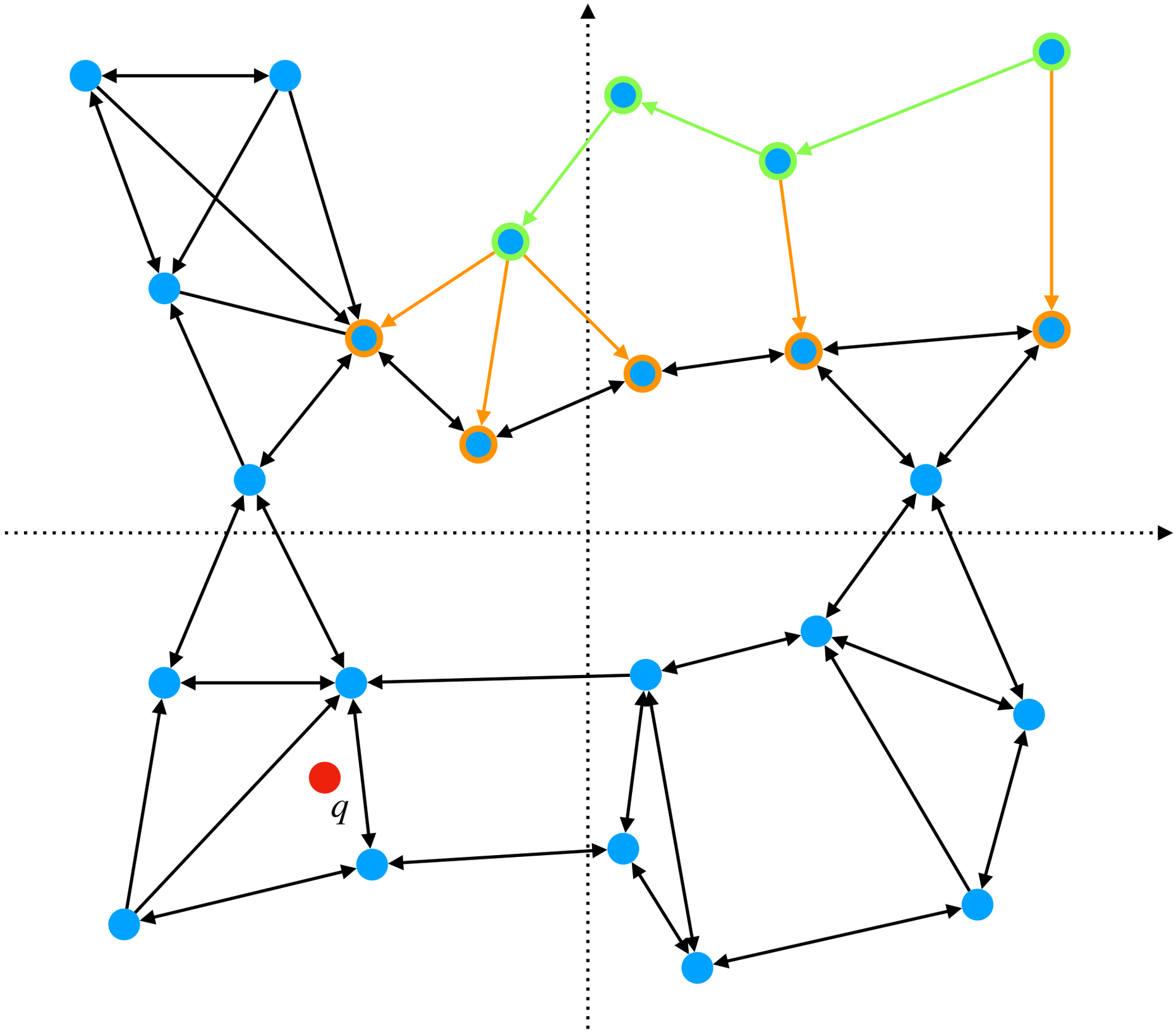

q

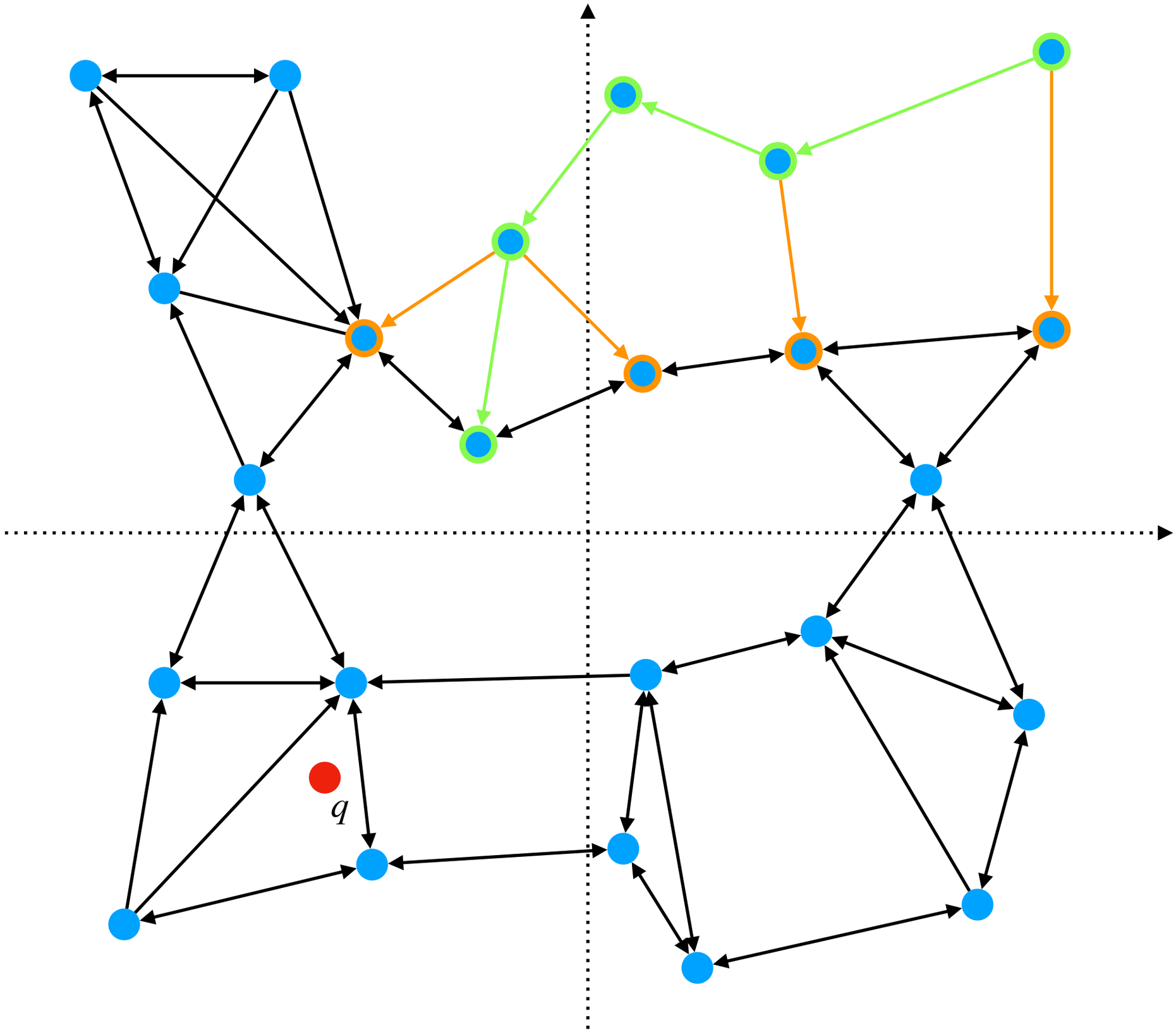

q

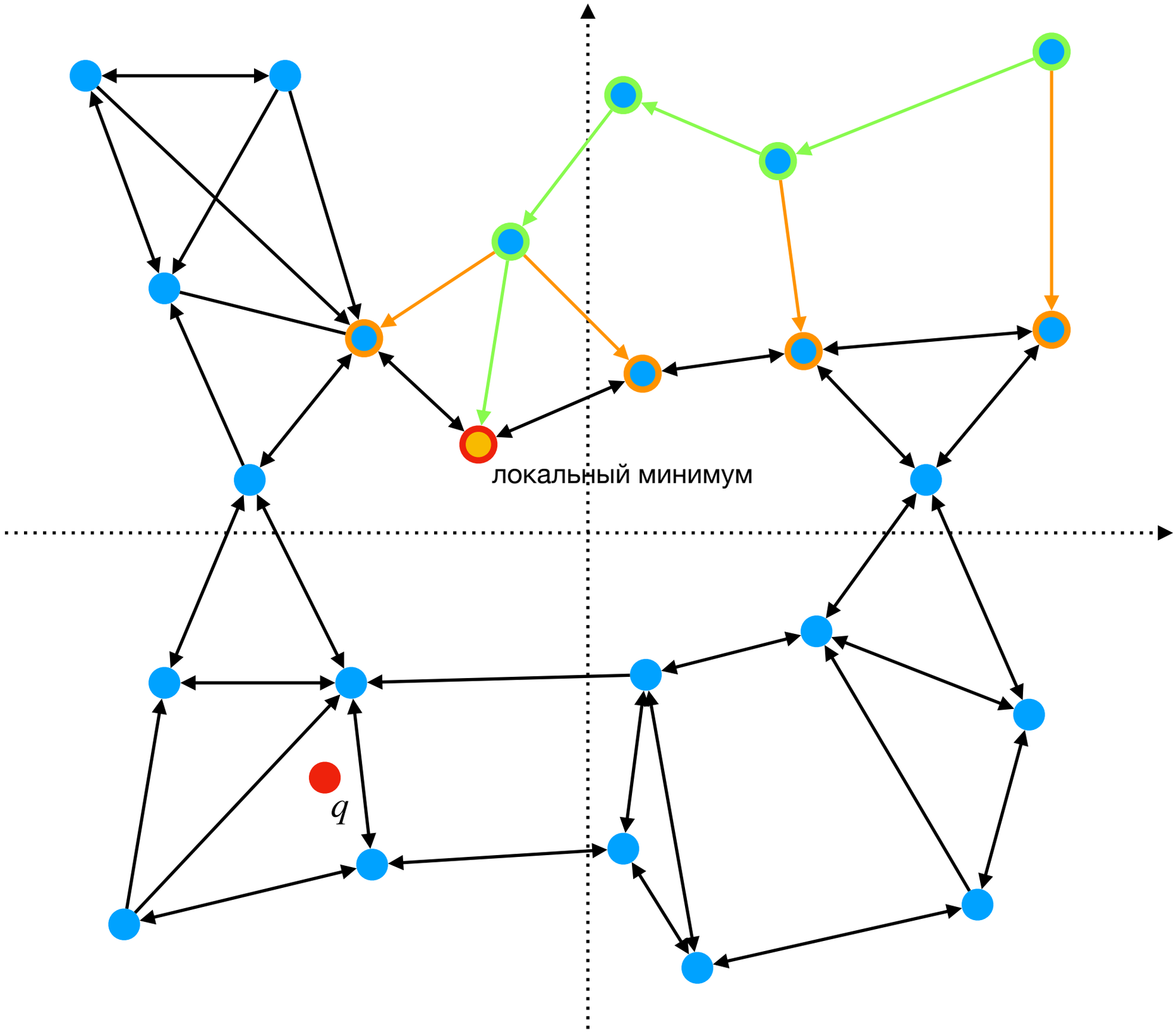

q


q

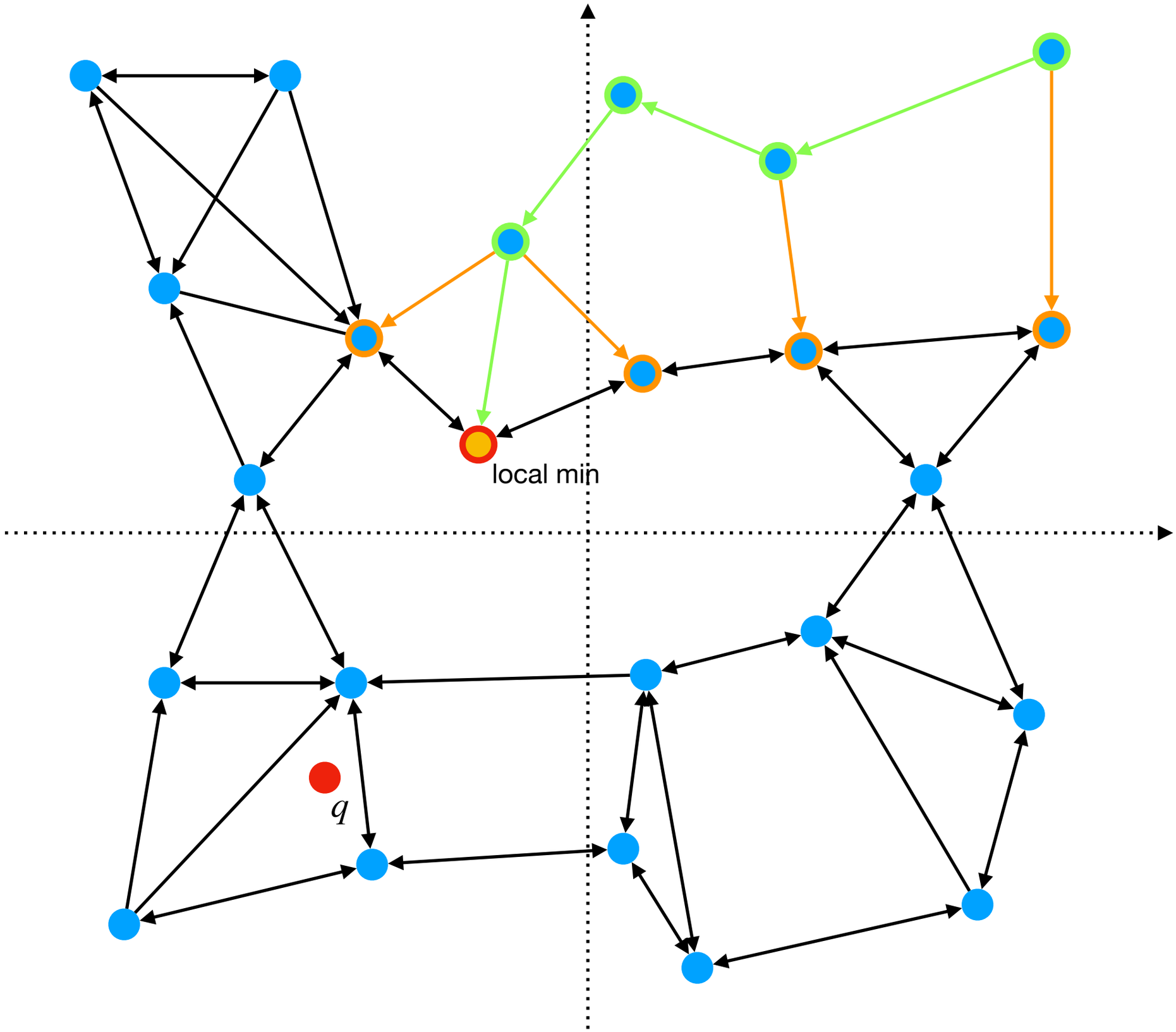

q

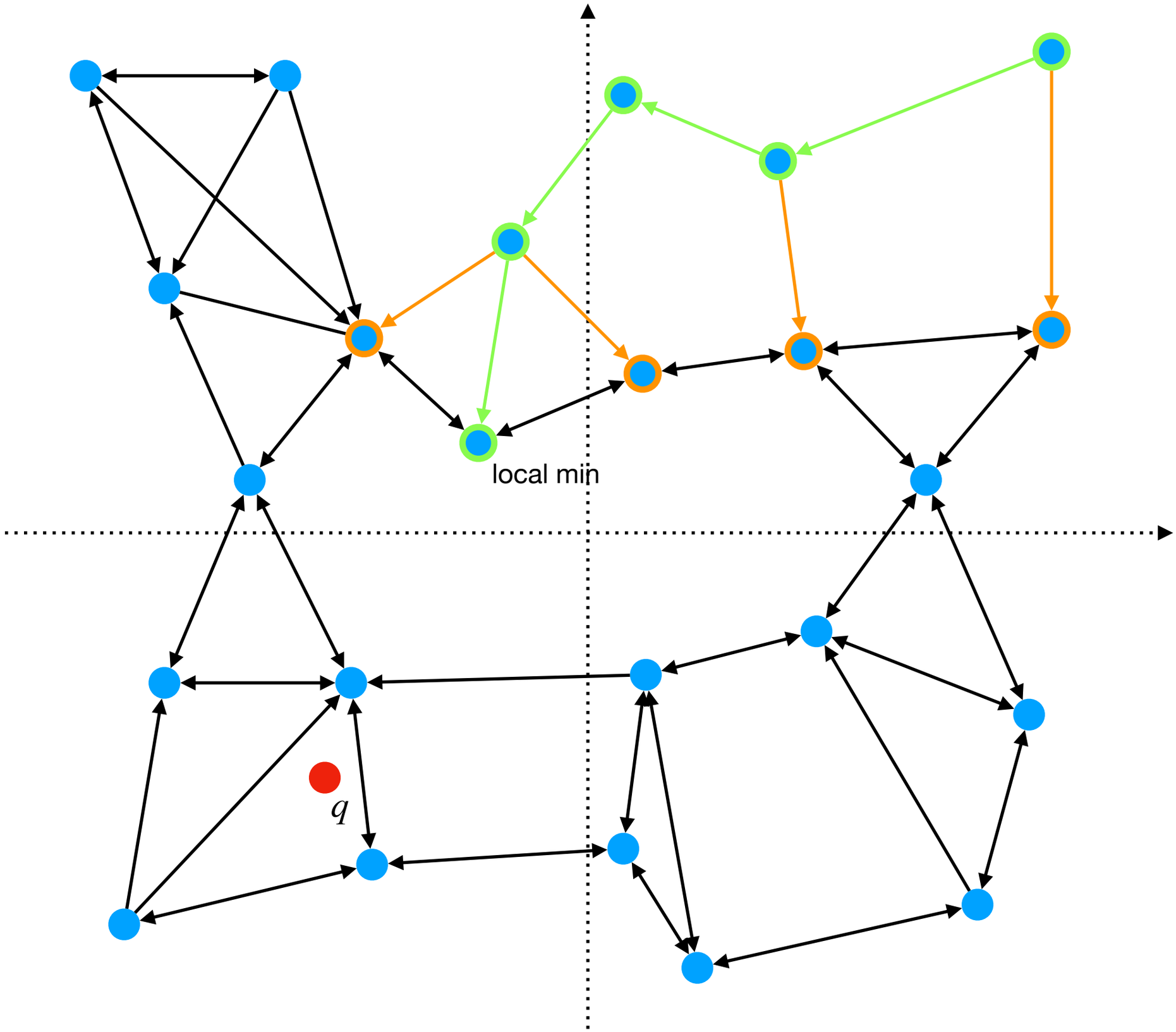

q

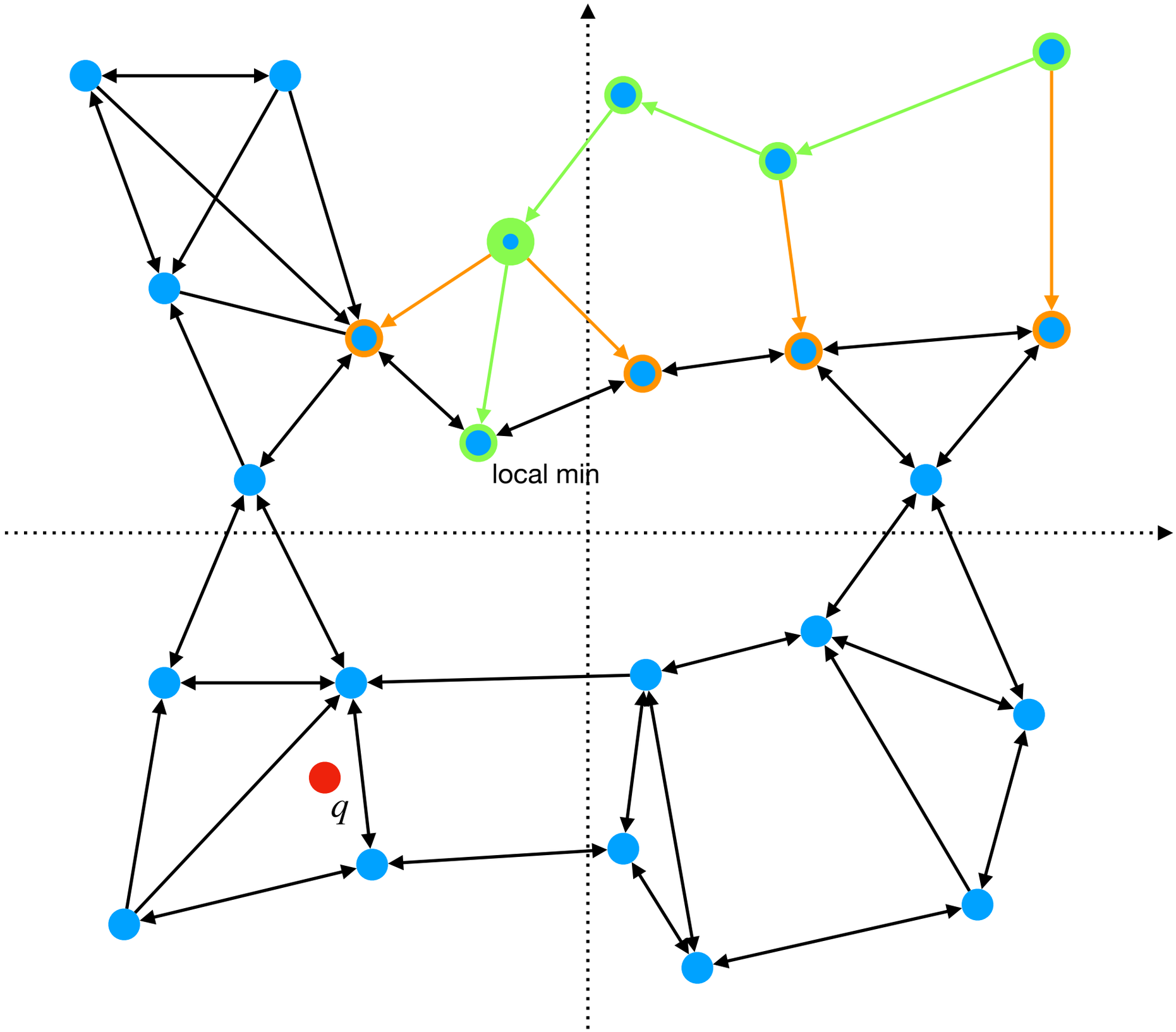

q

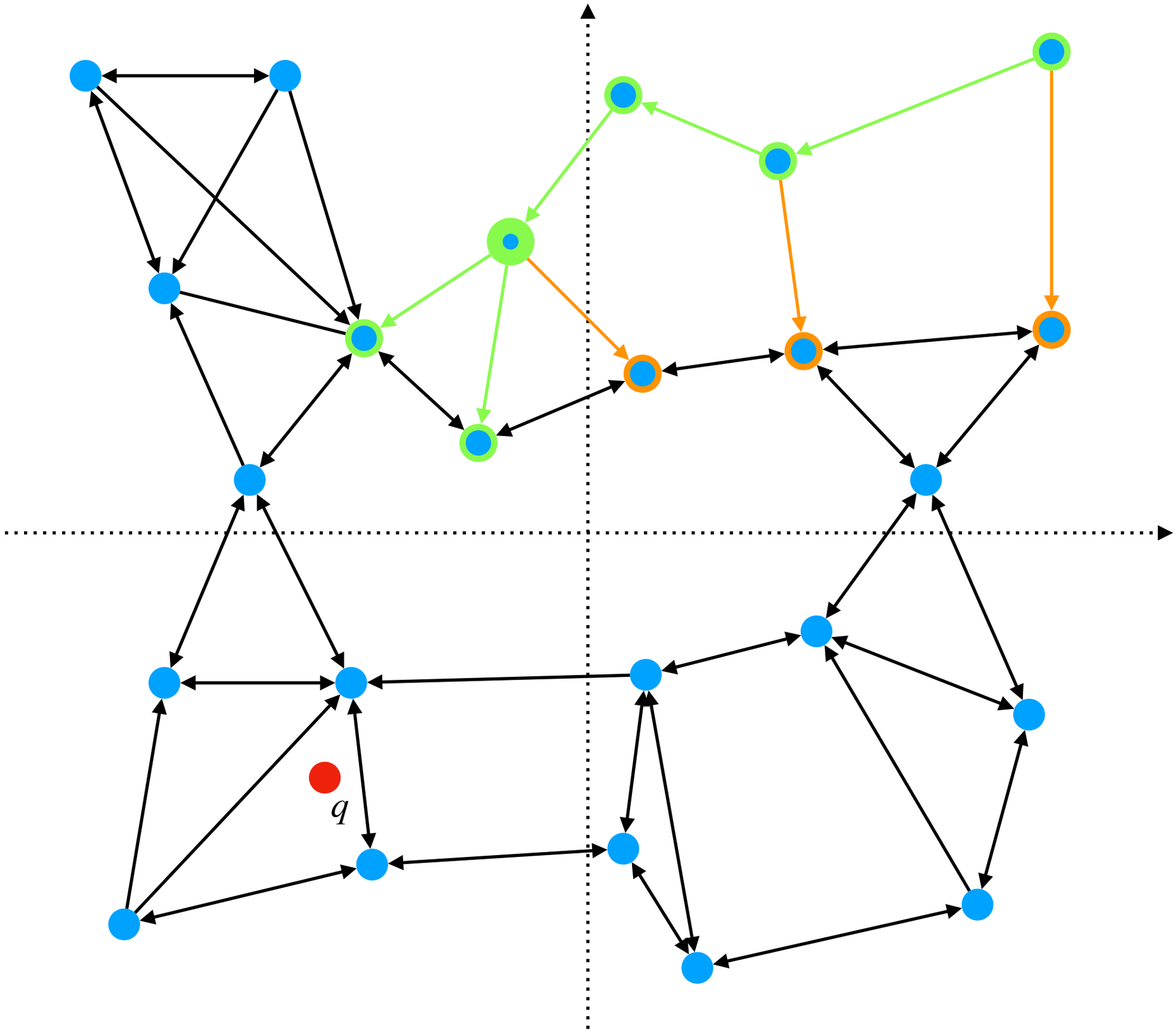

локальный минимум
q

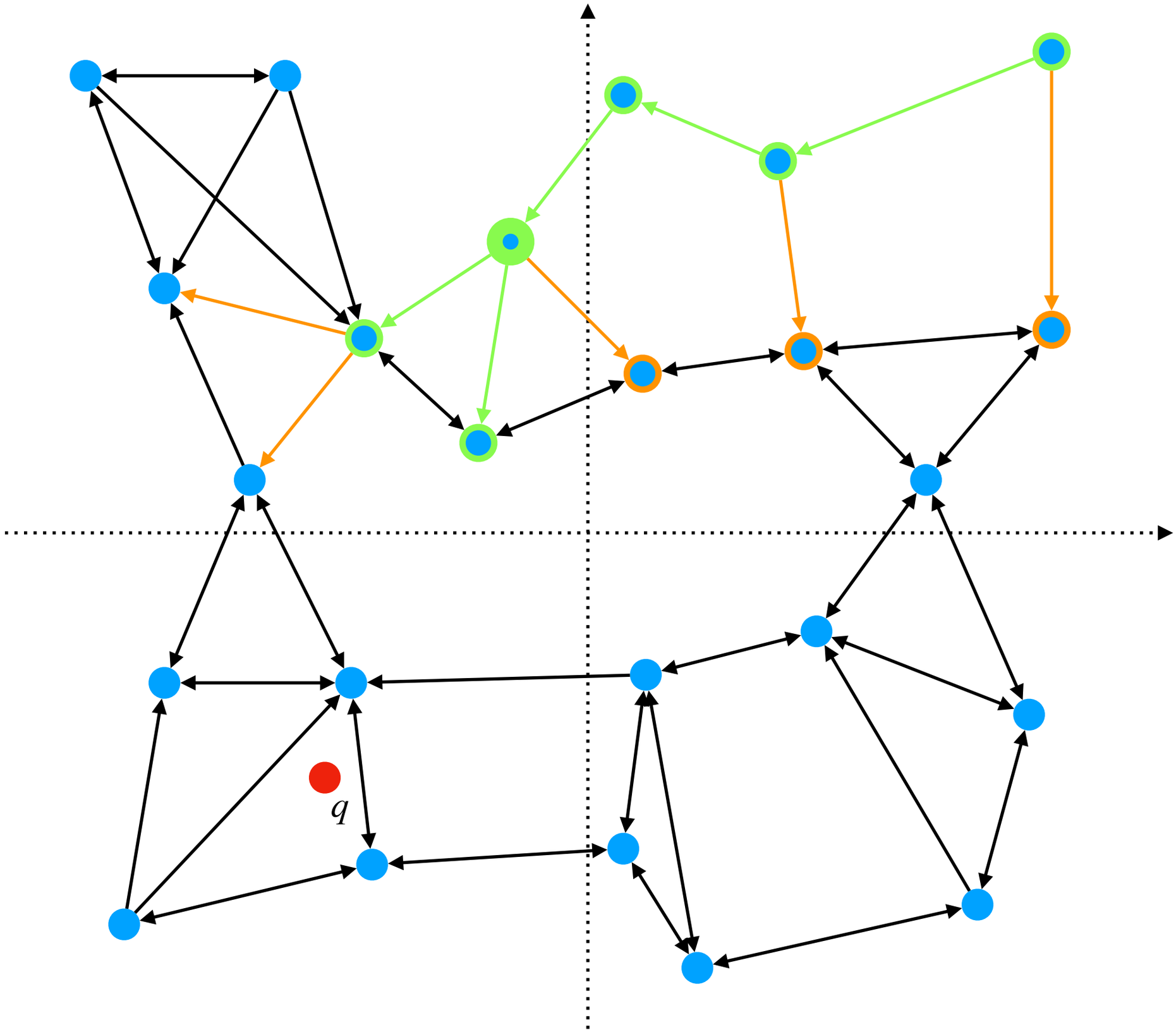
Lets add ability to avoid local minimums by backtracking

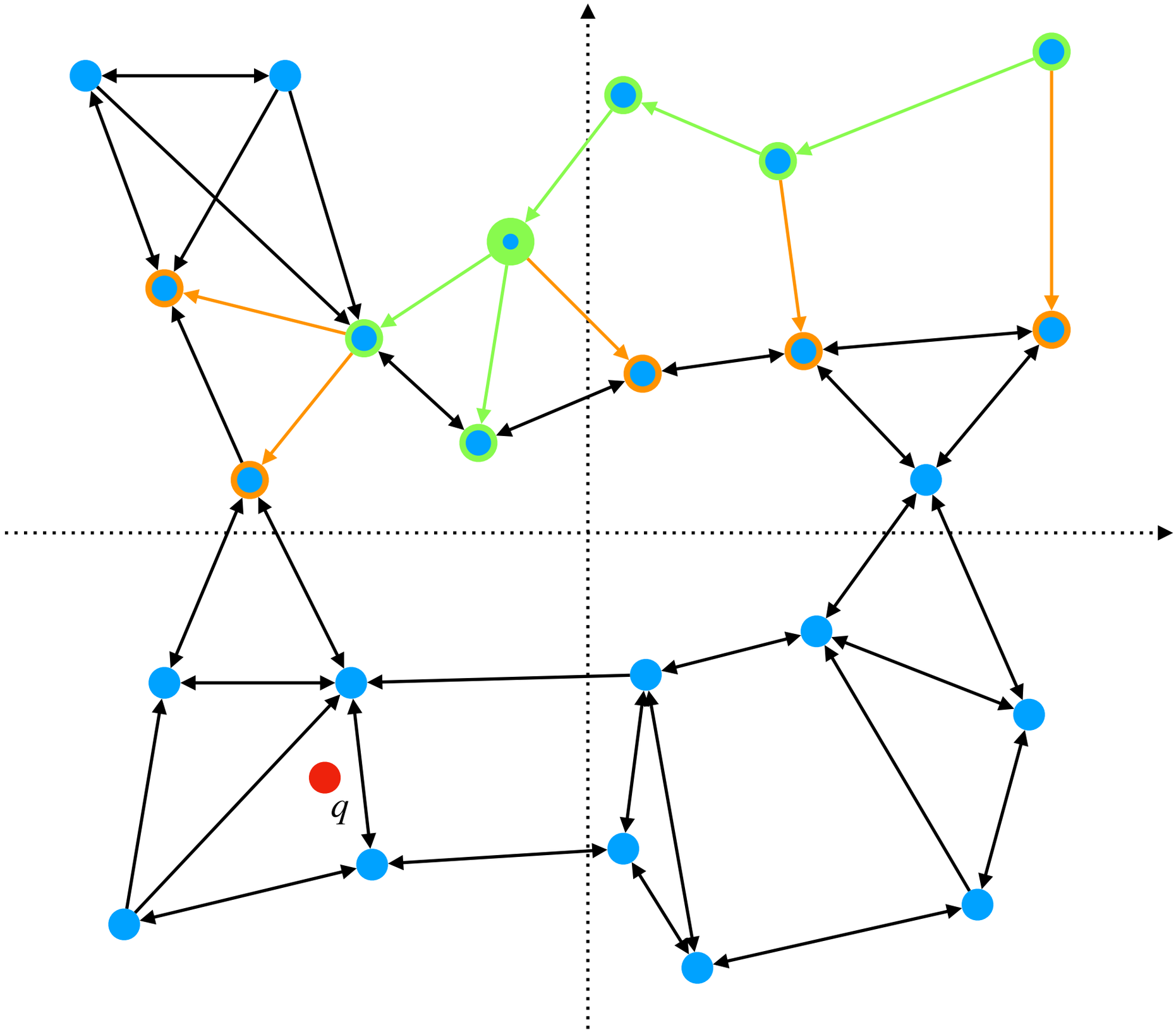

local min
q

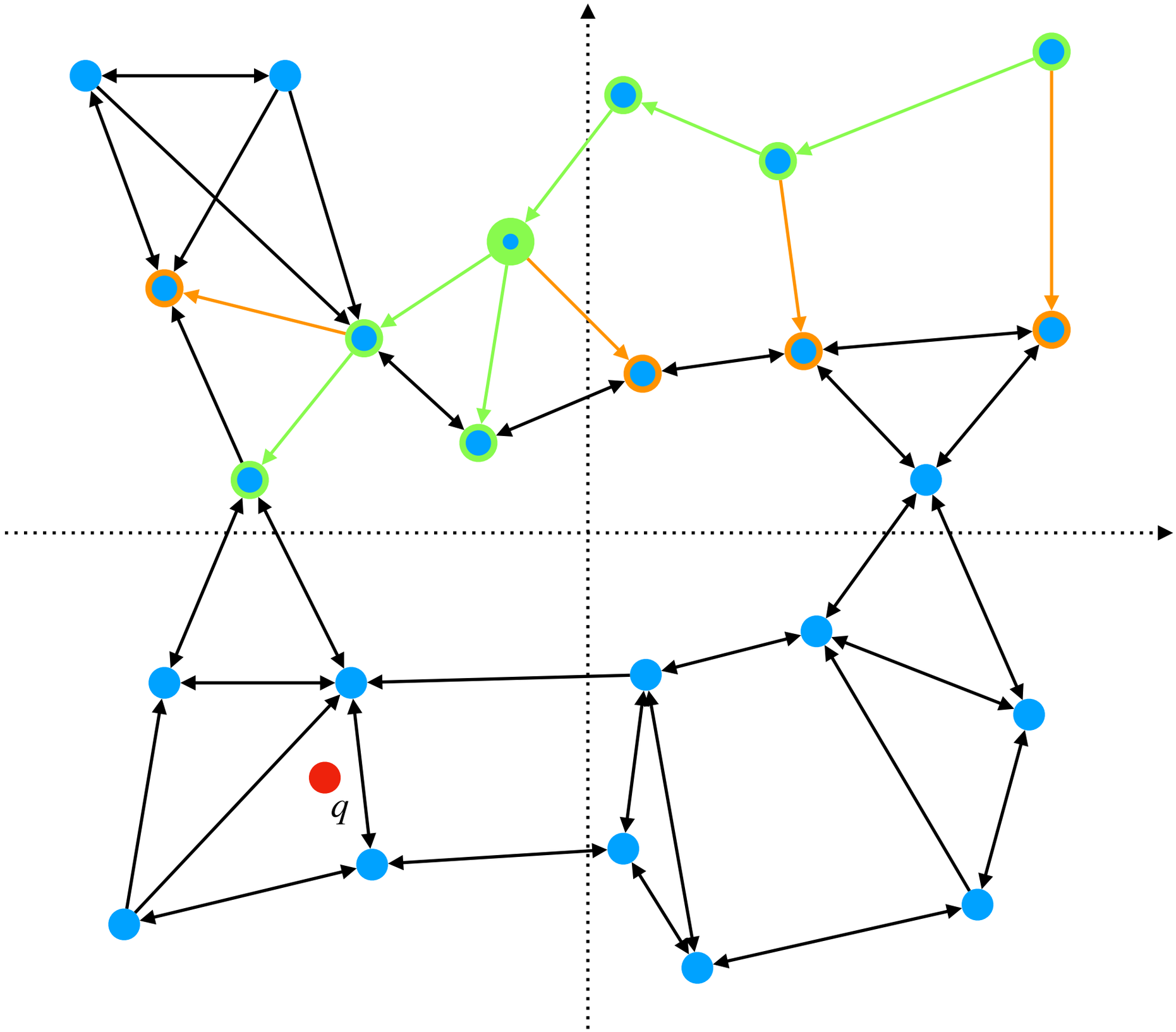

local min
q

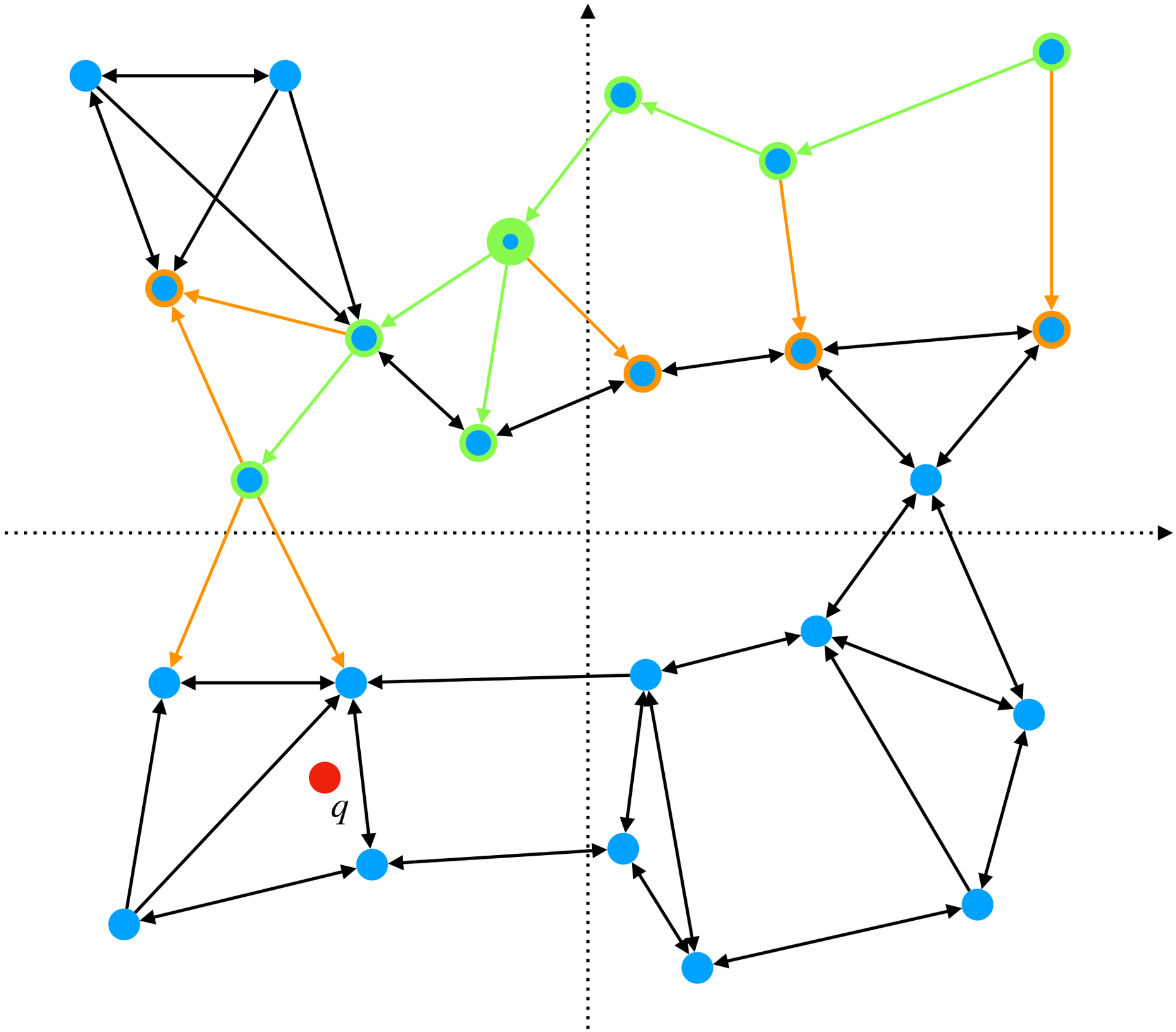

local min
q

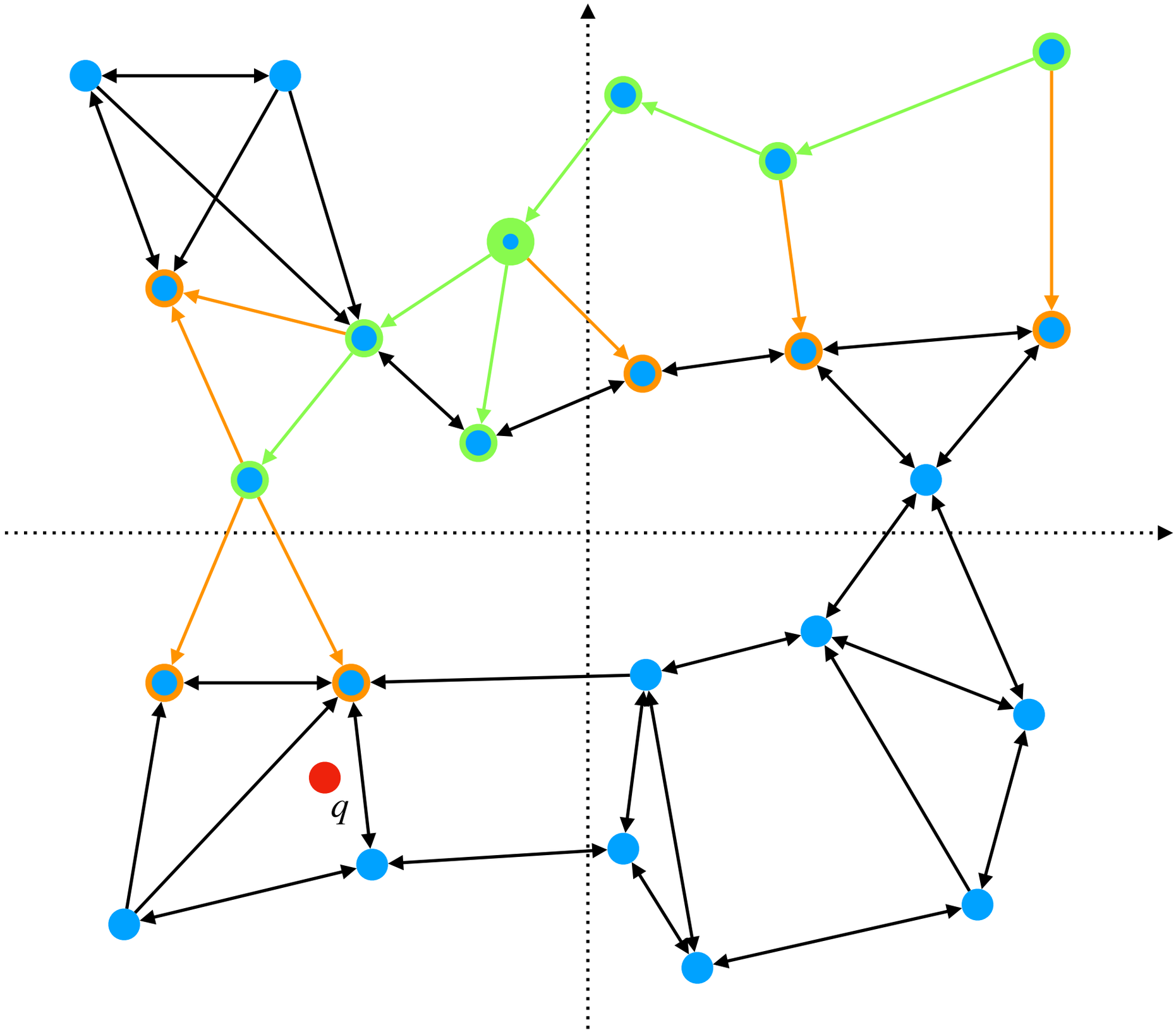

q

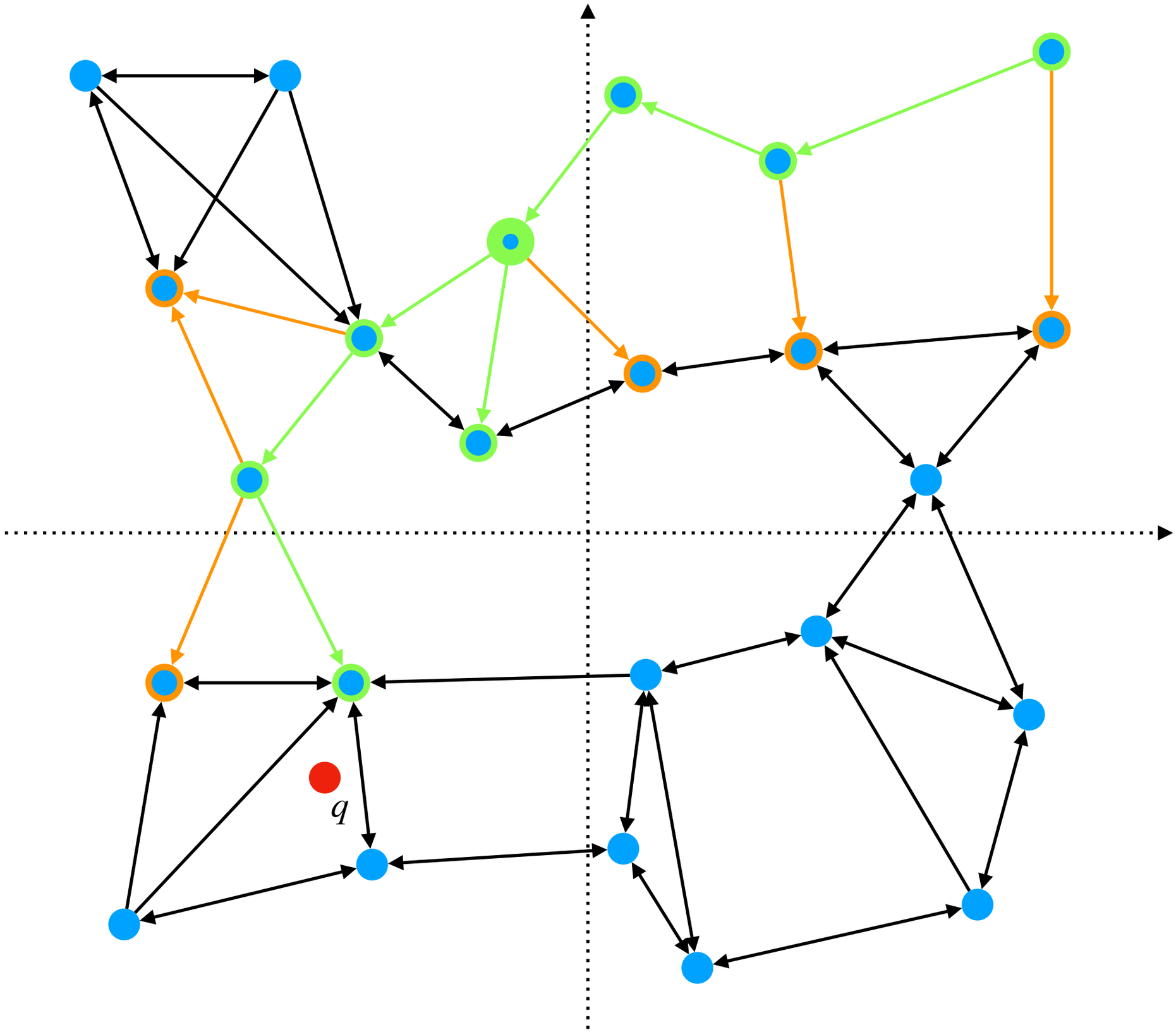

q

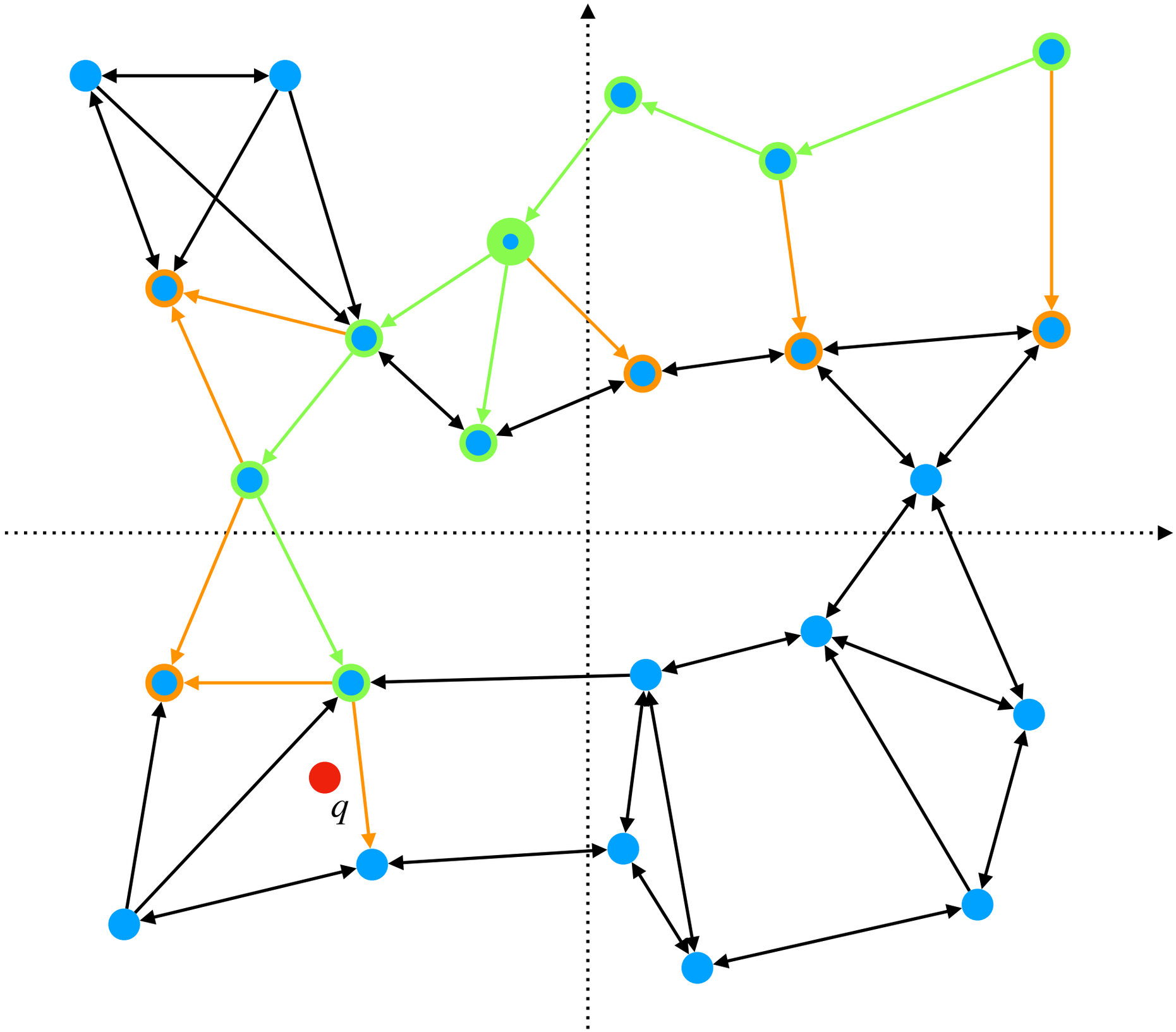

q

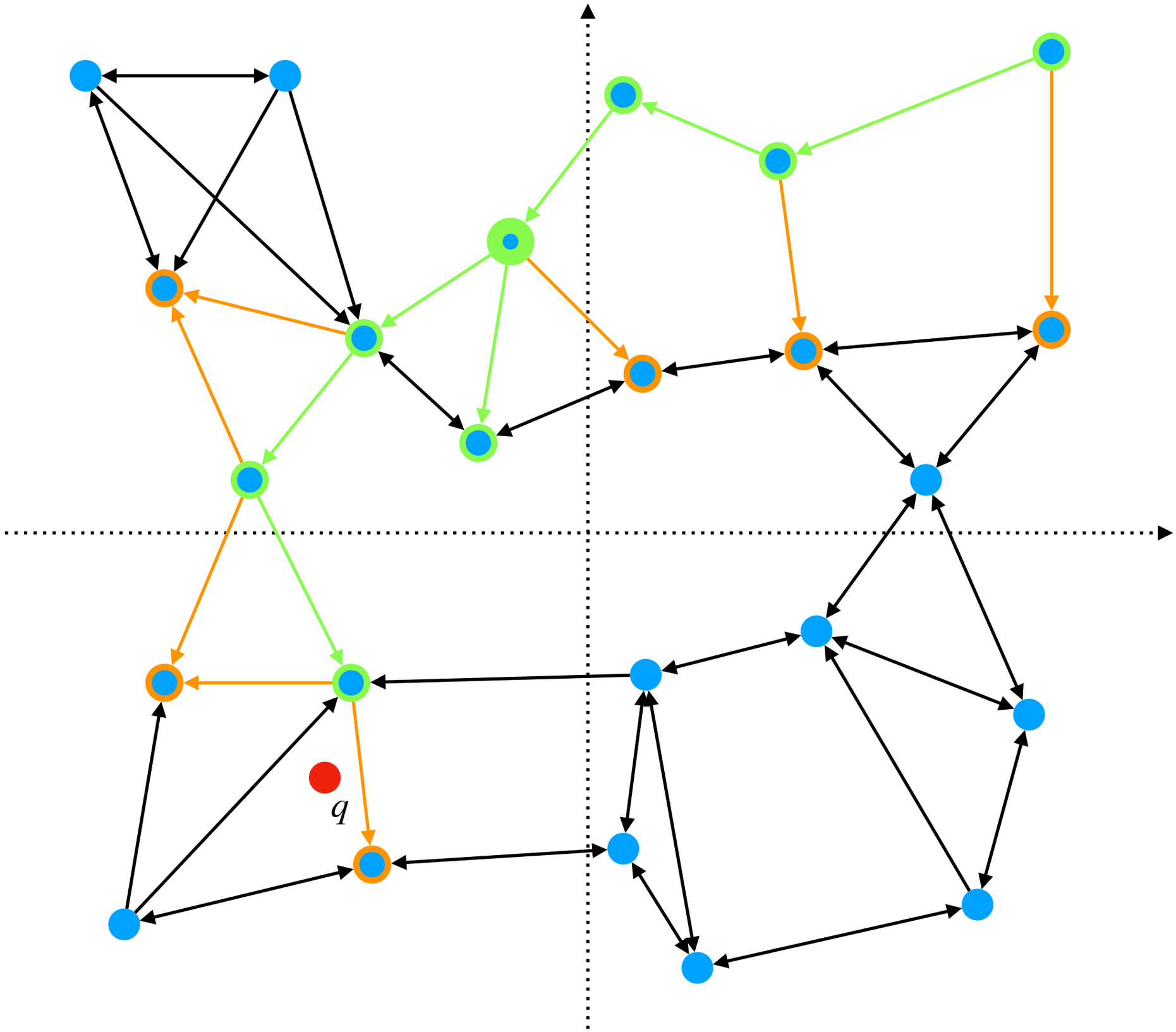

q

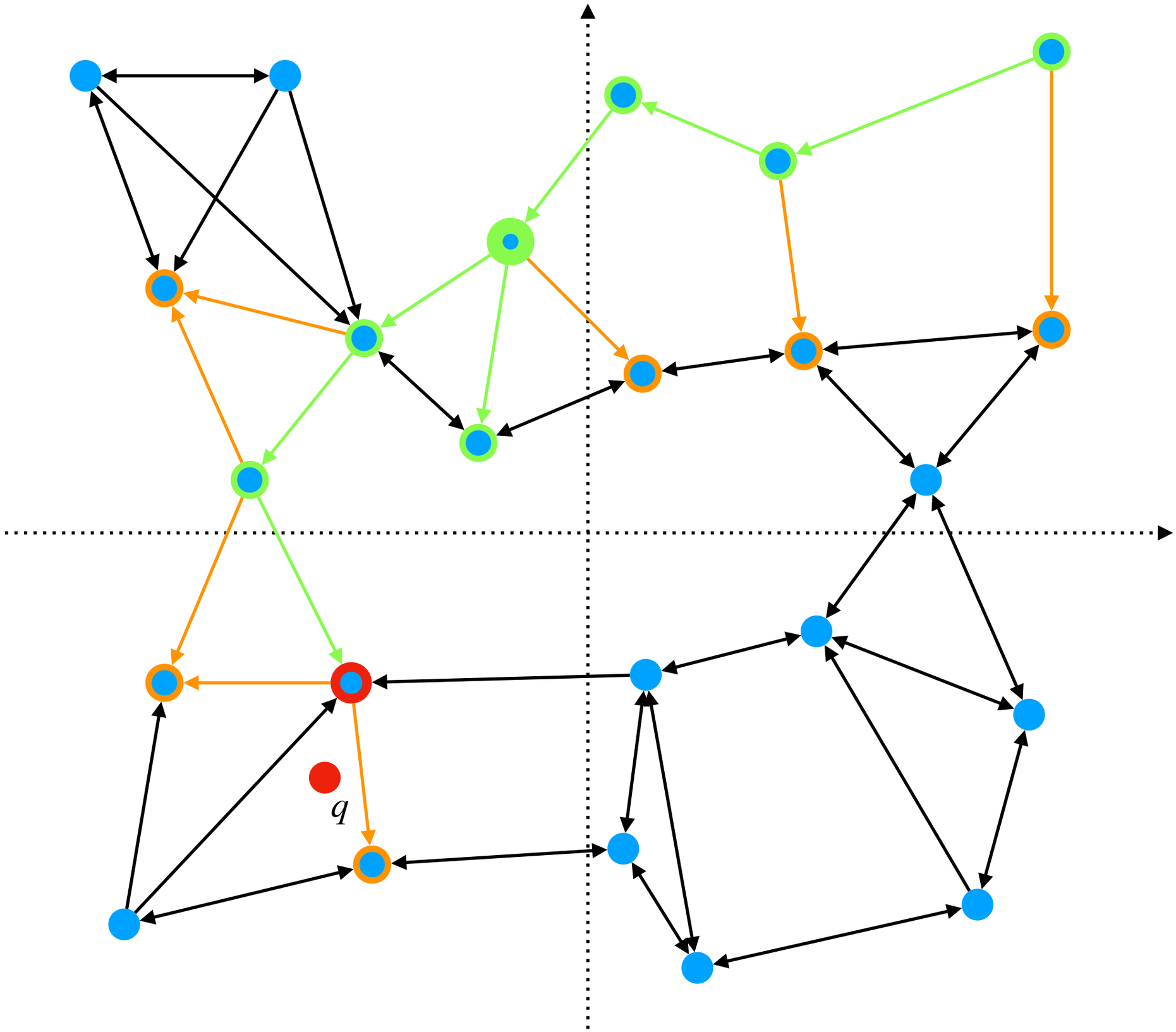

q


q


q


q


q


q

Neighbourhood Construction

KNNG

KNN-Graph

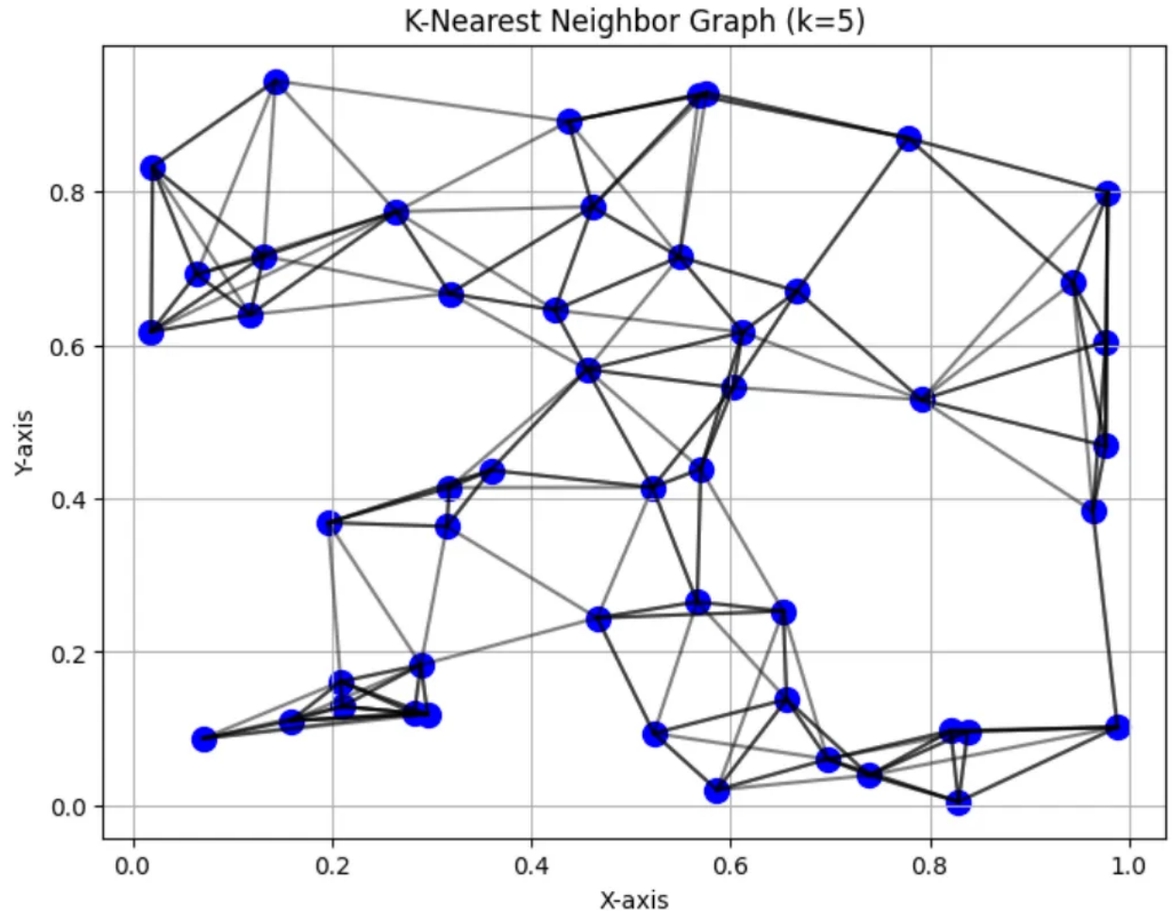
Let $P = \{p_1, p_2, \ldots, p_n\} \subset X$,

$\rho : X \times X \to \mathbb{R} \geq 0$,

$$\mathrm{NN}_K(i) = \underset{S \subseteq P \setminus \{p_i\},\ |S|=K}{\arg\min} \sum_{p \in S} \rho(p_i, p)$$

KNN-Graph is oriented graph $G = (V, E)$,
where:

- $V = P$
- $E = \{(p_i, p_j) \mid p_j \in \mathrm{NN}_K(i)\}$



Used in:

The KGraph directly uses the approximate K Nearest Neighbour Graph (KNNG) as the index

[Hajebi, Kiana, et al. "Fast approximate nearest-neighbor search with k-nearest neighbor graph." *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. No. 1. 2011.]

Efanna uses Kd-tree as navigation and continue search on approximate KNNG

[Fu, Cong, and Deng Cai. "Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph." *arXiv preprint arXiv:1609.07228* (2016)]

NSW uses as KNNG basic approximation of Delone Graph. At the same time incrementally KNNG emerge Small World properties.

[Пономаренко et al. "Структура со свойствами тесного мира для решения задачи поиска ближайшего соседа в метрическом пространстве." *Вестник Нижегородского университета им. НИ Лобачевского* 5-2 (2012): 409-415.]

Relative Neighbourhood Graph (RNG)

History Remark

Original Relative Neighbourhood Graph was developed not for ANNS.

“In computational perception we would like an algorithm to join pairs of points such that the final graph obtained is perceptually meaningful in some sense.”

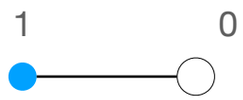
[Toussaint, Godfried T. "The relative neighbourhood graph of a finite planar set." *Pattern recognition* 12.4 (1980): 261-268.]

Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

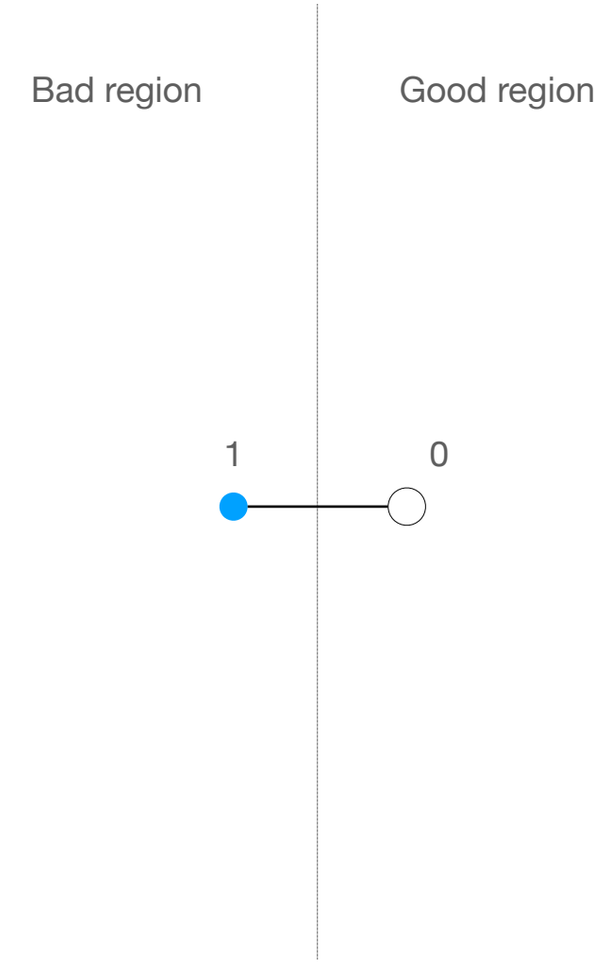


Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

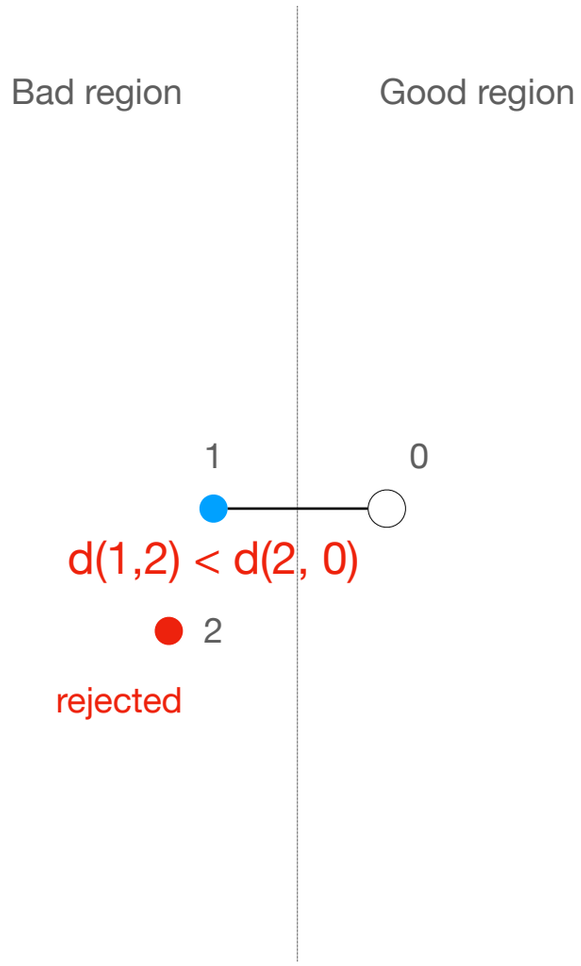


Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

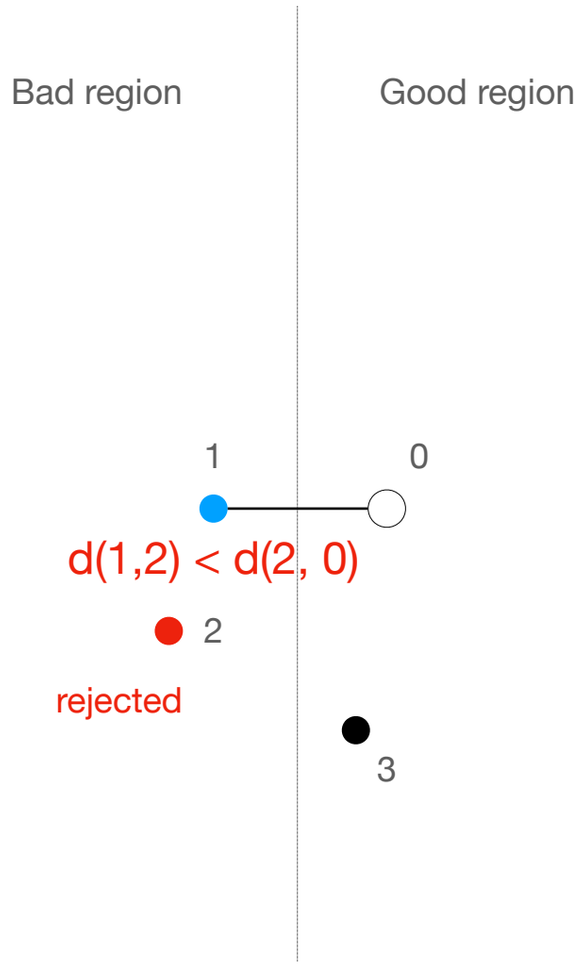


Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

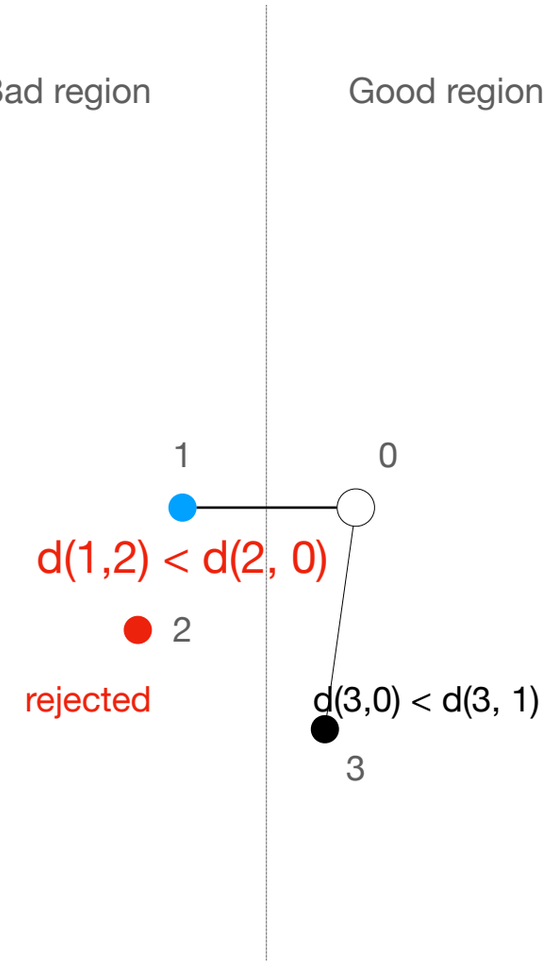


Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

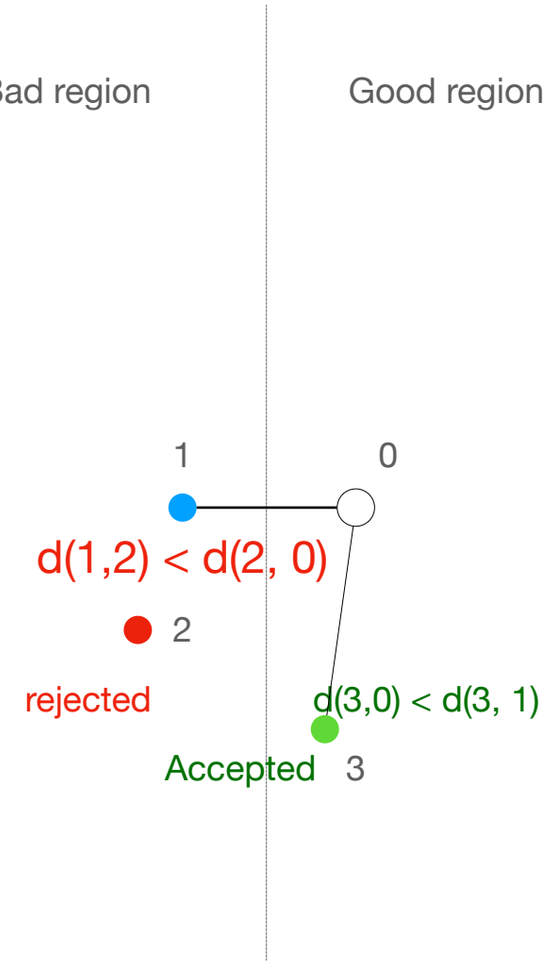


Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'



Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

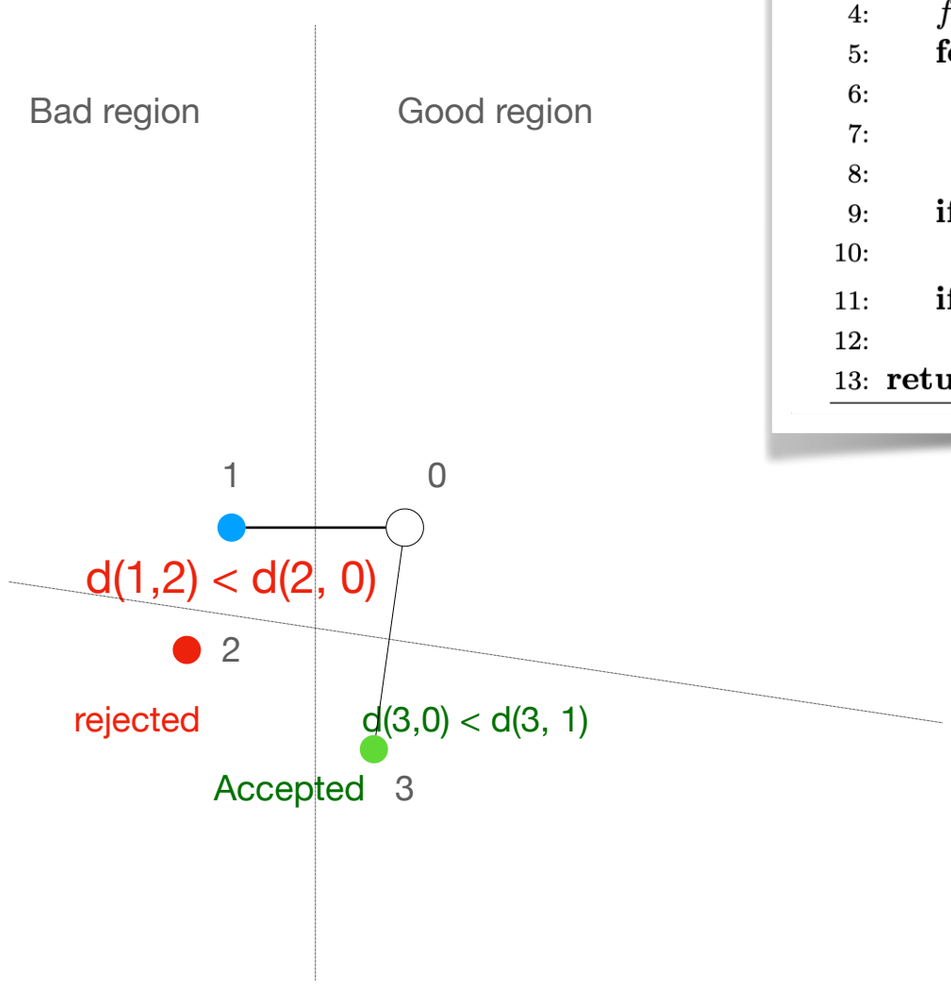
sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

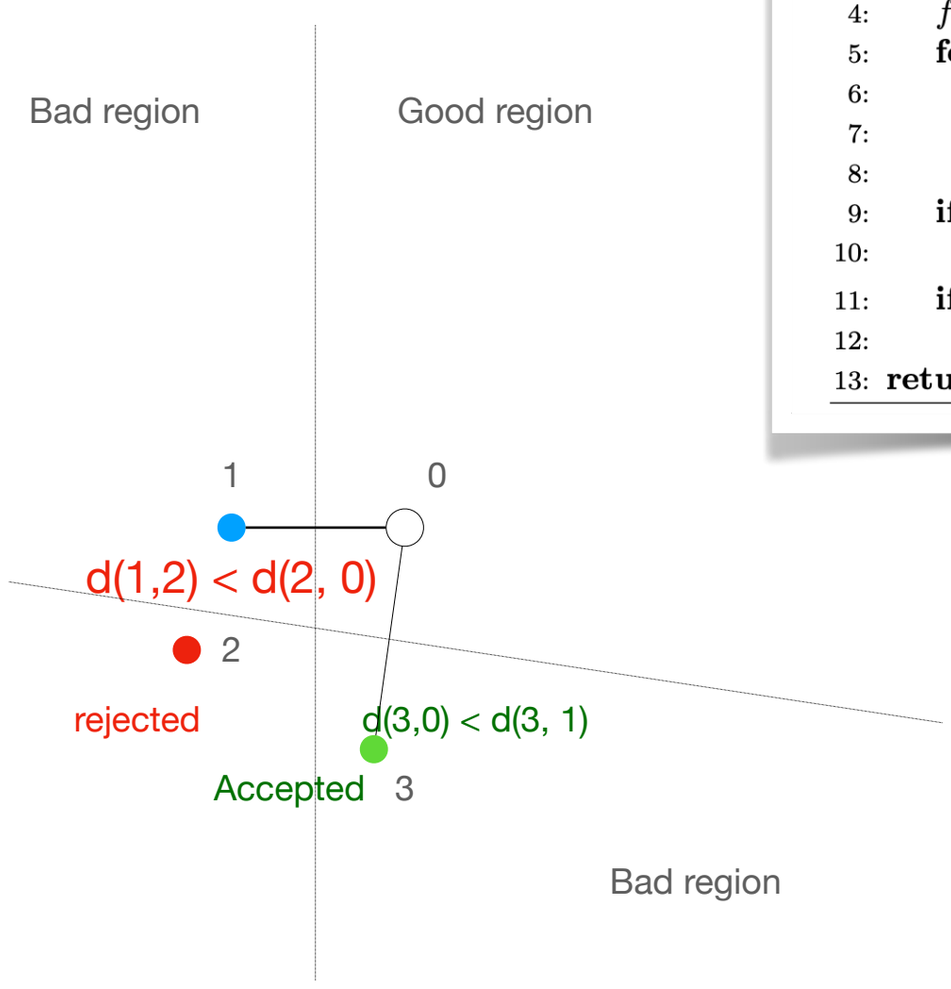
Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

```
sort v ∈ C in ascending order of ρ(v*, v)
C' ← ∅
for v ∈ U do
    f ← true
    for w ∈ C' do
        if ρ(v*, v) ≥ ρ(v, w) then
            f ← false
            break
    if f then
        C' ← C' ∪ {v}
    if |C'| ≥ m then
        break
return C'
```





Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

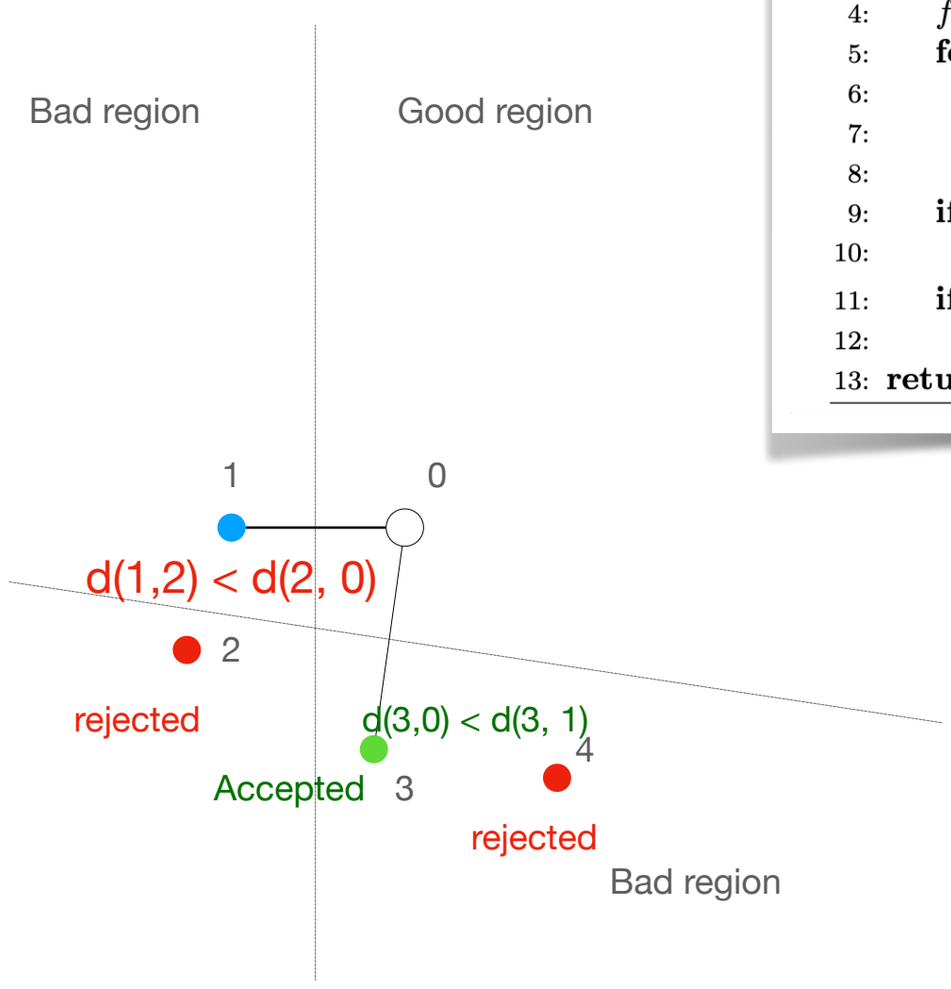
sort $v \in C$ in ascending order of $\rho(v^*, v)$
$C' \leftarrow \emptyset$
for $v \in U$ **do**
    $f \leftarrow$ true
    for $w \in C'$ **do**
        if $\rho(v^*, v) \geq \rho(v, w)$ **then**
            $f \leftarrow$ false
            break
    if f **then**
        $C' \leftarrow C' \cup \{v\}$
    if $|C'| \geq m$ **then**
        break
return C'

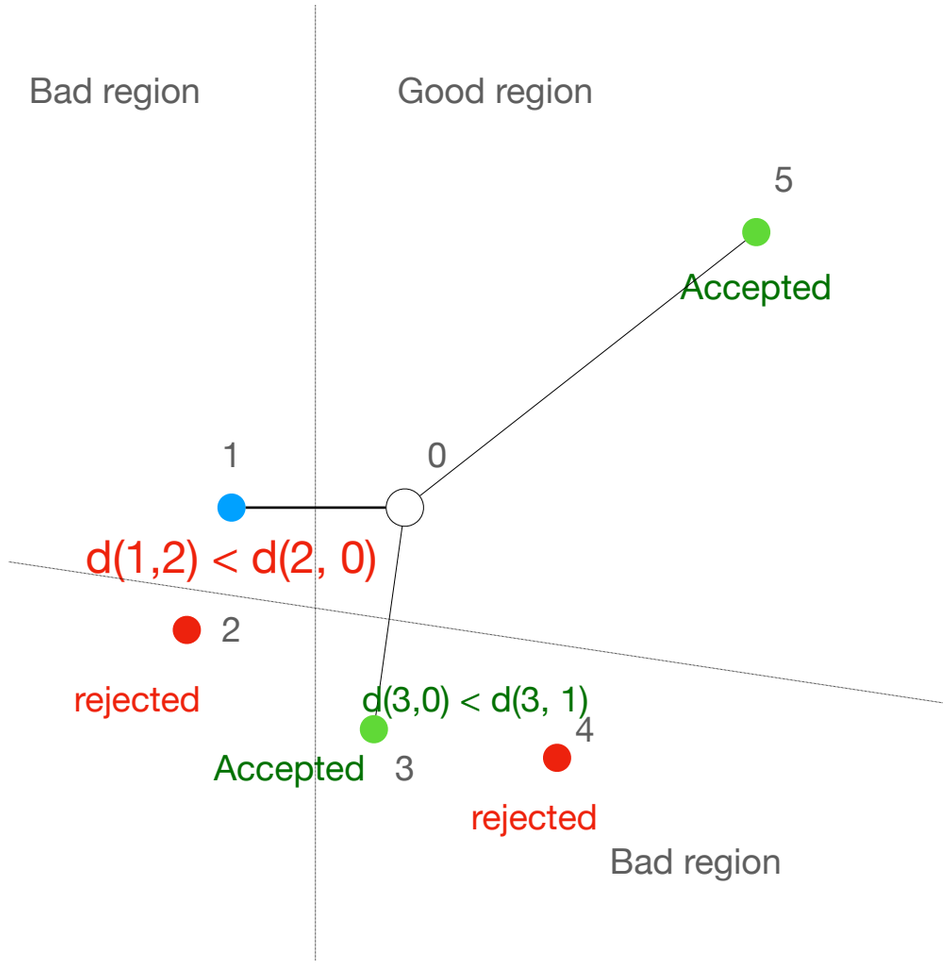
Algorithm 4 RNG-NEIGHBORHOOD-CONSTRUCTION(v^*, C, m)

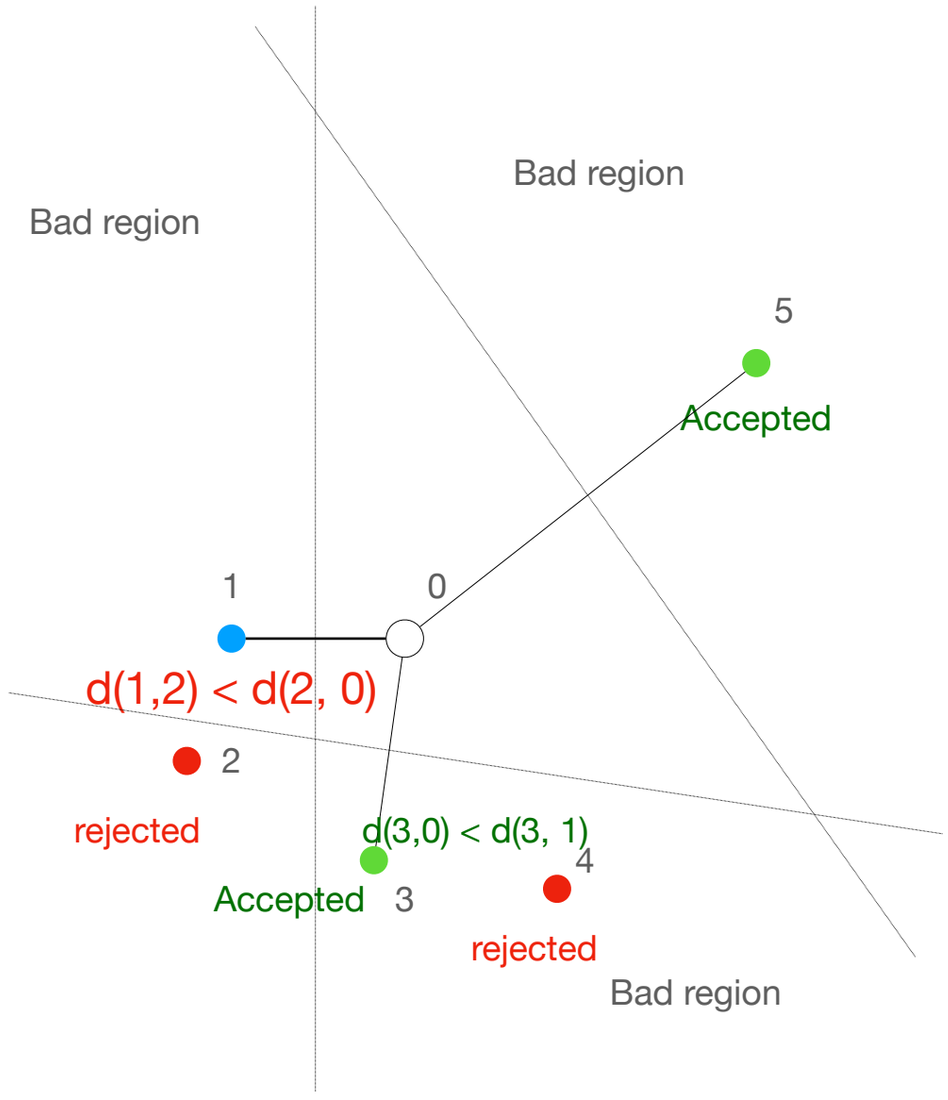
Require: vertex $v^* \in V$, neighbor candidates $C \subseteq V$, maximum neighborhood size $m \in \mathbb{N}$

Ensure: selected neighbors $C' \subseteq C$

```
sort v ∈ C in ascending order of ρ(v*, v)
C' ← ∅
for v ∈ U do
    f ← true
    for w ∈ C' do
        if ρ(v*, v) ≥ ρ(v, w) then
            f ← false
            break
    if f then
        C' ← C' ∪ {v}
    if |C'| ≥ m then
        break
return C'
```




Bad region
Good region
5
Accepted
1
0
d(1,2) < d(2, 0)
2
rejected
d(3,0) < d(3, 1)
4
Accepted
3
rejected
Bad region


Bad region
Bad region
5
Accepted
1
0
d(1,2) < d(2, 0)
2
rejected
d(3,0) < d(3, 1)
4
Accepted
3
rejected
Bad region

MSNET

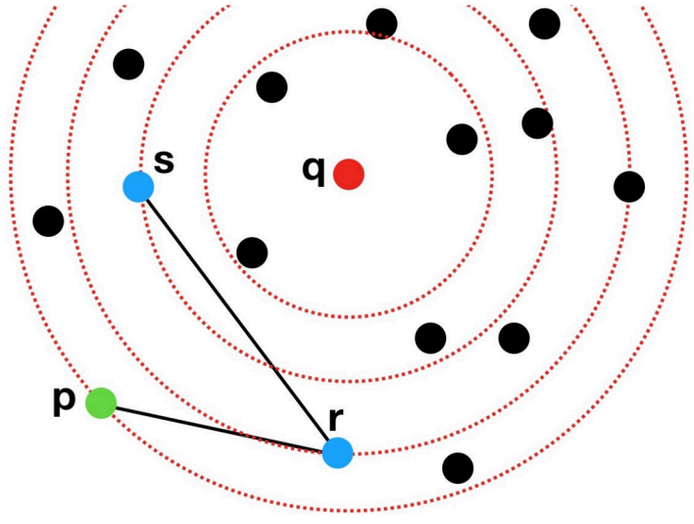
Monotonic Search Network (MSNET)

Definition:

Given a finite point $|X \subset \mathbb{R}^d| = n$, and $p, q \in X$ and $G(X, E)$ denotes a graph defined on X. Let $v_1, v_2, \ldots, v_k, (v_1 = p, v_k = q)$ denote a path from p to q in G, i.e., $\forall i = 1,2,...,k-1, (v_i, v_{i+1}) \in E$. This path is a monotonic path if and only if $\forall i = 1,...,k-1, \rho(v_i, q) > \rho(v_{i+1}, q)$

Definition:

Given a finite point set $|X \subset \mathbb{R}^d| = n$, a graph defined on X is a monotonic search network if and only if there exists at least one monotonic path from p to q for any two nodes $p, q \in X$.



[D. Dearholt, N. Gonzales, and G. Kurup. Monotonic search networks for computer vision databases. Signals, Systems and Computers, 1988. Twenty-Second Asilomar Conference on, 2:548–553, 1988]

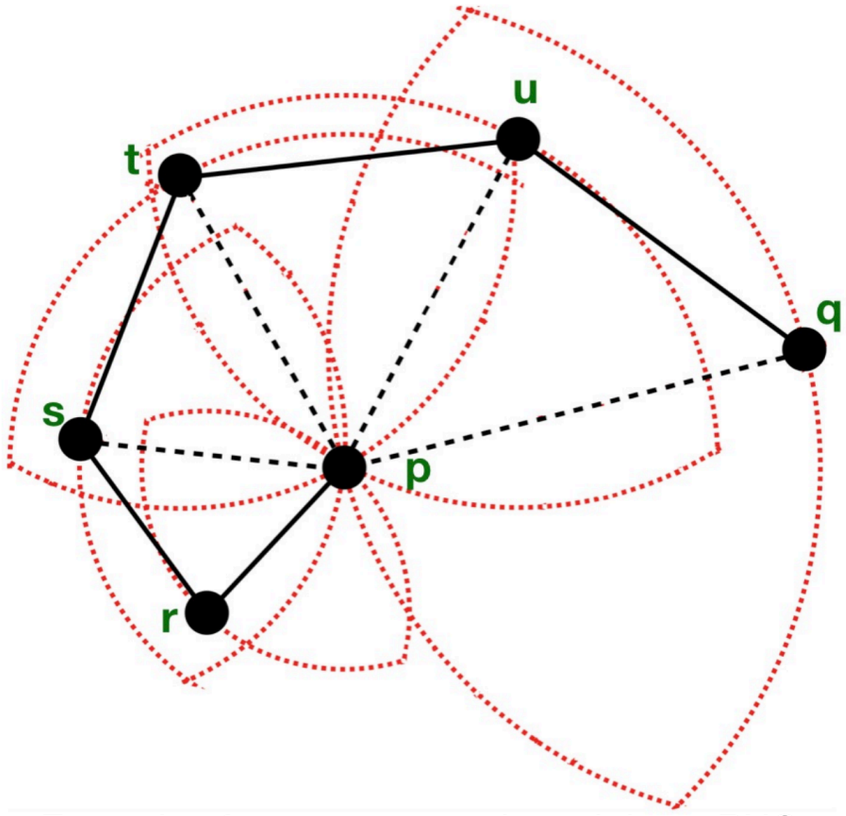
RNG is not MSNET

$$lune_{pq} = B(p, \rho(p,q)) \cap B(q, \rho(p,q))$$

Remark:

For any two nodes $p, q \in X$, $(p,q) \in$ RNG if and only if $liune_{pq} \cap X = \varnothing$

[J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. Proceedings of the IEEE, 80(9):1502–1517, 1992.]



Example of a non-monotonic path in an RNG

https://arxiv.org/pdf/1707.00143

MRNG

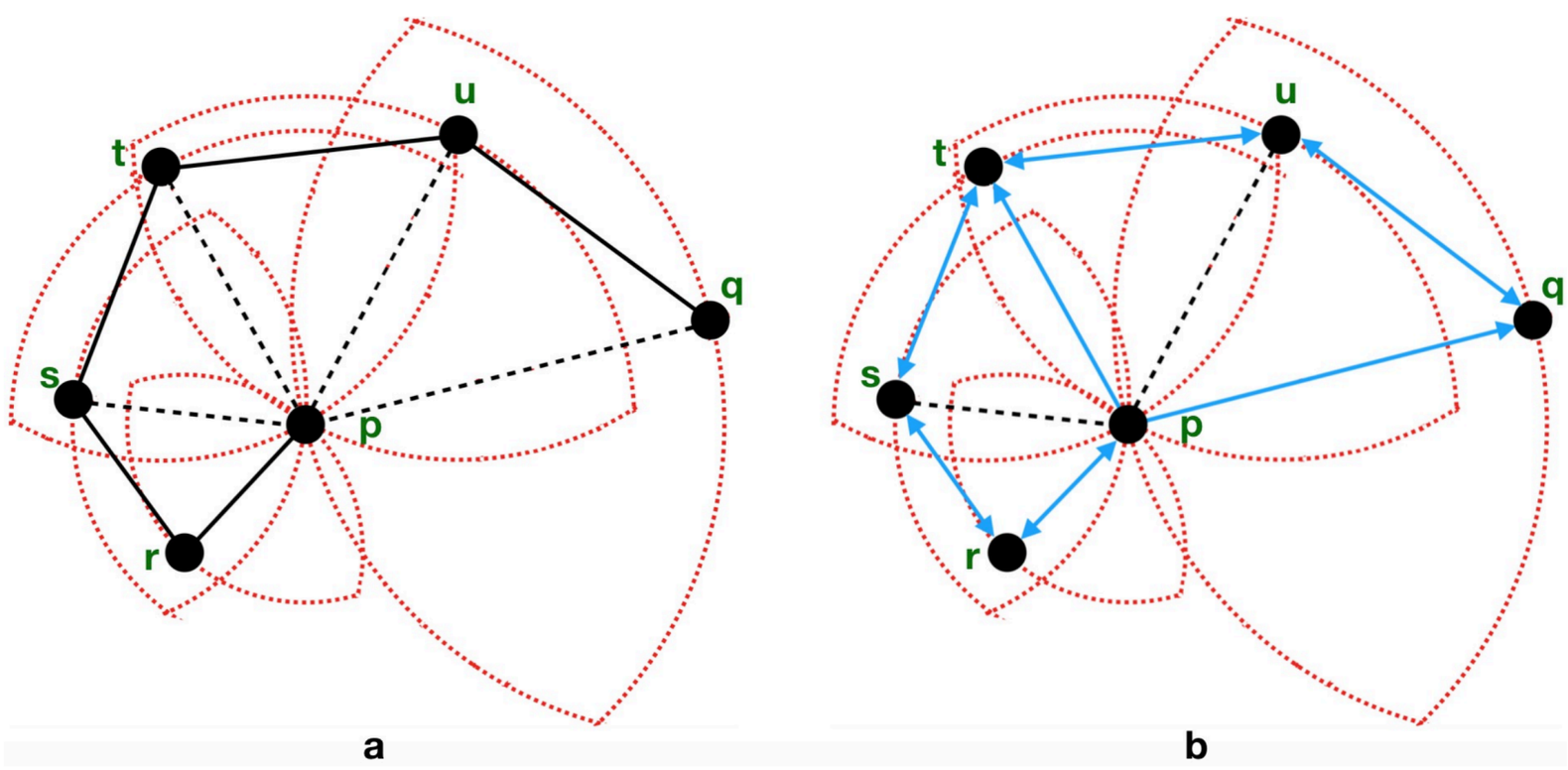
Monotonic Relative Neighbourhood Graph (MRNG)

Definition MRNG

Given a finite point $|X \subset \mathbb{R}^d| = n$, an MRNG is a directed graph with the set of edges satisfying the following property: for any edge (p, q),

$(p, q) \in$ MRNG if and only if $lune_{pq} \cap X = \varnothing$ or

$\forall r \in (lune_{pq} \cap X), (p, r) \notin$ MRNG



RNG(a) vs MRNG (b)

https://arxiv.org/pdf/1707.00143

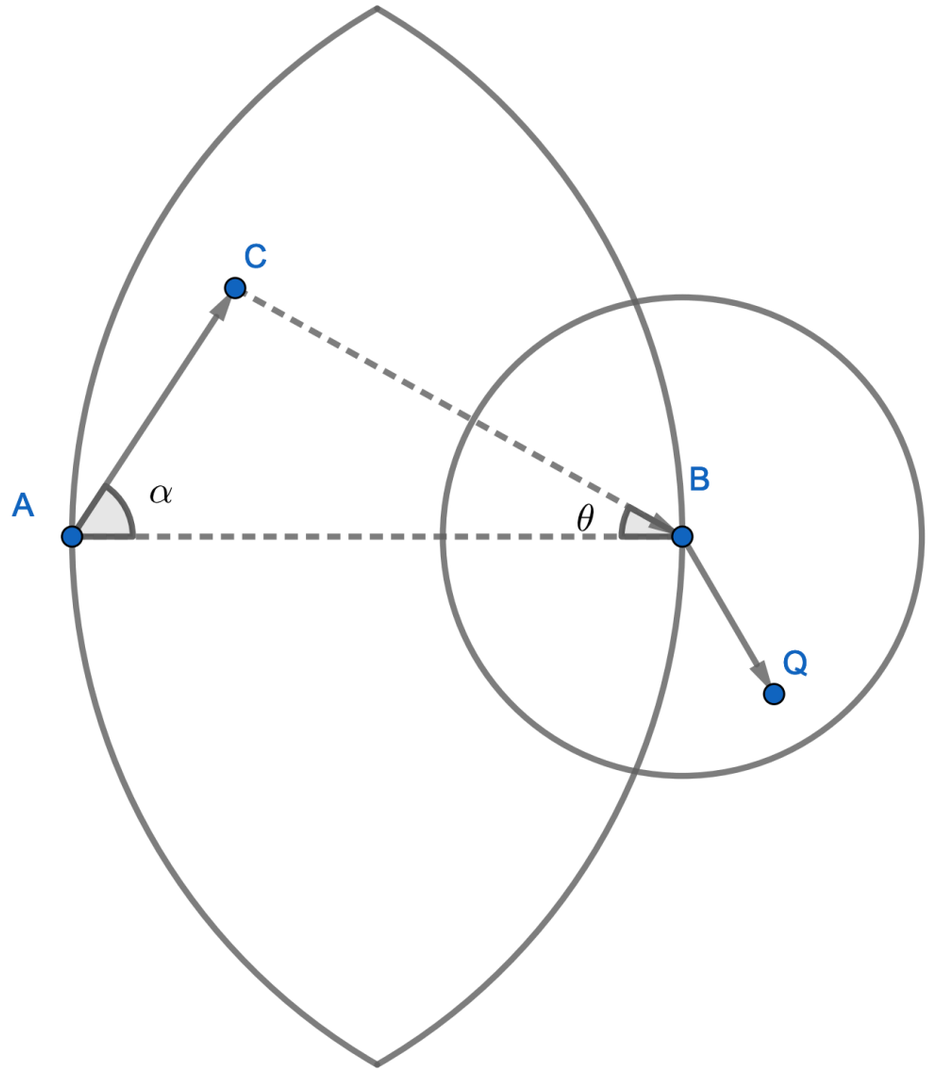
Theorem

Given a finite point $|X \subset \mathbb{R}^d| = n$, an MRNG defined on X is MSNET

https://arxiv.org/pdf/1707.00143

(r,p)-MNSET

Example where RNG fails



Point A is the current node, where point C is one of existing neighbours, point B is the candidate neighbour and point Q is the query point. Also, point B is the nearest neighbour of Q.

[Fan, Xiaobin, et al. "Tree-based search graph for approximate nearest neighbor search." *arXiv preprint arXiv:2201.03237* (2022).]

Definition (r, p)**-MSNET**

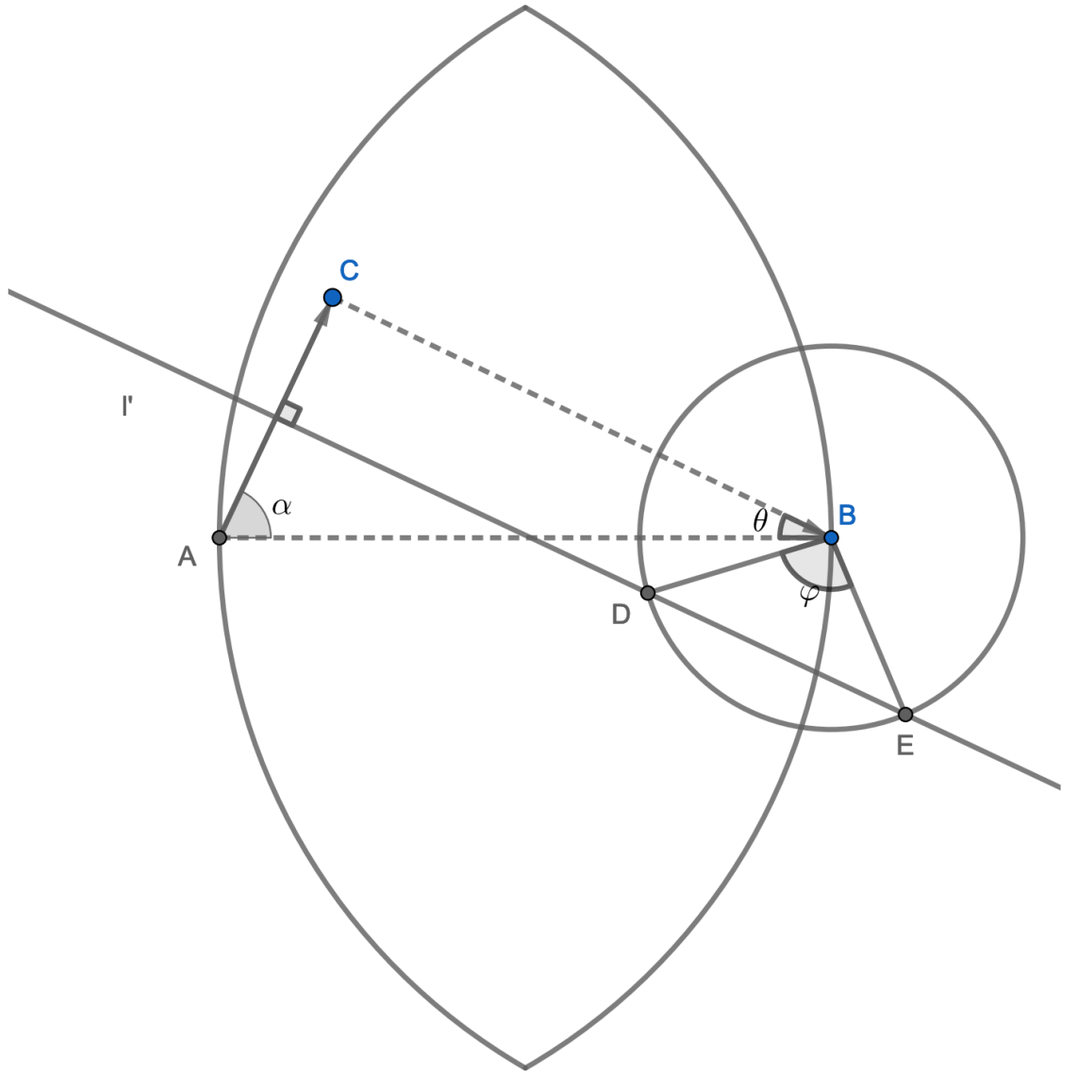
For a given query point $q \in \mathbb{R}^d$, if $\rho(q, o^*) \leq r$ where o^* is the nearest neighbour of q, a graph is a (r, p)**-MSNET** when the probability of achieving monotonic search is greater than p.

[Fan, Xiaobin, et al. "Tree-based search graph for approximate nearest neighbor search." *arXiv preprint arXiv:2201.03237* (2022).]

(r, p)-MSNET

$$p = \frac{\int_{-r\cos\frac{\varphi}{2}}^{r} \int_{-\sqrt{r^2-x_1^2}}^{\sqrt{r^2-x_1^2}} \cdots \int_{-\sqrt{r^2-x_1^2-x_2^2-\ldots-x_{d-1}^2}}^{\sqrt{r^2-x_1^2-x_2^2-\ldots-x_{d-1}^2}} dx_d}{\int_{-r}^{r} \int_{-\sqrt{r^2-x_1^2}}^{\sqrt{r^2-x_1^2}} \cdots \int_{-\sqrt{r^2-x_1^2-x_2^2-\ldots-x_{d-1}^2}}^{\sqrt{r^2-x_1^2-x_2^2-\ldots-x_{d-1}^2}} dx_d}$$

$$p \geq 1 - \frac{\varphi}{2\pi} = 1 - \frac{\arccos\left(\frac{l}{r}\frac{\sin(2\alpha+\theta)}{2\sin(\alpha+\theta)}\right)}{\pi}$$



Point A is the current node, where point C is one of existing neighbours, point B is the candidate of point A. Point B is the nearest neighbour of the point Q.

[Fan, Xiaobin, et al. "Tree-based search graph for approximate nearest neighbor search." *arXiv preprint arXiv:2201.03237* (2022).]

TBSG

Approximation of (r, p)-MNSET

TBSG (Tree-based Search Graph)

TBSG = Cover Tree + Bi-directed KNNG + (r, p)-MSNET based selection strategy

[Fan, Xiaobin, et al. "Tree-based search graph for approximate nearest neighbor search." *arXiv preprint arXiv:2201.03237* (2022).]

Algorithm 2 NeighborSelection(s, E, m, mp, r)

Require: node s selecting neighbors, neighbor candidate set E, maximum of neighbor size m, threshold of min_prob mp, radius r

Ensure: selected neighbor set V

```
V ← φ
sort E in the ascending order of distance to s
for all node e in E do
    if V.size() == m then break
    exclude ← false
    for all node v in V do
        if δ(v, e) < δ(s, e) then
            m_prob ← calculate the min_prob with edge sv⃗
            if m_prob ≥ mp then
                exclude ← true
                break
    if not exclude then V.add(e)
return V
```

Algorithm 3 TBSG_Construction(CT, KG, m, mp, r, n)

Require: Cover Tree CT, KNNG KG, maximum of neighbor size m, threshold of min_prob mp, radius r, dataset cardinality n

Ensure: TBSG G with enter point ep

```
ep ← root of CT
G ←Graph with n nodes and no edges
BG ← add reversed edges to KG
for all node s in G do
    E ← neighbors of s in BG
    E.add(s.children()) in CT
    V ←NeighborSelection(s, E, m, mp, r)
    set the neighbors of s in G as V
return G
```

DPG

Diversified Proximity Graph

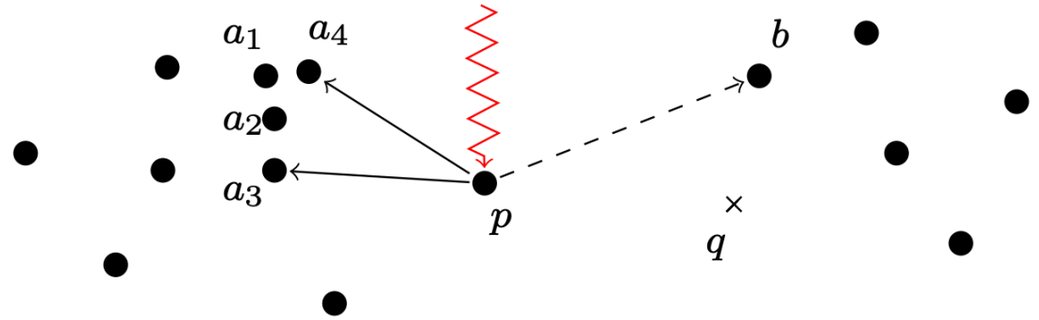
Diversified Proximity Graph (DPG)

Diversified Proximity Graph selects neighbours by minimizing average pairwise angle from KNNG and adds reverse edges.

$$N_i = S = \arg \min_{S \subseteq L, |S| = \kappa} \sum_{x_i, x_j \in S} \theta(x_i, x_j), \text{ where } L \text{ is KNN}(x_i)$$

This problem is NP-hard

[Kuo, Ching-Chung, Fred Glover, and Krishna S. Dhir. "Analyzing and modeling the maximum diversity problem by zero-one programming." *Decision Sciences* 24.6 (1993): 1171-1185.]



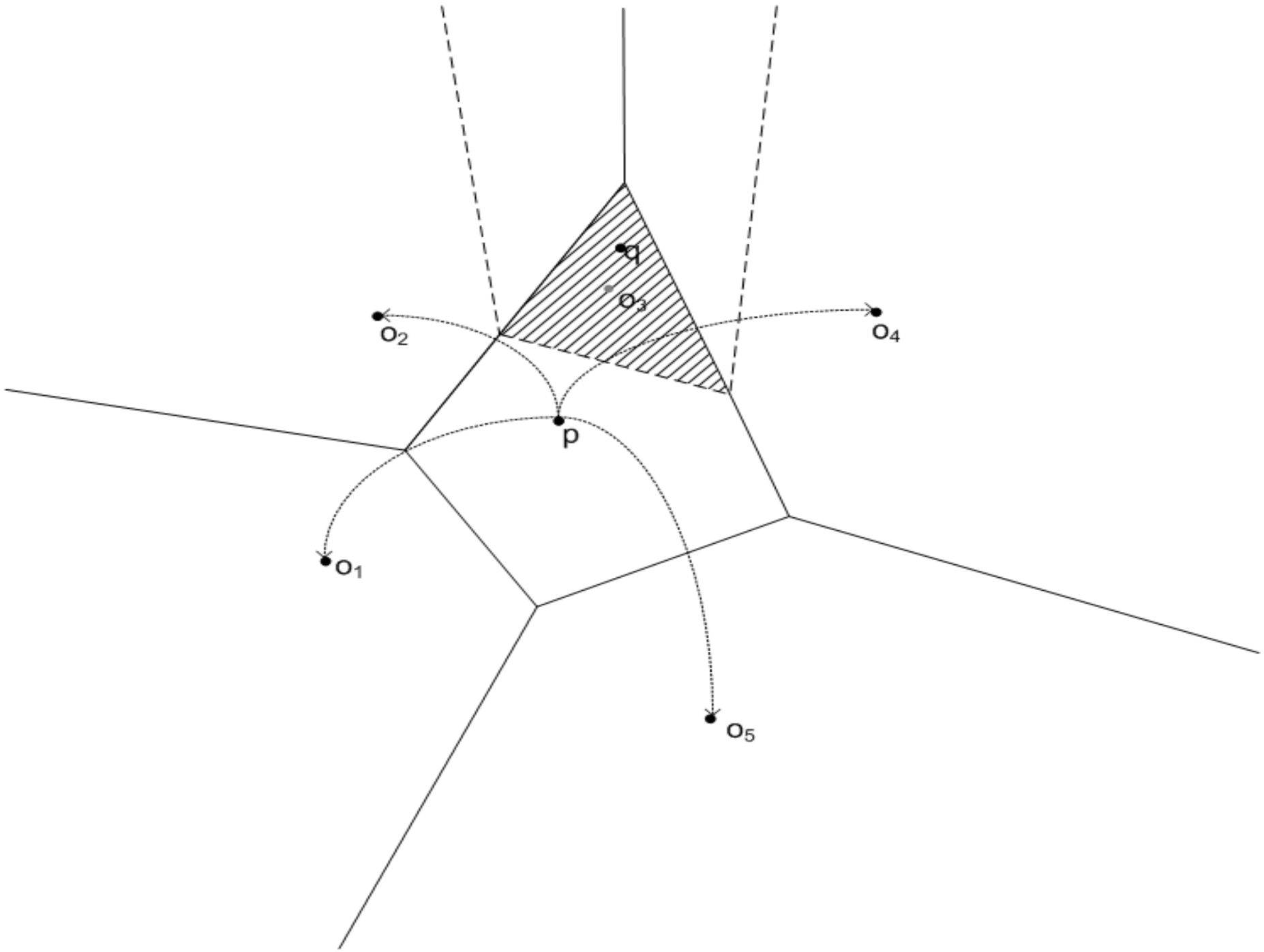
Greedy Heuristic:

1. Initially, S is set to the closest point of p in L. In each of the following κ−1 iterations,
2. a point is moved from L \ S to S so that the average pairwise angular similarity of the points in S is minimized.
3. each of the following K−1 iterations, a point is moved from L \ S to S so that the average pairwise angular similarity of the points in S is minimized. Then for each data point u in S, we include both edges (p, u) and (u, p) in the diversified proximity graph

[Li, Wen, et al. "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement." *IEEE Transactions on Knowledge and Data Engineering* 32.8 (2019): 1475-1488.]

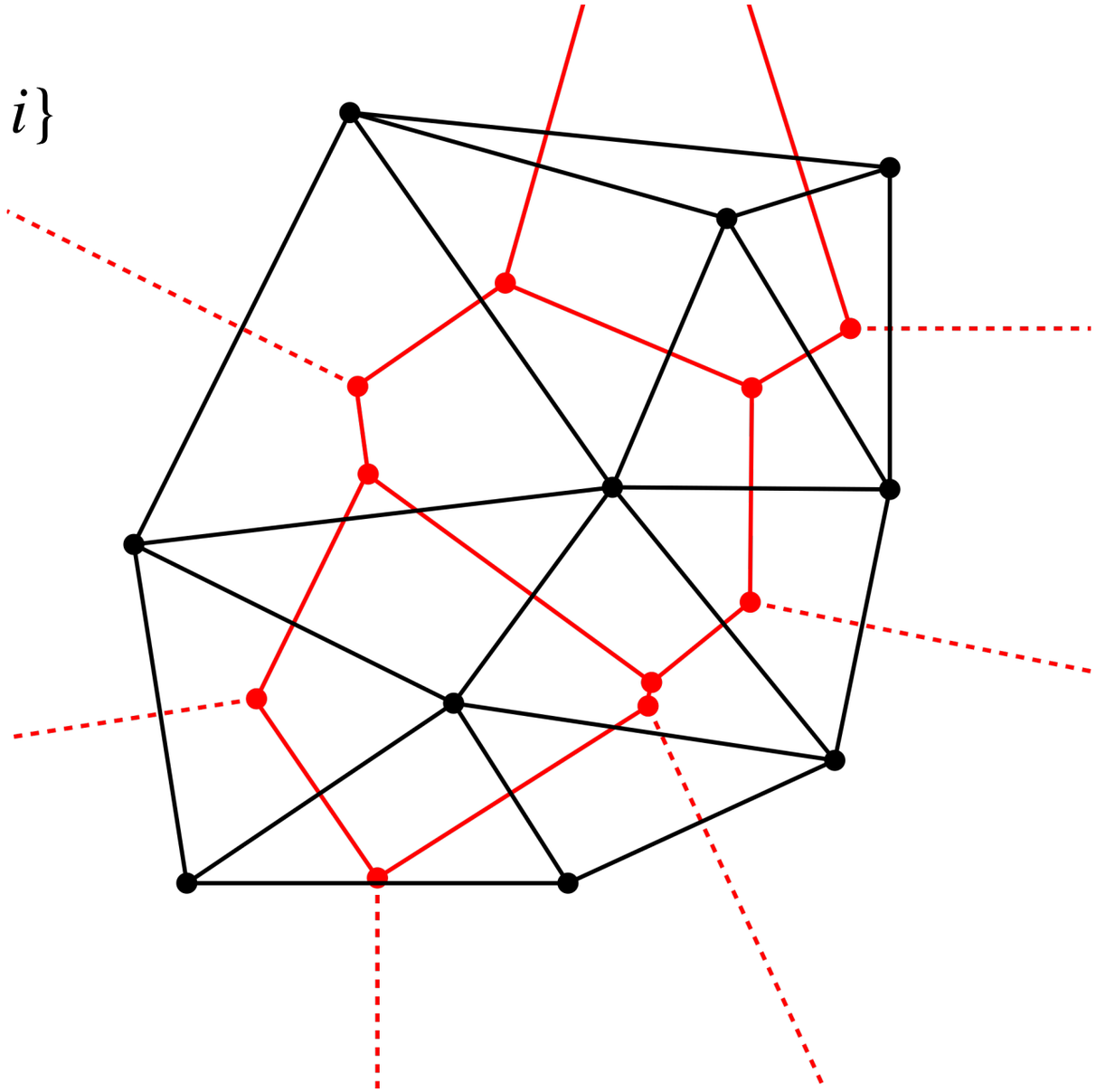
Graph Delone

How to construct a graph without local minimums?


q
o3
o2
o4
p
o1
o5

Let $\mathbb{R}^d$ – set of all possible objects, and
$\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \geq 0$ – distance function,
$X = \{x_i \in \mathbb{R}^d\}_{i=1}^n$ – k points.

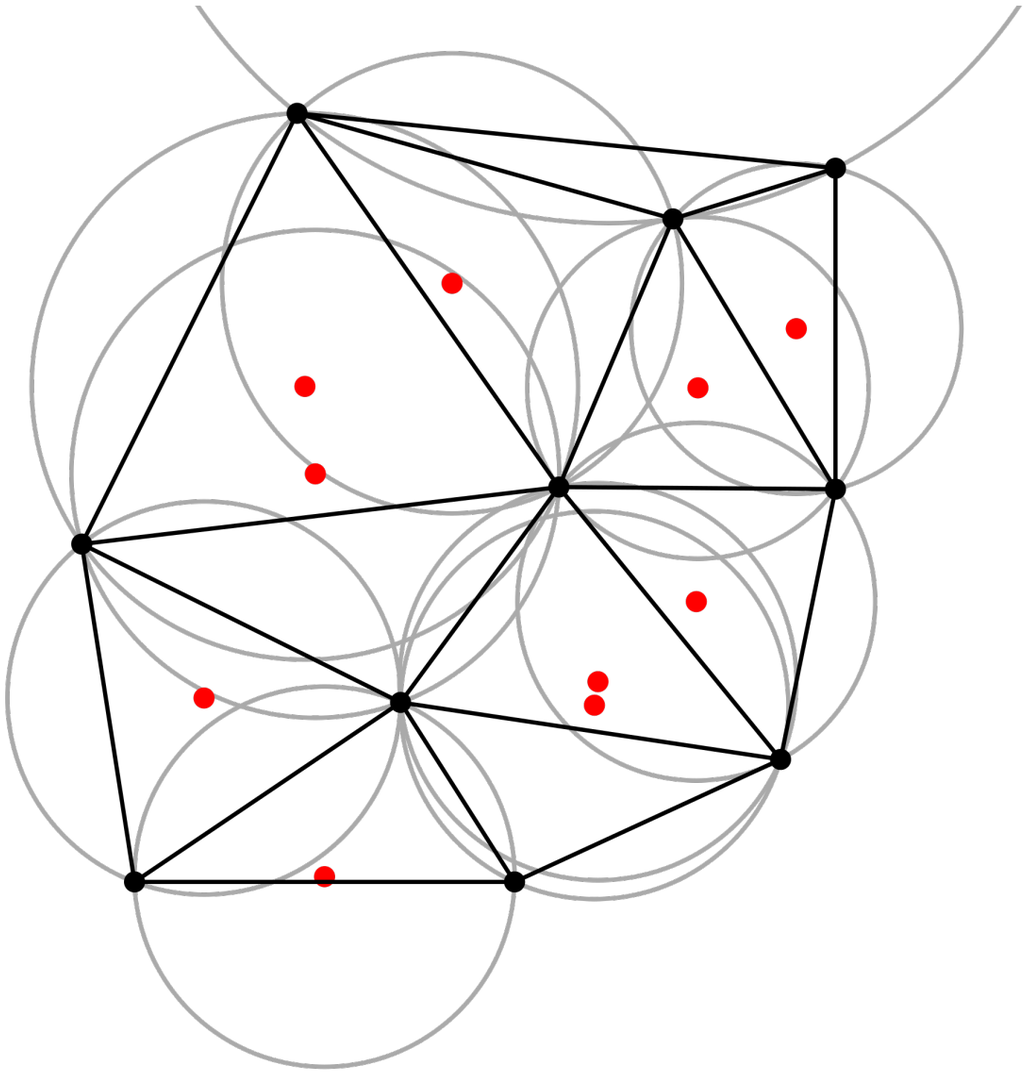
Voronoi cell:
$R_i = \{x \in \mathbb{R}^d \mid \rho(x, x_i) \leq \rho(x, x_j) \text{ for all } j \neq i\}$
Voronoi diagram is $\{R_i\}_{i=1}^n$



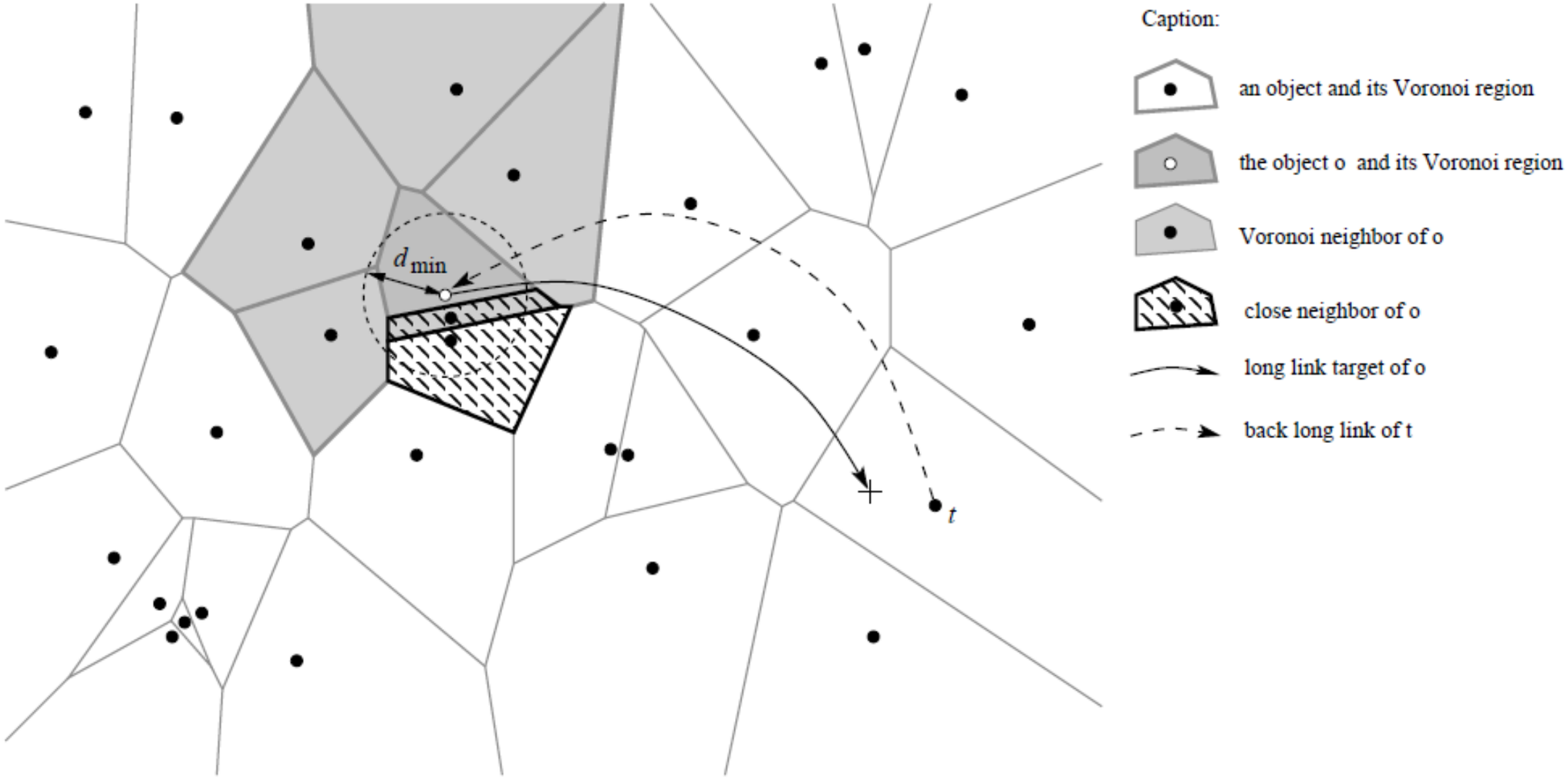
Delaunay Graph $G_D(V) = (V, E_D(V))$ of the vertices V is an undirected embedded graph defined by

$$\{u, v\} \in E_D(V) \iff u \neq v \wedge \exists C = B(x, r) : C \cap V = \{u, v\}$$

i.e., u and v are connected, if and only if there is a disk containing only these two points of V.



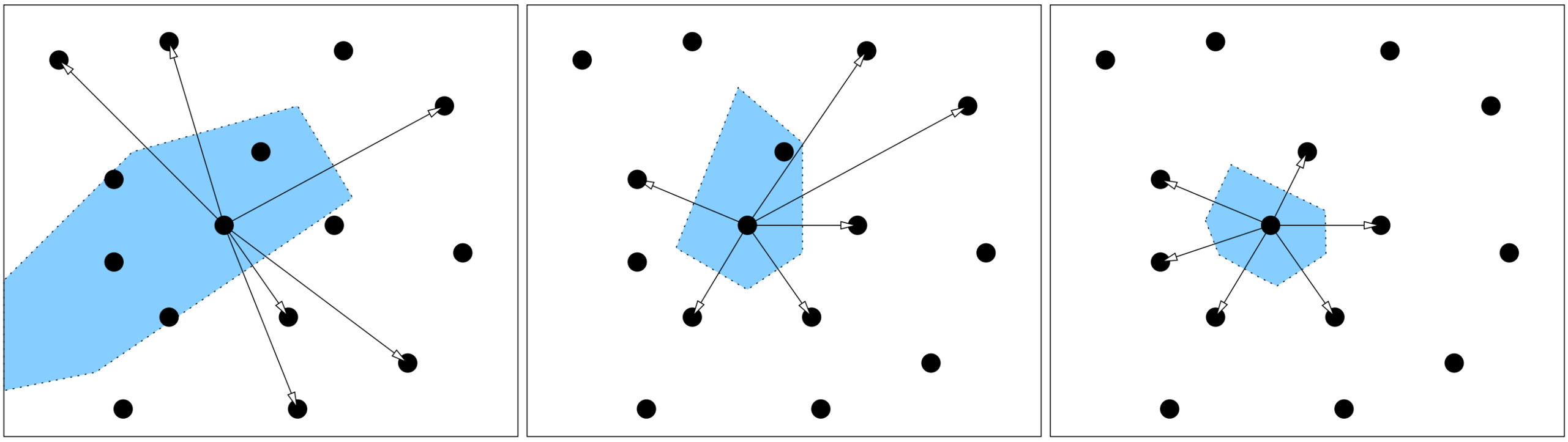
VoroNet: A scalable object network based on Voronoi tessellations



Distance function:

$$d(x, y) = \sqrt[2]{x^2 + y^2}$$

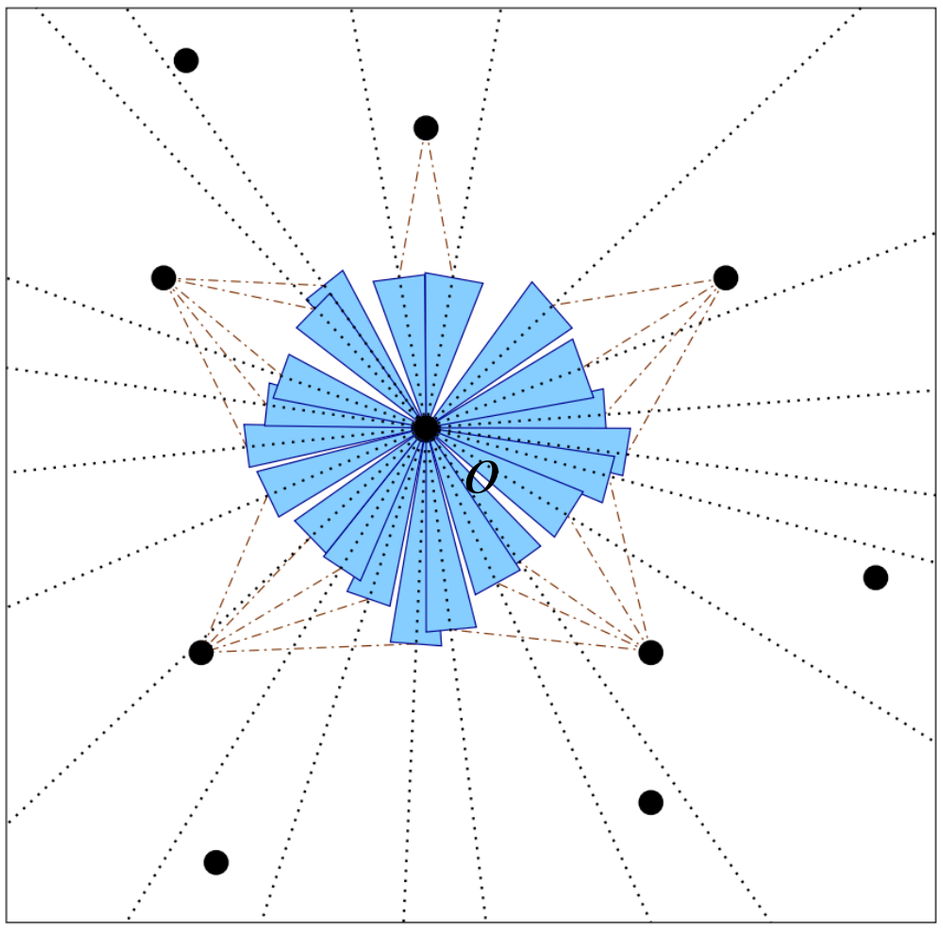
[Beaumont O. et al. VoroNet: A scalable object network based on Voronoi tessellations. – 2006]



[Beaumont, Olivier, Anne-Marie Kermarrec, and Étienne Rivière. "Peer to peer multidimensional overlays: Approximating complex structures." *International Conference On Principles Of Distributed Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.]

Monte-Carlo Algorithm For Estimating the Volume of the Voronoi Cell

```
calcVolume()
parameters: config (SET[objects])
begin
    SET[double] Λ ← ∅
    o.rays ← createRays(R)
(a) for double[] r ∈ o.rays do
        double λ ← ∞
        for object o_j ∈ config do
            double l ← compDistOnRay(r,o_j)
            if l < dist then
                λ ← l
(b)     Λ ← Λ ∪ λ
    /* BallVol contains the unit Ball volume in
    dimension d */
(c) return (BallVol × Σ_{λ∈Λ}(λ^d)) / R
end
```

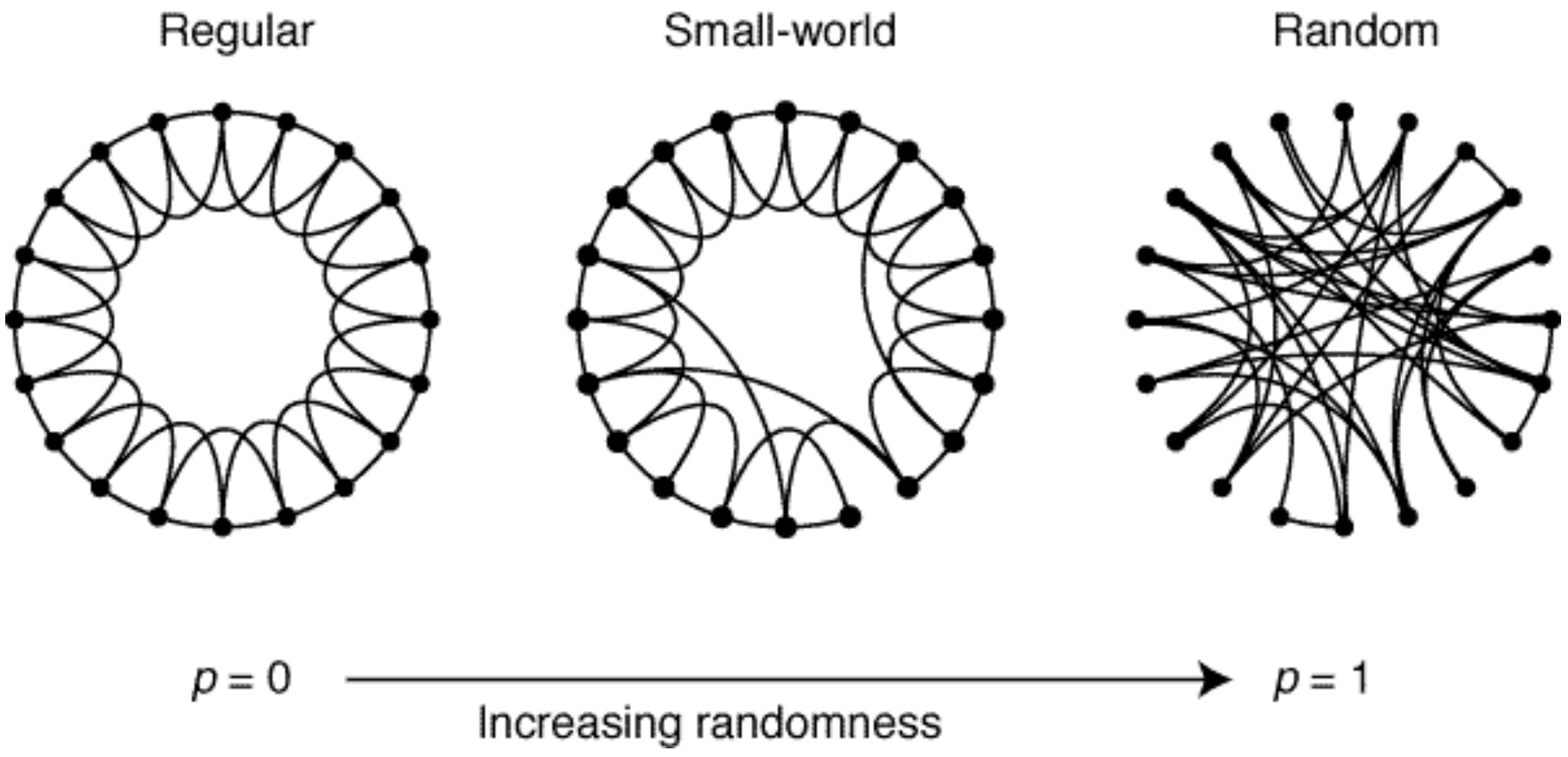


compDistOnRay(r, o_j) – compute distance from o to p_{int}, where $p_{int} \in R \ \wedge \ \|o, p_{int}\| = \|o_j, p_{int}\|$

[Beaumont, Olivier, Anne-Marie Kermarrec, and Étienne Rivière. "Peer to peer multidimensional overlays: Approximating complex structures." *International Conference On Principles Of Distributed Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.]

Emerging Small World

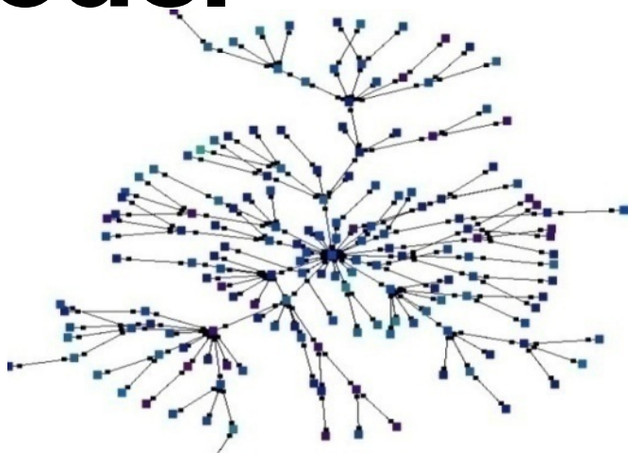
Watts-Strogatz

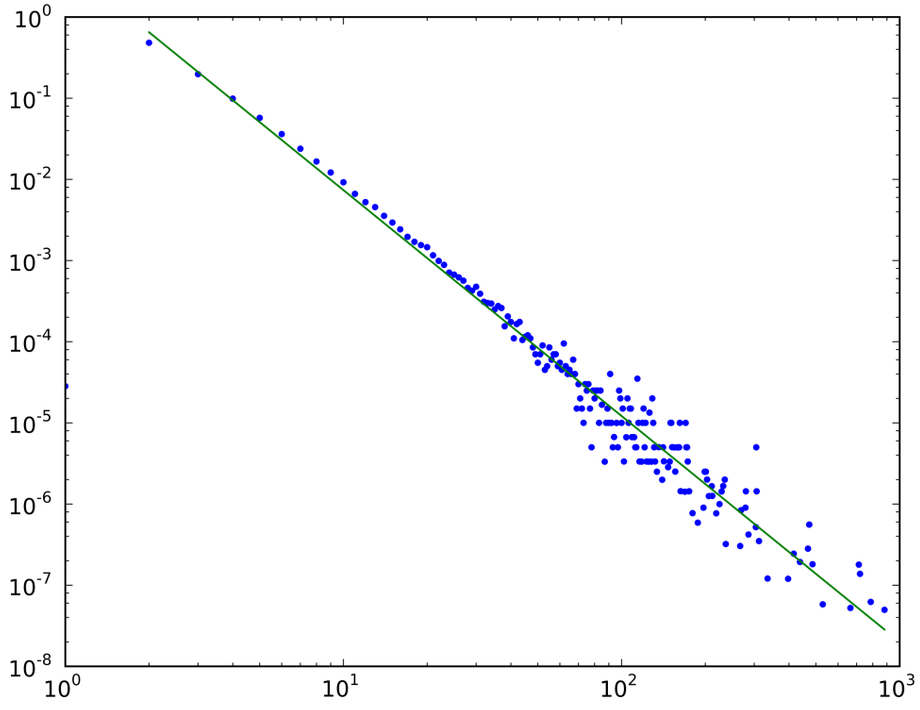


Barabási–Albert model

$$p_i = \frac{k_i}{\sum_j k_j}$$

$$P(k) \approx k^{-3}$$





The probability that the new node is connected to node ***i***

see also [Generalized preferential attachment by Ostroumova et.al.”]

Kleinberg Model

$$d((i,j),(k,l)) = |k-i| + |l-j|$$

$$P((u,v) \in E)^{-r} = \sum_{v} d(u,v)^{-r}$$

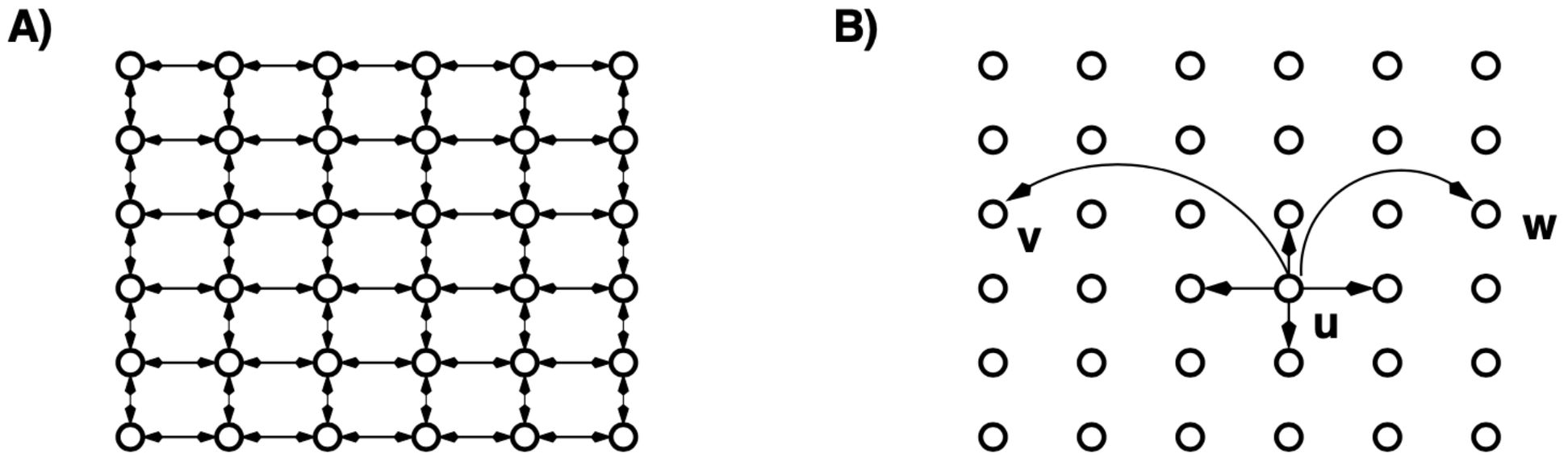


Figure 1: (A) A two-dimensional grid network with $n = 6$, $p = 1$, and $q = 0$. (B) The contacts of a node u with $p = 1$ and $q = 2$. v and w are the two long-range contacts.

[Kleinberg J. The small-world phenomenon: An algorithmic perspective //Proceedings of the thirty-second annual ACM symposium on Theory of computing. – ACM, 2000. – С. 163-170.]

Theorem 1 *There is a constant α_0, depending on p and q but independent of n, so that when $r = 0$, the expected delivery time of any decentralized algorithm is at least $\alpha_0 n^{2/3}$. (Hence exponential in the expected minimum path length.)*

Theoreme 2

There is a decentralized algorithm ***A*** , so that when r = 2 and p = q = 1, the expected delivery time of **A** is $O\left((\log n)^2\right)$

Theorem 3 *(a) Let $0 \leq r < 2$. There is a constant α_r, depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least $\alpha_r n^{(2-r)/3}$.*

(b) Let $r > 2$. There is a constant α_r, depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least $\alpha_r n^{(r-2)/(r-1)}$.

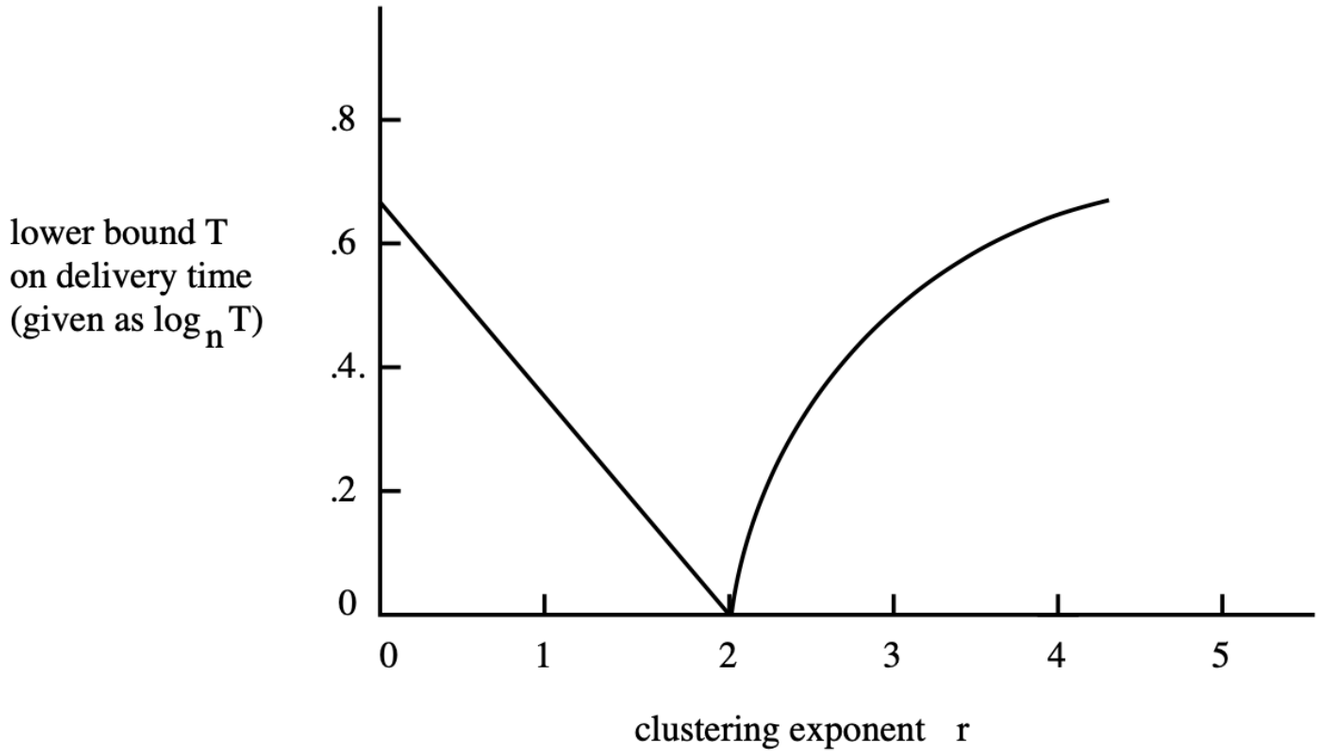


Figure 2: The lower bound implied by Theorem 3. The x-axis is the value of r; the y-axis is the resulting exponent on n.

Theoreme 2

There is a decentralized algorithm ***A*** , so that when r = 2
and p = q = 1, the expected delivery time of **A** is $\mathcal{O}((\log n)^2)$

$$P((u,v) \in E) = \frac{d(u,v)^{-2}}{\sum_{u \neq v} d(u,v)^{-2}}$$

$$\sum_{u \neq v} d(u,v)^{-2} \leq \sum_{j=1}^{2n-2} (4j)(j^{-2}) = 4 \sum_{j=1}^{2n-2} j^{-1} \leq 4 + 4\ln(2n-2) \leq 4\ln(6n)$$

$$P((u,v) \in E) \geq \frac{1}{4\ln(6n) d(u,v)^2}$$

We are at the stage j when $2^j < d(t,u) \leq 2^{j+1}$

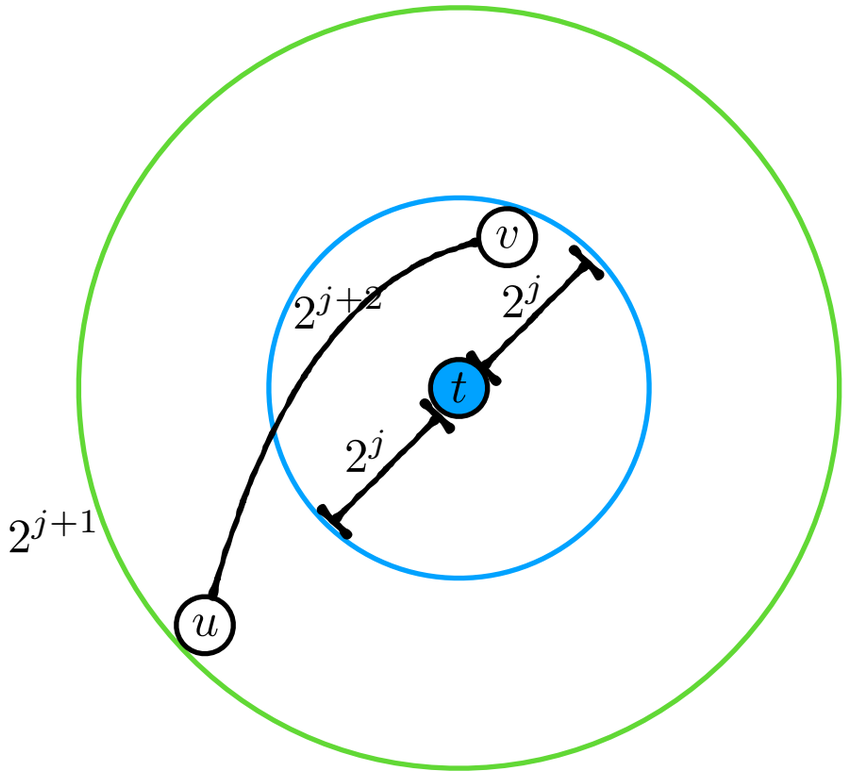
The initial value of *j* is at most log *n*.

Suppose we are at the stage *j*, and $\log(\log n) \leq j < \log(n)$

What is the probability that phase *j* will end in this step?

The **stage *j*** if the message at ***x***, and $2^j < d(t,x) \leq 2^{j+1}$

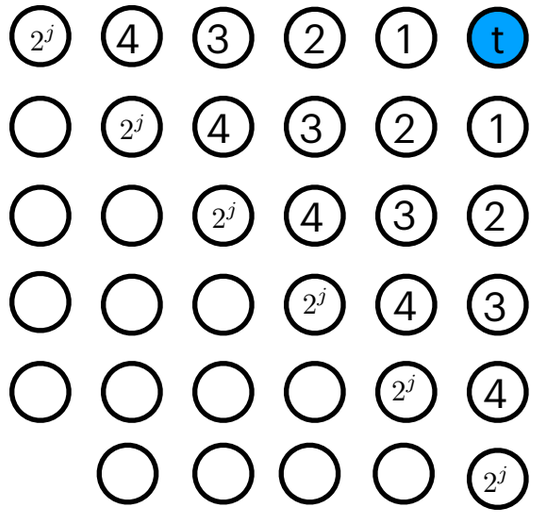
$$d(u,v) \leq 2^{j+2}$$



$$P((u,v) \in E) \geq \frac{1}{4\ln(6n)d(u,v)^2}$$

$$P\Big((u,v) \in E\Big) \geq \frac{1}{4\ln(6n)2^{(j+2)^2}} = \frac{1}{4\ln(6n)2^{2j+4}}$$

Let the **stage *j*** if the message at ***x***, and $2^j < d(t,x) \le 2^{j+1}$



Let $B_j = \{x : d(t,x) \le 2^j\}$

$$P\Big((u,v) \in E\Big) \ge \frac{1}{4\ln(6n)2^{2j+4}}$$

$$|B_j| \ge \sum_{i=1}^{2^{j+1}} i > \sum_{i=1}^{2^j} i = 2^j(2^j+1)\frac{1}{2} = 2^{2j}\frac{1}{2} + 2^j\frac{1}{2} > 2^{2j-1}$$

$$P\Big(\Big\{\{(u,v) : v \in B_j\} \cap E\Big\} \neq \emptyset\Big) \ge \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i)$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i)$$

Если случайная величина X может принимать только натуральные значения $(0, 1, 2, \ldots)$, то имеется красивая формула для её математического ожидания:

$$\begin{aligned} \mathsf{M}[X] &= \sum_{i=0}^{\infty} i\mathsf{P}\{X = i\} \\ &= \sum_{i=0}^{\infty} i(\mathsf{P}\{X \geqslant i\} - \mathsf{P}\{X \geqslant i+1\}) \\ &= \sum_{i=1}^{\infty} \mathsf{P}\{X \geqslant i\}. \end{aligned} \tag{6.28}$$

В самом деле, каждый член $\mathsf{P}\{X \geqslant i\}$ присутствует в сумме i раз со знаком плюс и $i-1$ раз со знаком минус (исключение составляет член $\mathsf{P}\{X \geqslant 0\}$, вовсе отсутствующий в сумме).

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i)$$

Если случайная величина X может принимать только натуральные значения $(0, 1, 2, \ldots)$, то имеется красивая формула для её математического ожидания:

$$\begin{aligned} \mathsf{M}[X] &= \sum_{i=0}^{\infty} i\mathsf{P}\{X = i\} \\ &= \sum_{i=0}^{\infty} i(\mathsf{P}\{X \geqslant i\} - \mathsf{P}\{X \geqslant i+1\}) \\ &= \sum_{i=1}^{\infty} \mathsf{P}\{X \geqslant i\}. \end{aligned} \tag{6.28}$$

В самом деле, каждый член $\mathsf{P}\{X \geqslant i\}$ присутствует в сумме i раз со знаком плюс и $i-1$ раз со знаком минус (исключение составляет член $\mathsf{P}\{X \geqslant 0\}$, вовсе отсутствующий в сумме).

	+	+	+
0*(	p{X>=0} - p{X>=1})	0*p{X>=0}	
1*(	p{X>=1} - p{X>=2})	1*p{X>=1} - 0*p{X>=1}	1*p{X>=1}
2*(	p{X>=2} - p{X>=3})	-1*p{X>=2} + 2*p{X>=2}	1*p{X>=2}
3*(	p{X>=3} - p{X>=4})	-2*p{X>=3} + 3*p{X>=3}	1*p{X>=3}
i-1*(	p{X>=i-1} - p{X>=i})		
i*(	p{X>=i} - p{X>=i+1})	-(i-1)*p{X>=i} + i*p{X>=i}	1*p{X>=i}
i+1*(	p{X>=i+1} - p{X>=i+2})		1*p{X>=i+1}

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1}$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1}$$

$$S_n = b_1 + b_2 + ... + b_n = b_1 + b_1 q + b_1 q^2 + ... + b_1 q^{n-1}$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1}$$

$$S_n = b_1 + b_2 + ... + b_n = b_1 + b_1 q + b_1 q^2 + ... + b_1 q^{n-1}$$

$$q = 1 - \frac{1}{128\ln(6n)}$$

$$b_1 = 1$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1}$$

$$S_n = b_1 + b_2 + ... + b_n = b_1 + b_1 q + b_1 q^2 + ... + b_1 q^{n-1}$$

$$q = 1 - \frac{1}{128\ln(6n)}$$

$$b_1 = 1$$

$$S_n = \frac{b_1(q^n - 1)}{q - 1}$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1}$$

$$S_n = b_1 + b_2 + ... + b_n = b_1 + b_1 q + b_1 q^2 + ... + b_1 q^{n-1}$$

$$q = 1 - \frac{1}{128\ln(6n)}$$

$$b_1 = 1$$

$$S_n = \frac{b_1(q^n - 1)}{q - 1} \quad \lim_{n\to\infty} S_n = \frac{b_1}{1 - q}$$

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1} = 128\ln(6n)$$

$$S_n = b_1 + b_2 + ... + b_n = b_1 + b_1 q + b_1 q^2 + ... + b_1 q^{n-1}$$

$$q = 1 - \frac{1}{128\ln(6n)}$$

$$b_1 = 1$$

$$S_n = \frac{b_1(q^n - 1)}{q - 1} \quad \lim_{n\to\infty} S_n = \frac{b_1}{1 - q}$$

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1} = 128\ln(6n)$$

Let X_j — the number of steps at stage j

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1} = 128\ln(6n)$$

$$X = \sum_{j=0}^{\log n} X_j \qquad \text{— total number of steps}$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1} = 128\ln(6n)$$

$$X = \sum_{j=0}^{\log n} X_j \qquad \text{— total number of steps}$$

$$E[X] \leq (1 + \log n)(128\ln(6n)) = O\left((\log n)^2\right)$$

$$P\left(\left\{\{(u,v) : v \in B_j\} \cap E\right\} \neq \emptyset\right) \geq \frac{2^{2j-1}}{4ln(6n)2^{2j+4}} = \frac{1}{128\ln(6n)}$$

Let X_j — the number of steps at stage j

$$E[X_j] = \sum_{i=1}^{\infty} P(X_j \geq i) \quad \leq \sum_{i=1}^{\infty} \left(1 - \frac{1}{128\ln(6n)}\right)^{i-1} = 128\ln(6n)$$

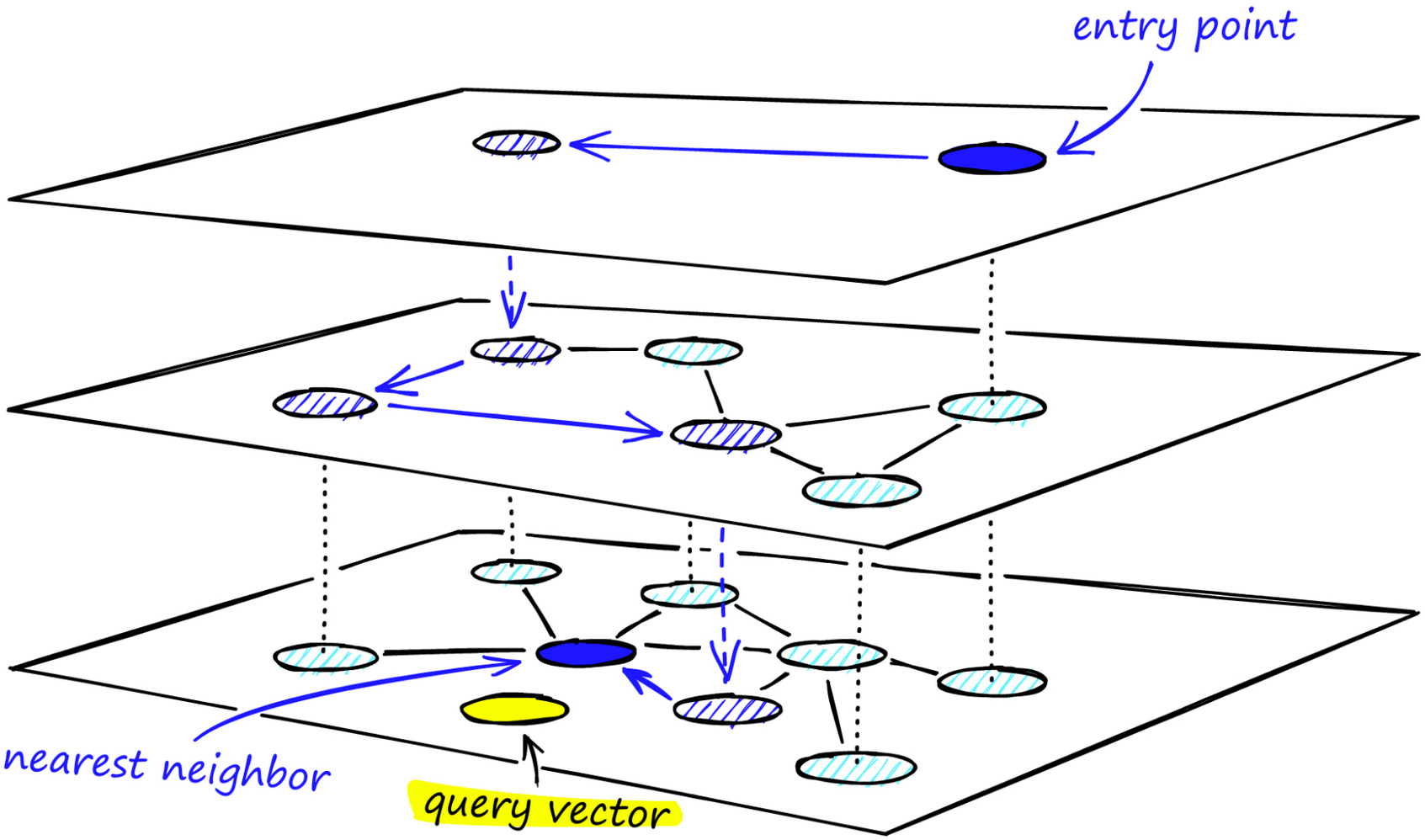
$$X = \sum_{j=0}^{\log n} X_j \qquad \text{— total number of steps}$$

$$E[X] \leq (1 + \log n)(128\ln(6n)) = O\left((\log n)^2\right)$$

■

HNSW

HSNW (Hierarchical Navigable Small Worlds)



https://www.pinecone.io/learn/series/faiss/hnsw/

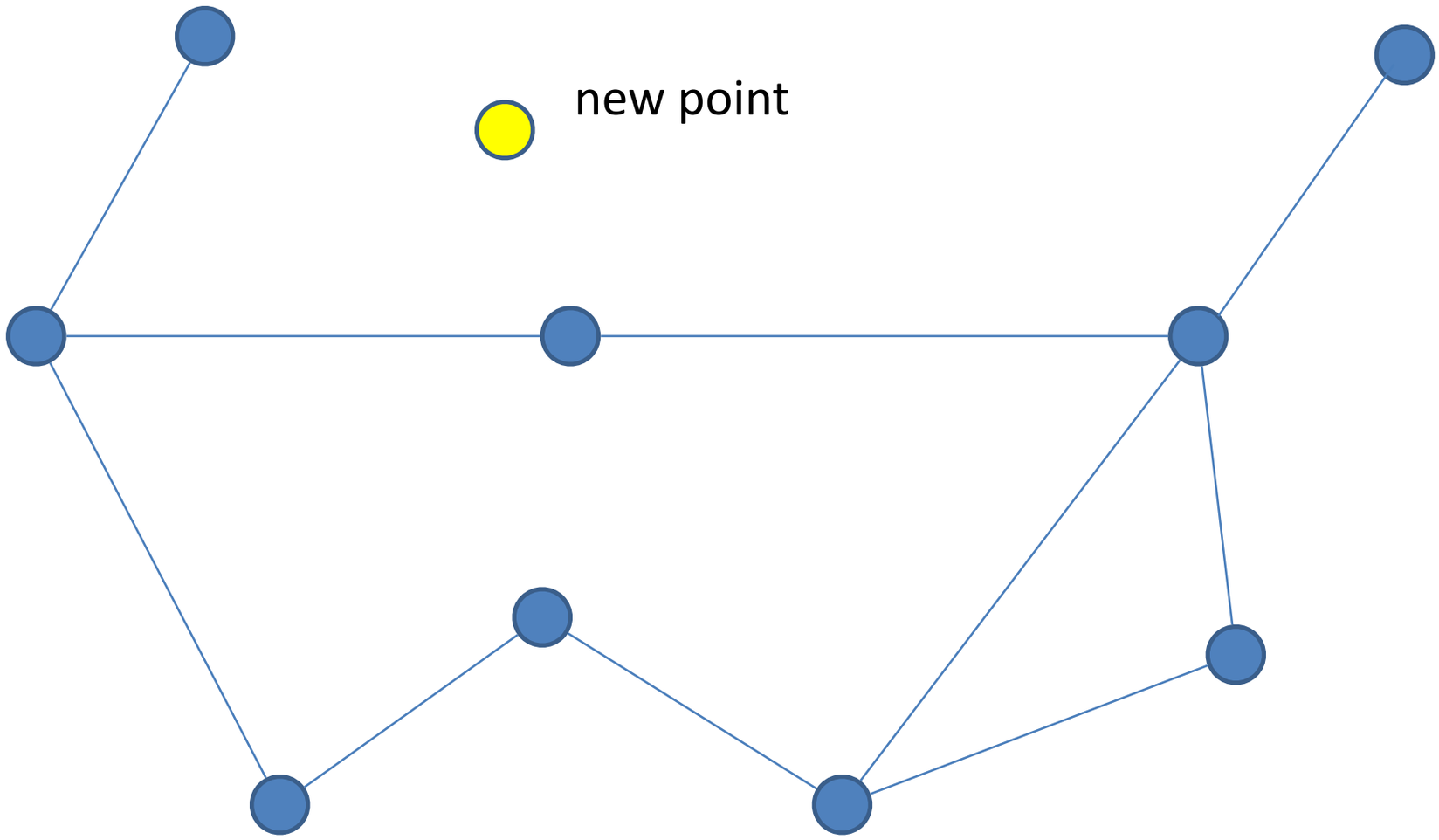
[Malkov, Yu A., and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018): 824-836.]

Query-Based Improvement Procedure and Self-Adaptive Graph Construction Algorithm

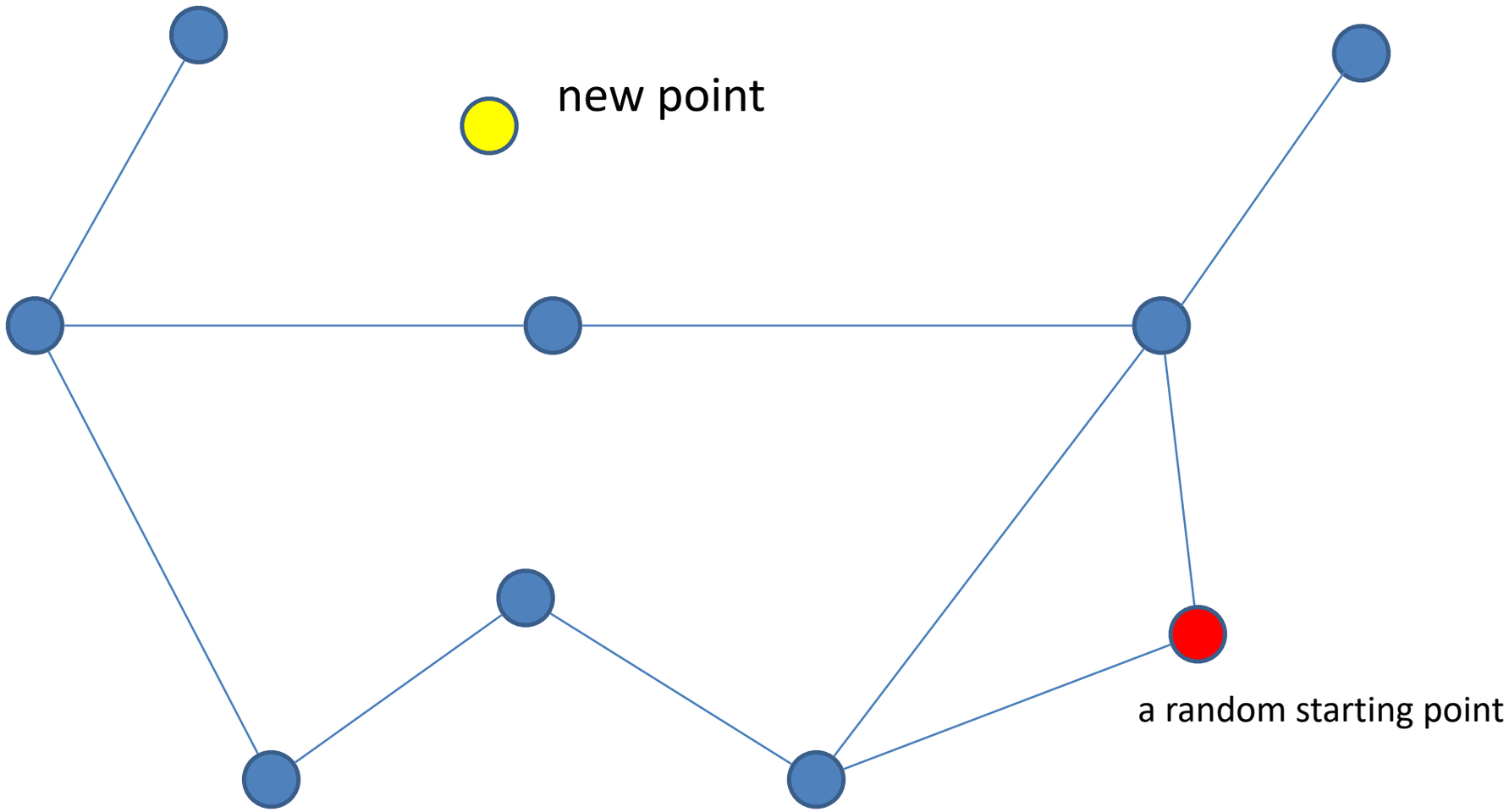
Self-adaptive Graph Construction Algorithm

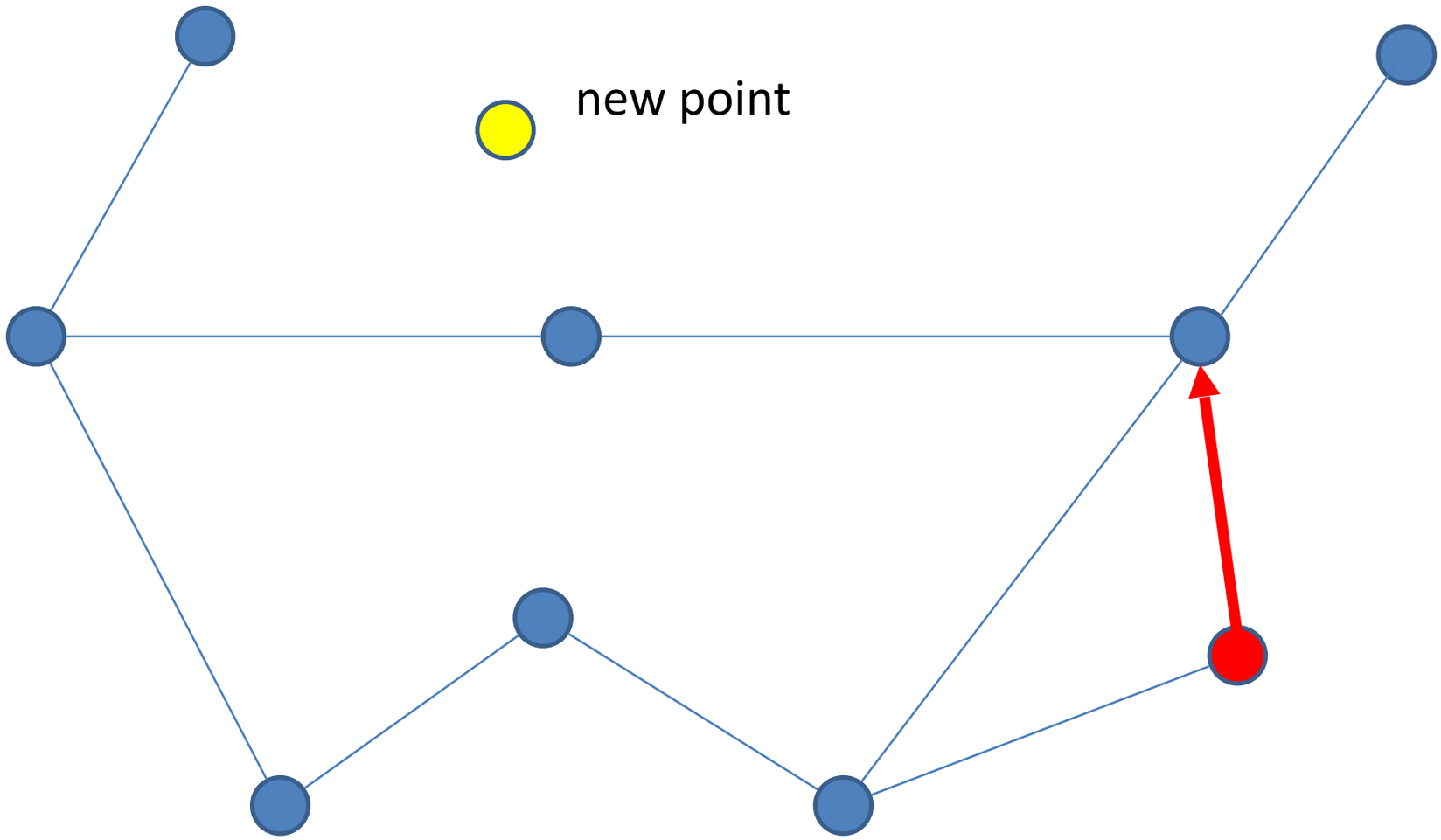
```
Get_Local_Minimums(q ∈ D, G(V,E), τ ∈ N)
01 L ← {}; τ' ← 0
02  while τ' < τ do
03    v_start ← Random(V) //put to v_start random vertex from V
04    v, P ← Greedy_Walk(q, G, v_start)
05    if v ∉ L then τ' ← 0; L ← L ∪ v
08    else τ' ← τ'+1
09  end while
11  return L, P
```

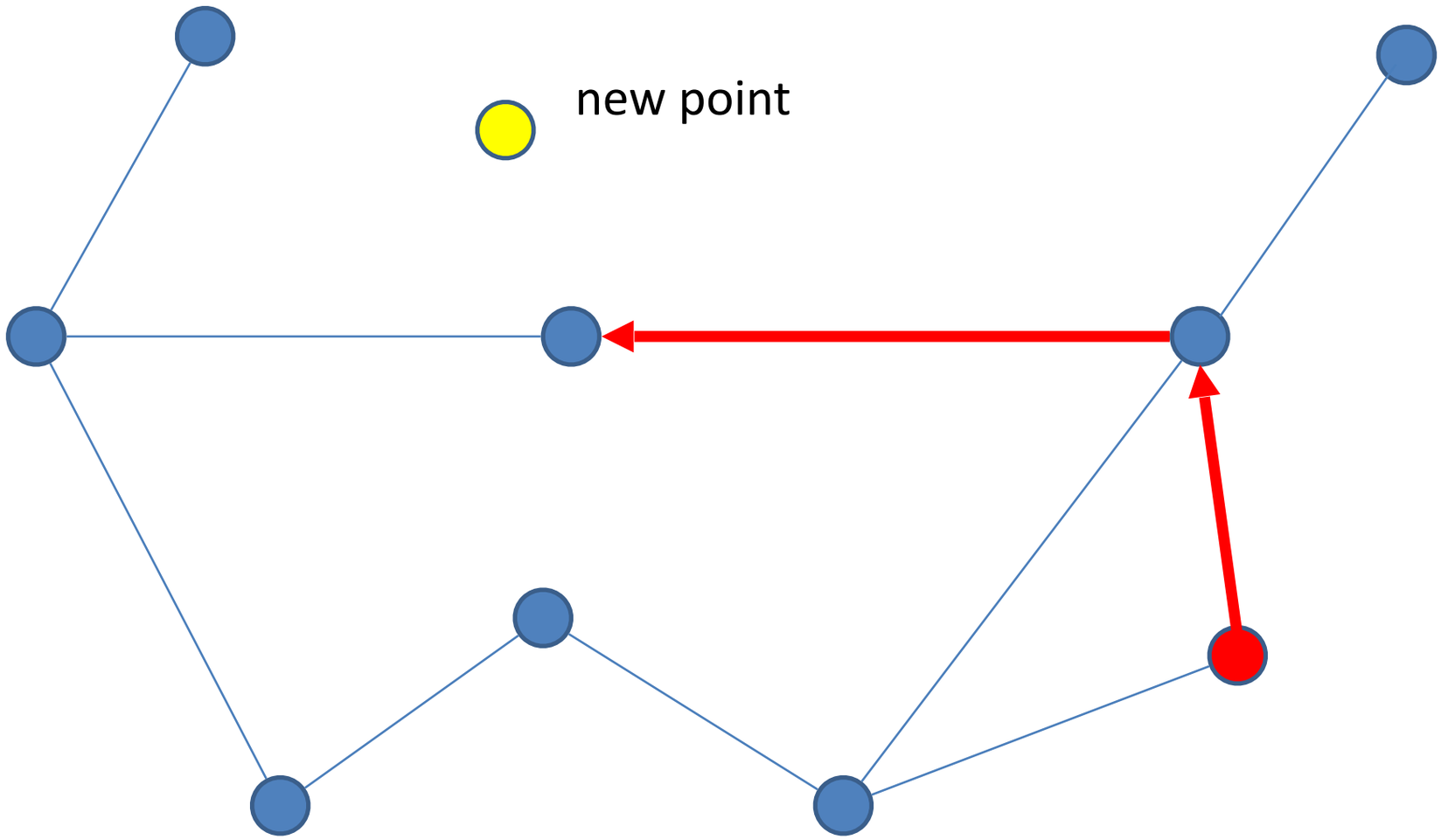
```
Insert_By_Repairing(x ∈ X, G(V,E), τ ∈ N)
01   V ← V ∪ x; P ← {}
02   L, P ← Get_Local_Minimums(x, G, τ)
03   P ← P ∪ x
04   for each z ∈ L do
05     P' ← {y ∈ P : d(y,x) < d(z,x)}
06     x' ← arg min_{y∈P'} (d(z,y))
07     E ← E ∪ (x',z) ∪ (z,x')
08   end for
```

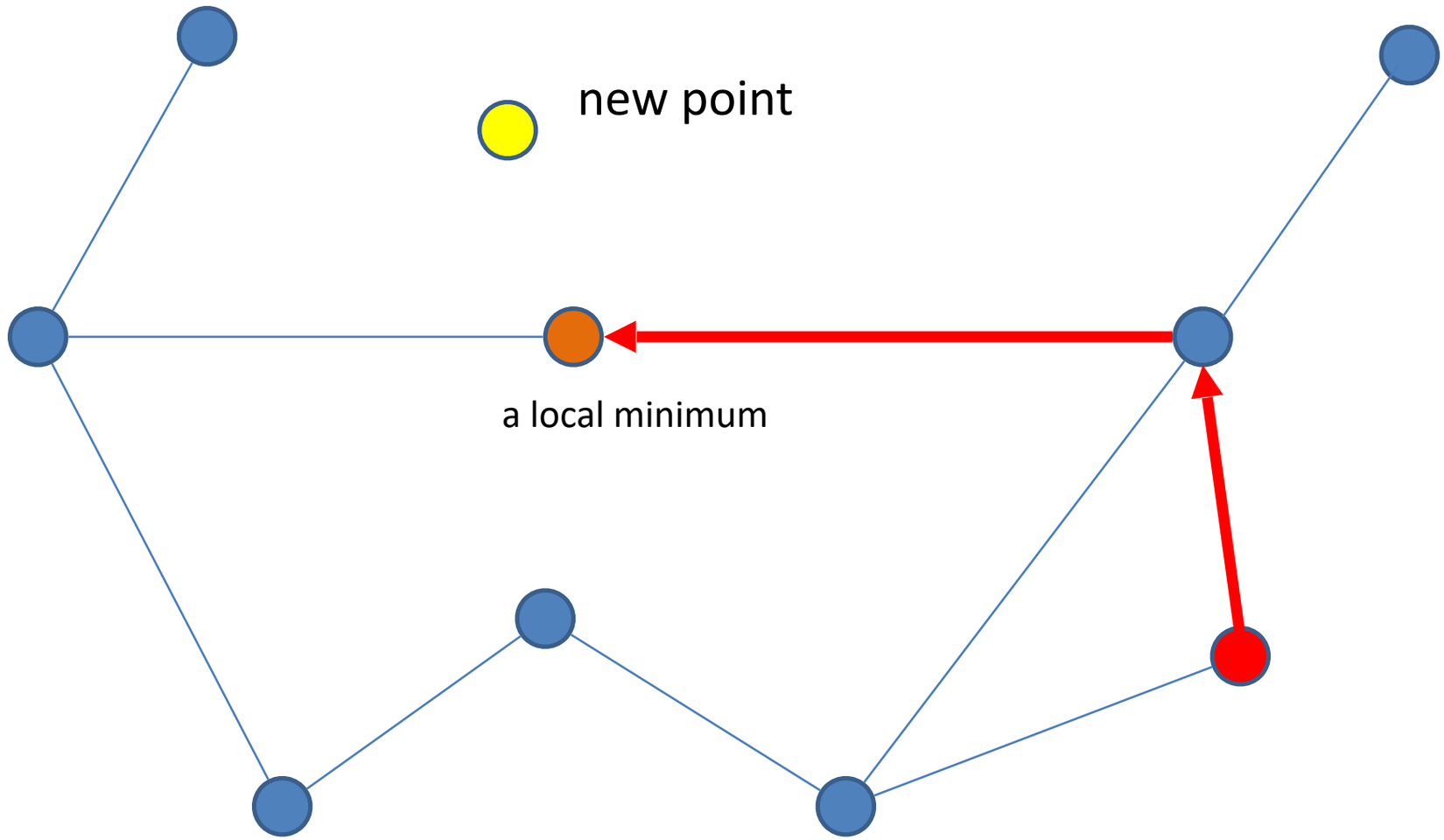

new point

Select a random starting point

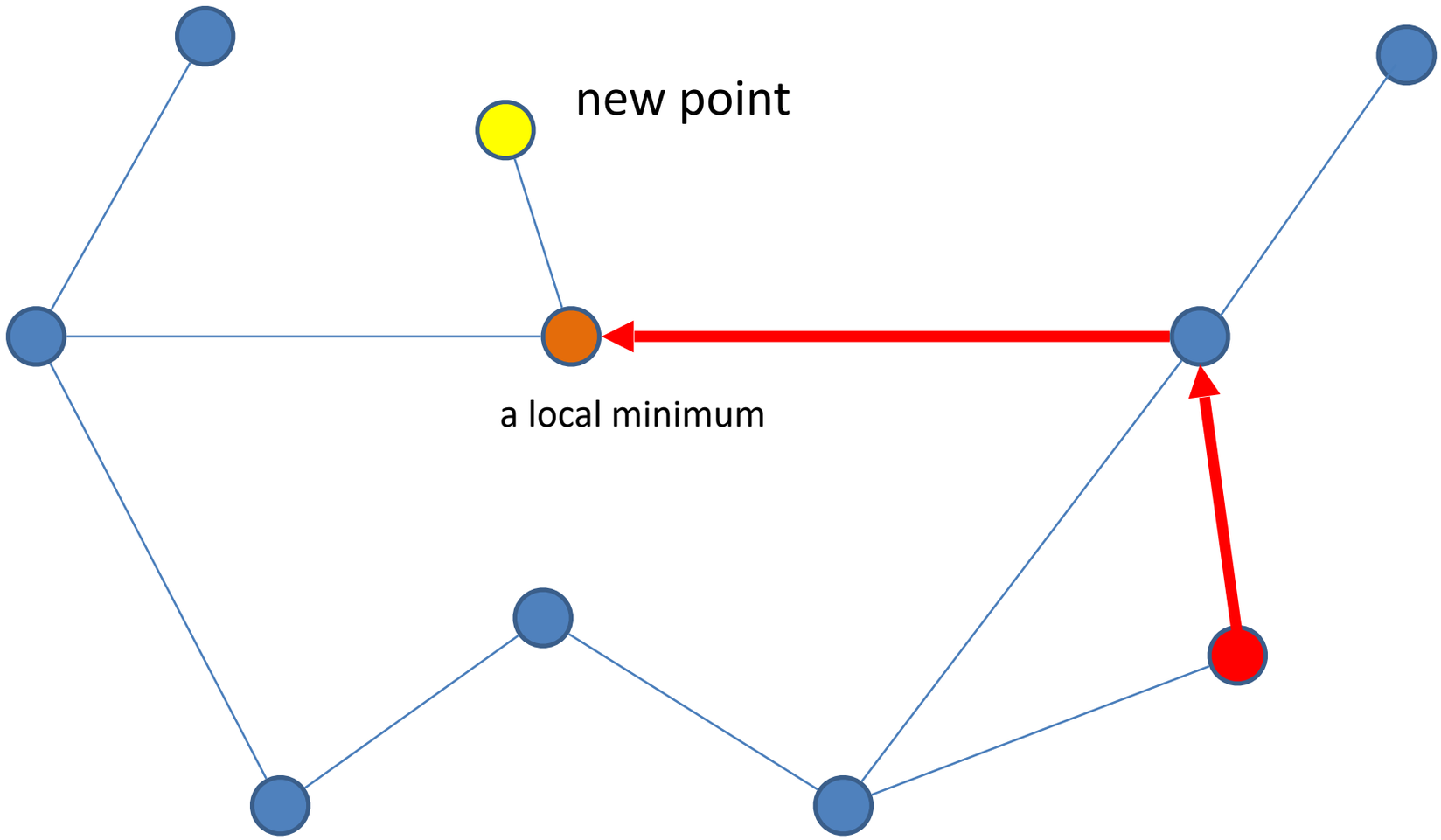



new point

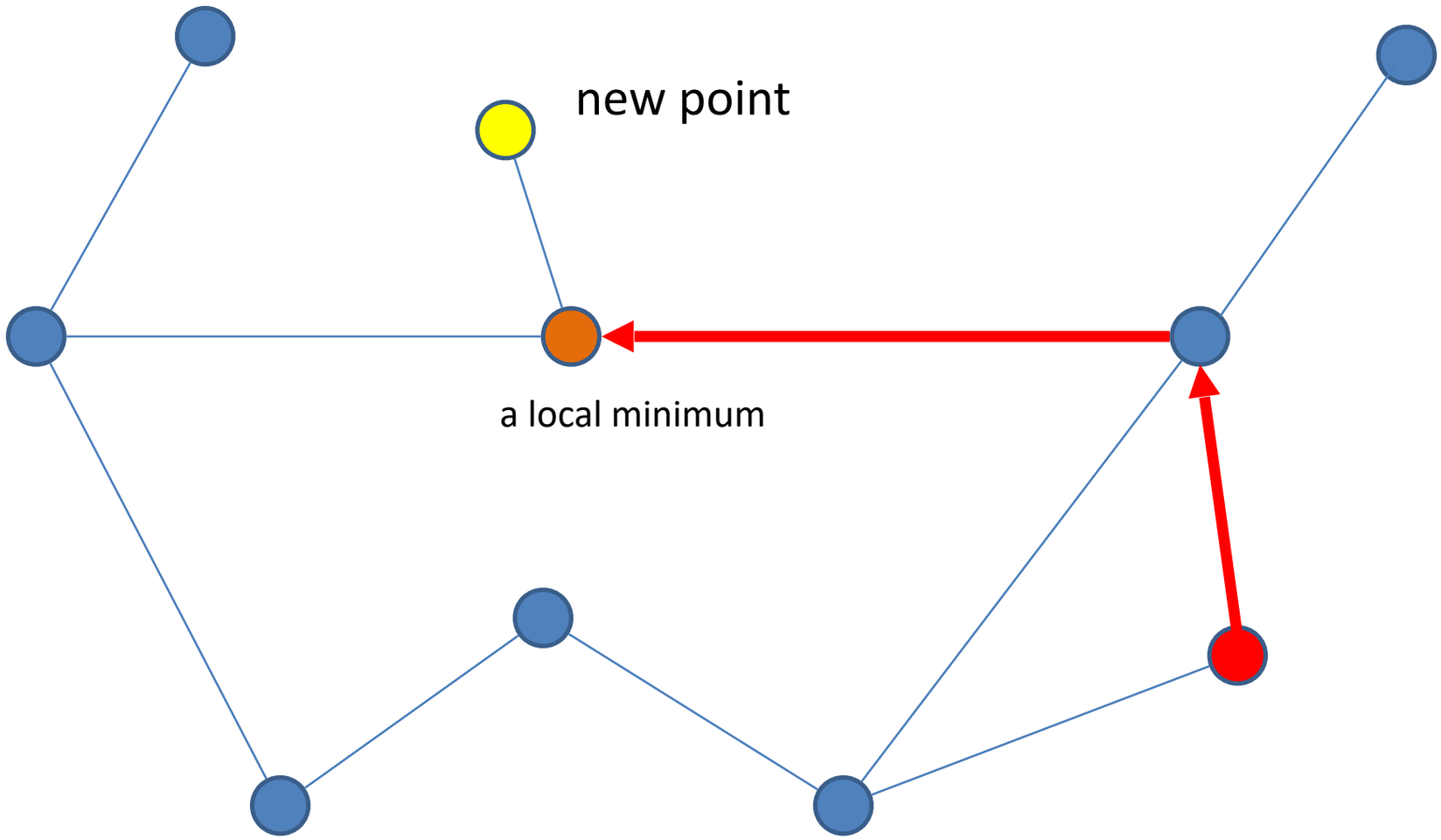

new point


new point
a local minimum

Adding new edge, which will remove a local minimum

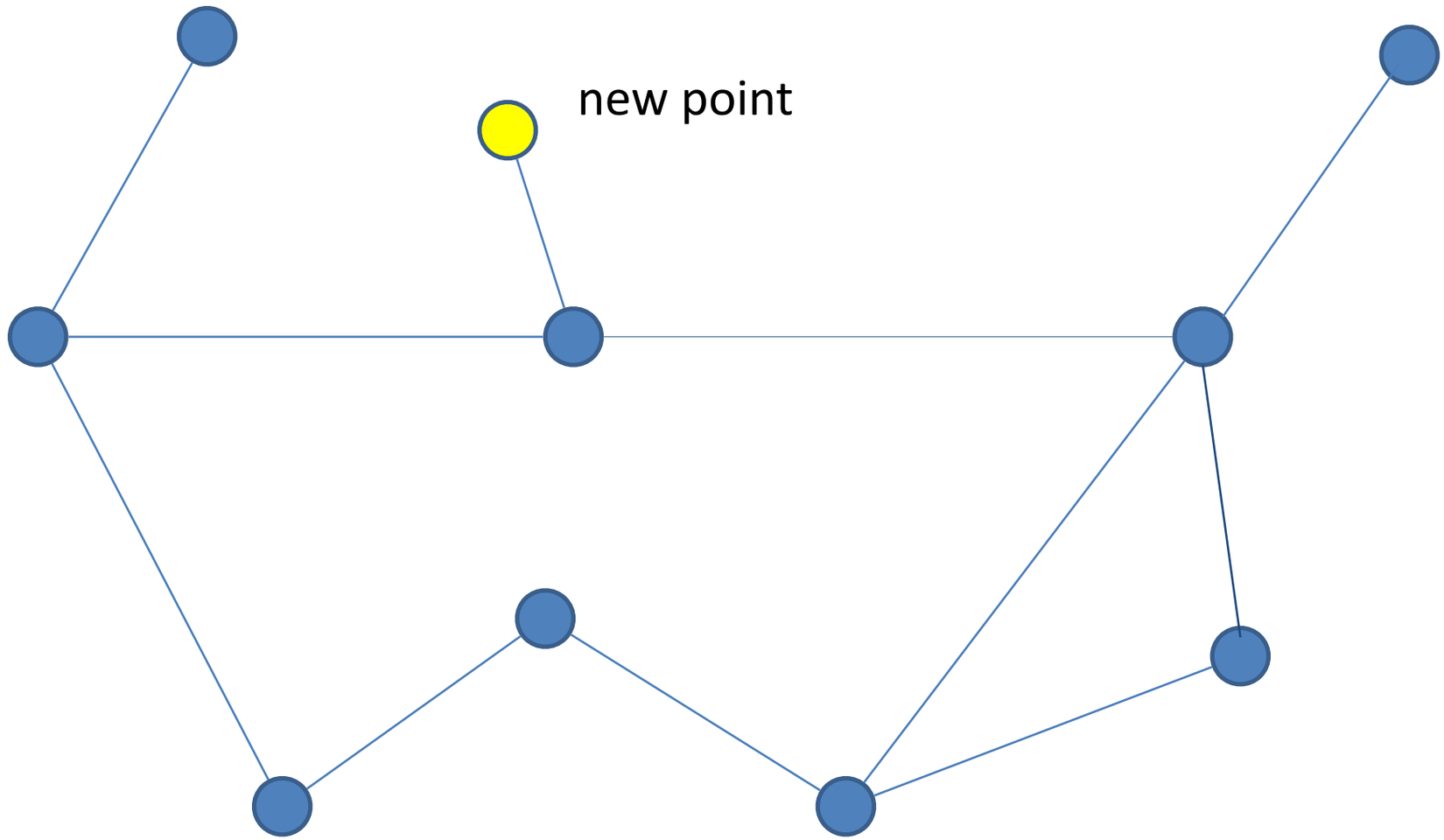


Note, that our greedy walk has not reach the global optimum (new point)



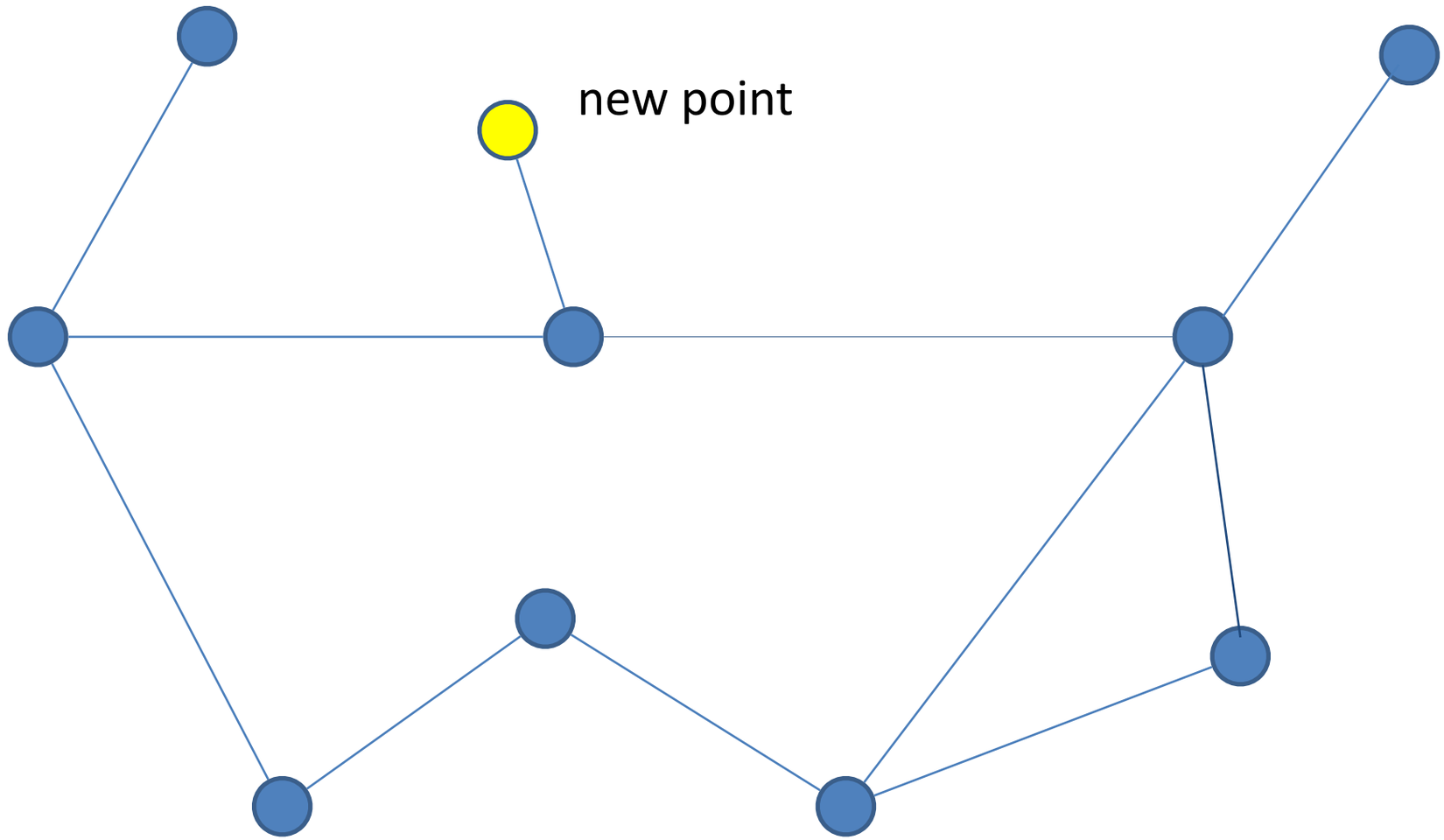
fail

Lets do another attempt!



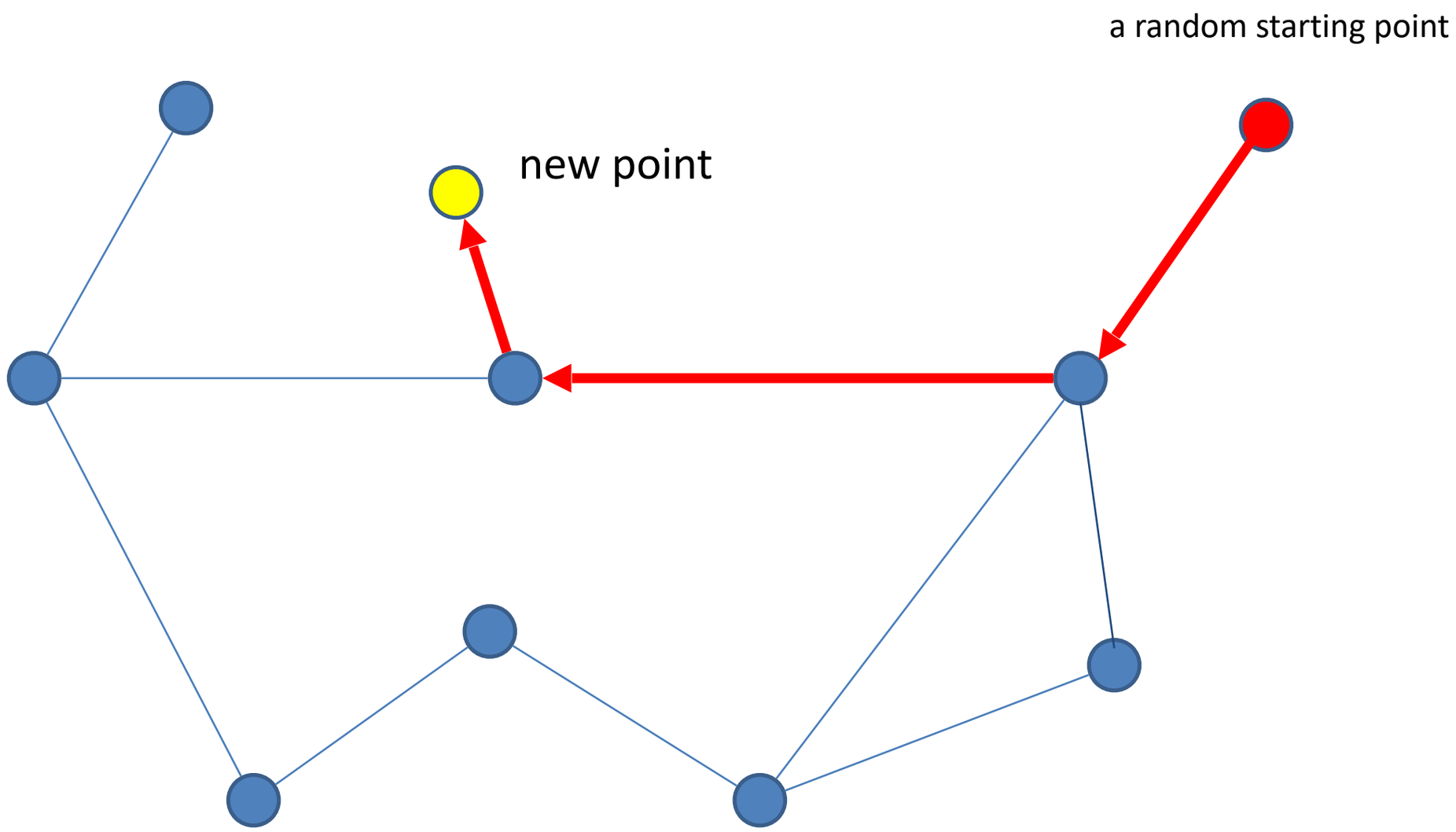
fail

Select a random starting point and try to rich a new point by greedy walk



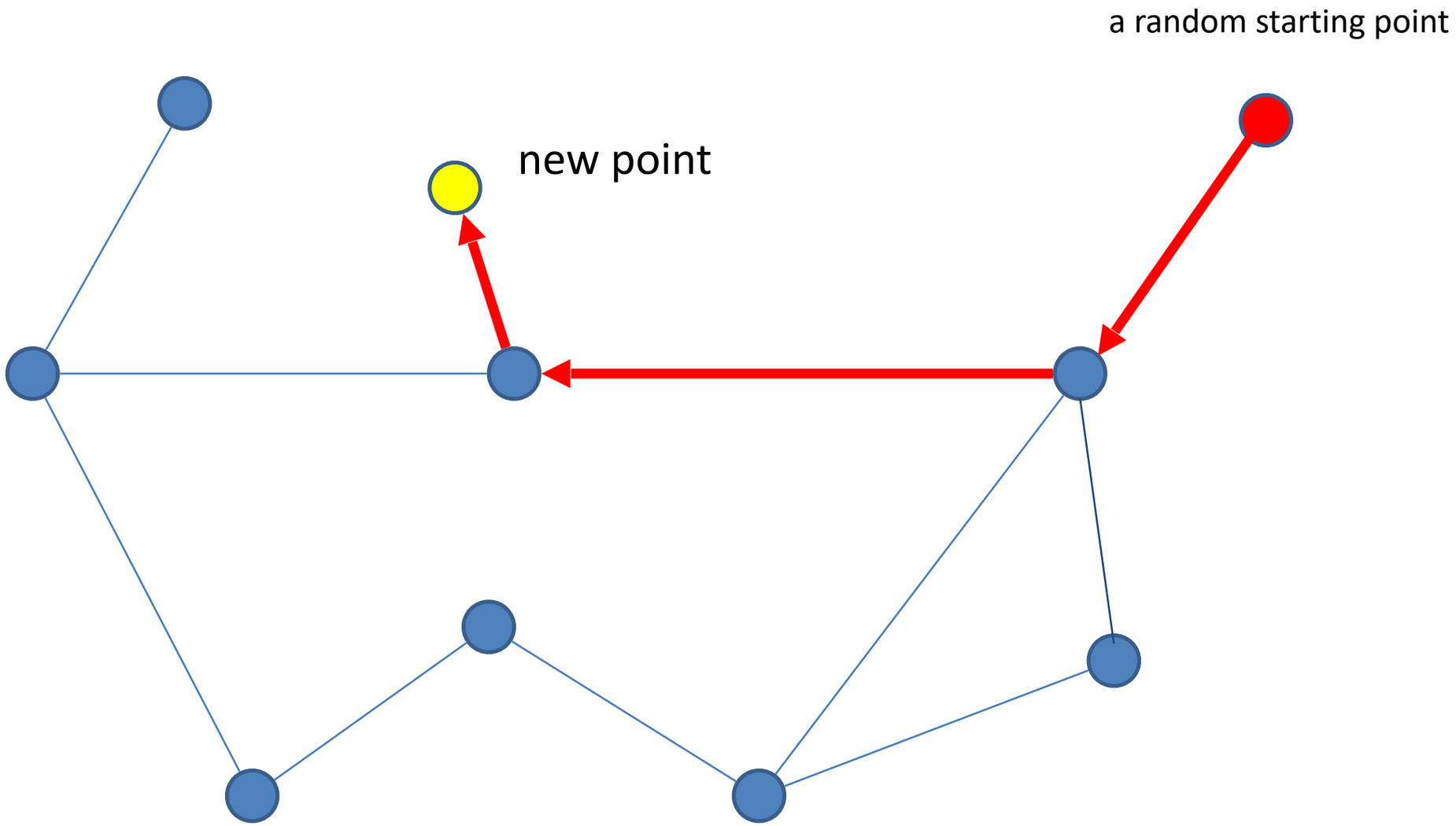
fail

Success!



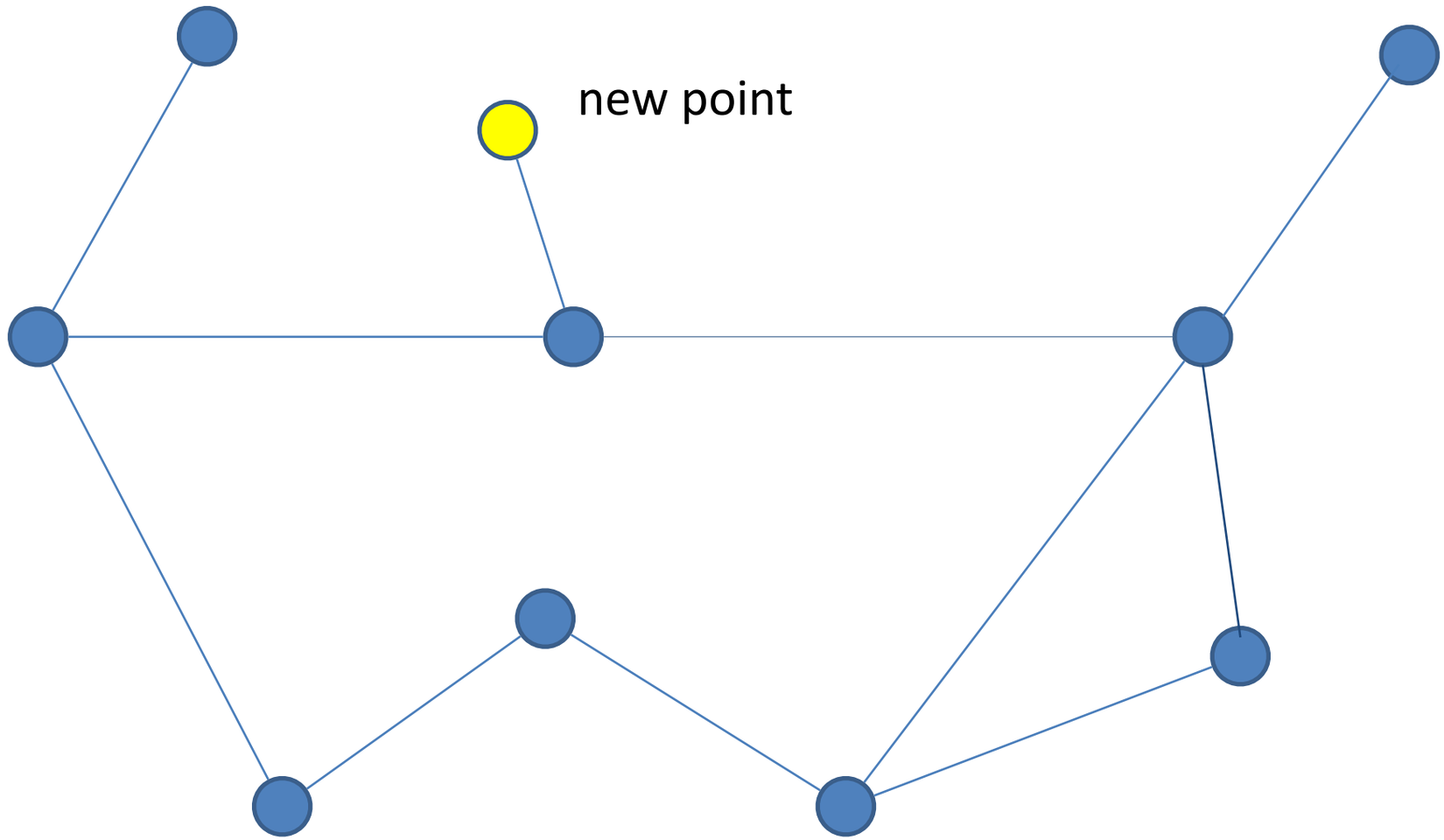
fail

Celebrate our success by adding new mark to the “success” table



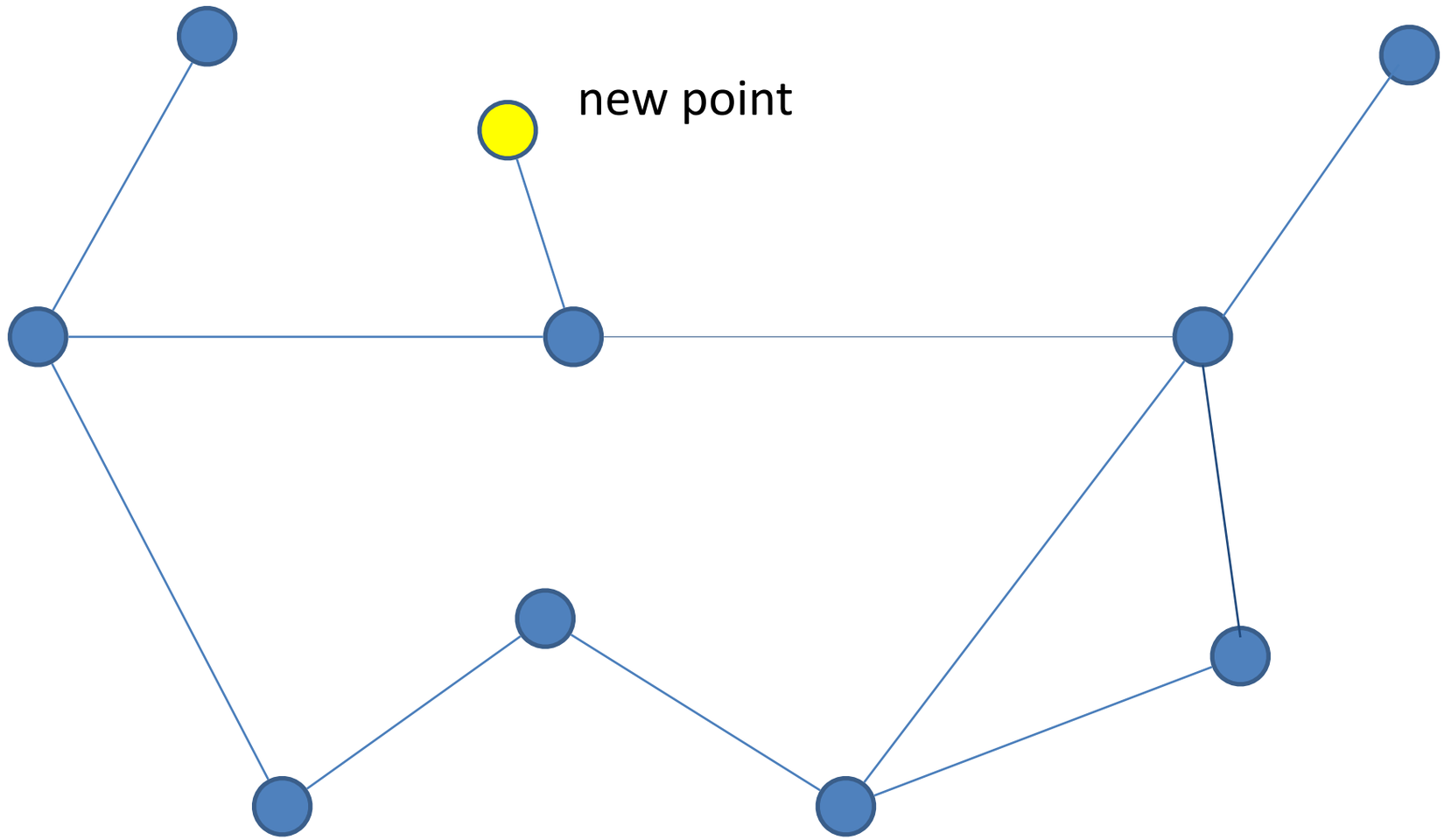
fail
success

Lets do another attempt!



fail
success

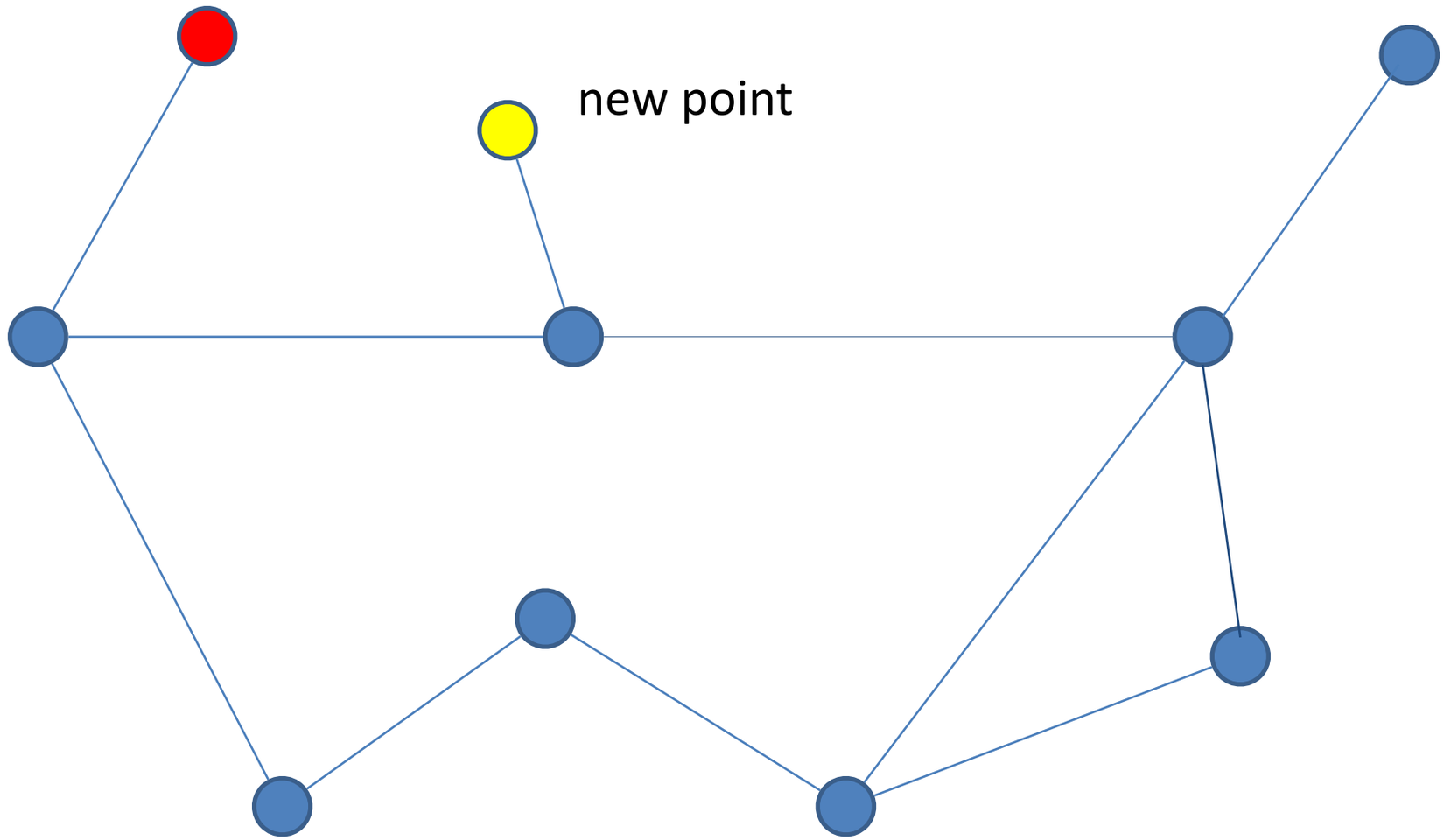
Select a random starting point and try to rich a new point by greedy walk



fail
success

Select a random starting point and try to rich a new point by greedy walk

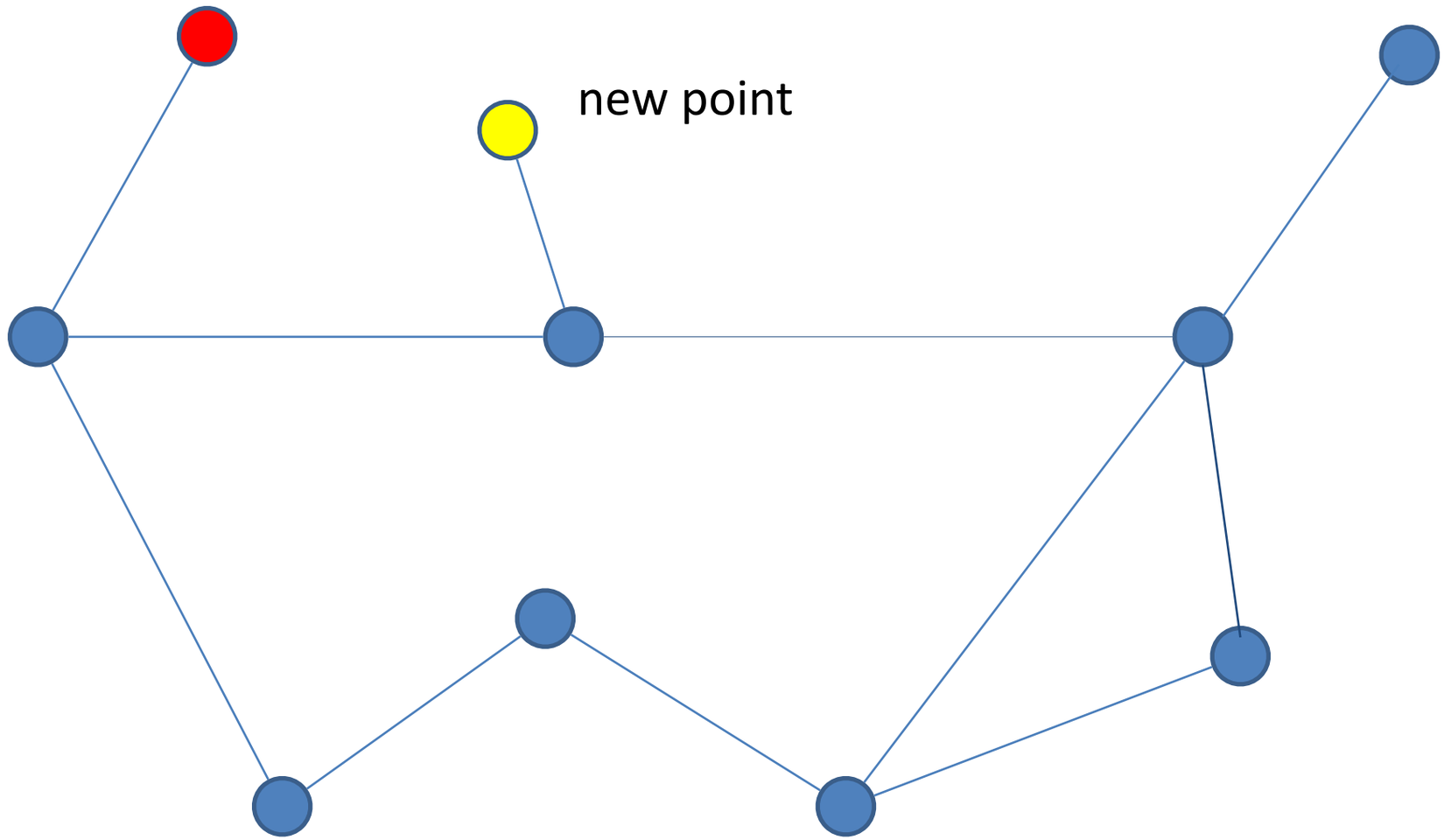
a random starting point



fail
success

Our starting point is local minimum!

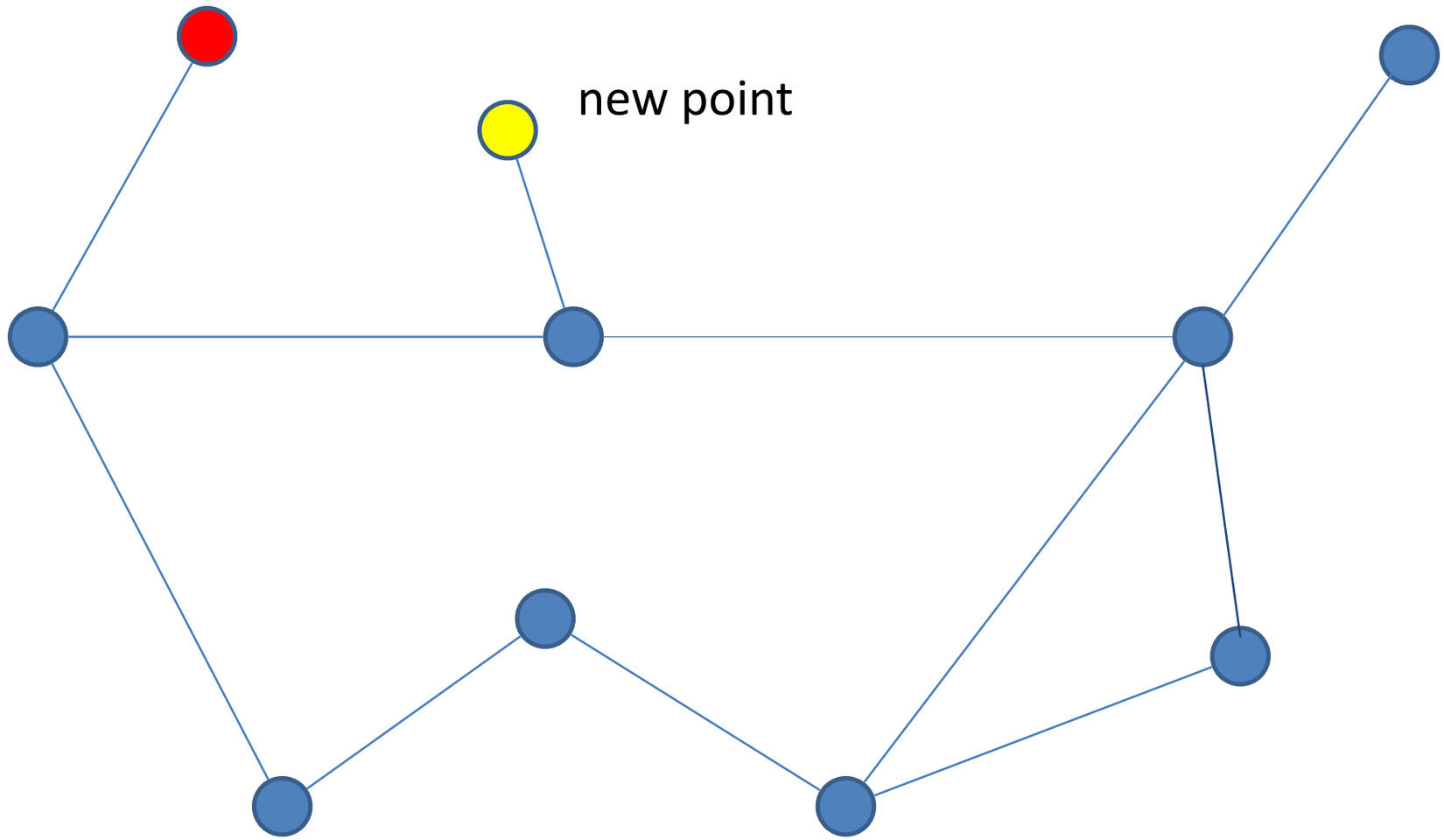
Local minimum



fail
success

Note that we have fail to reach global optimum

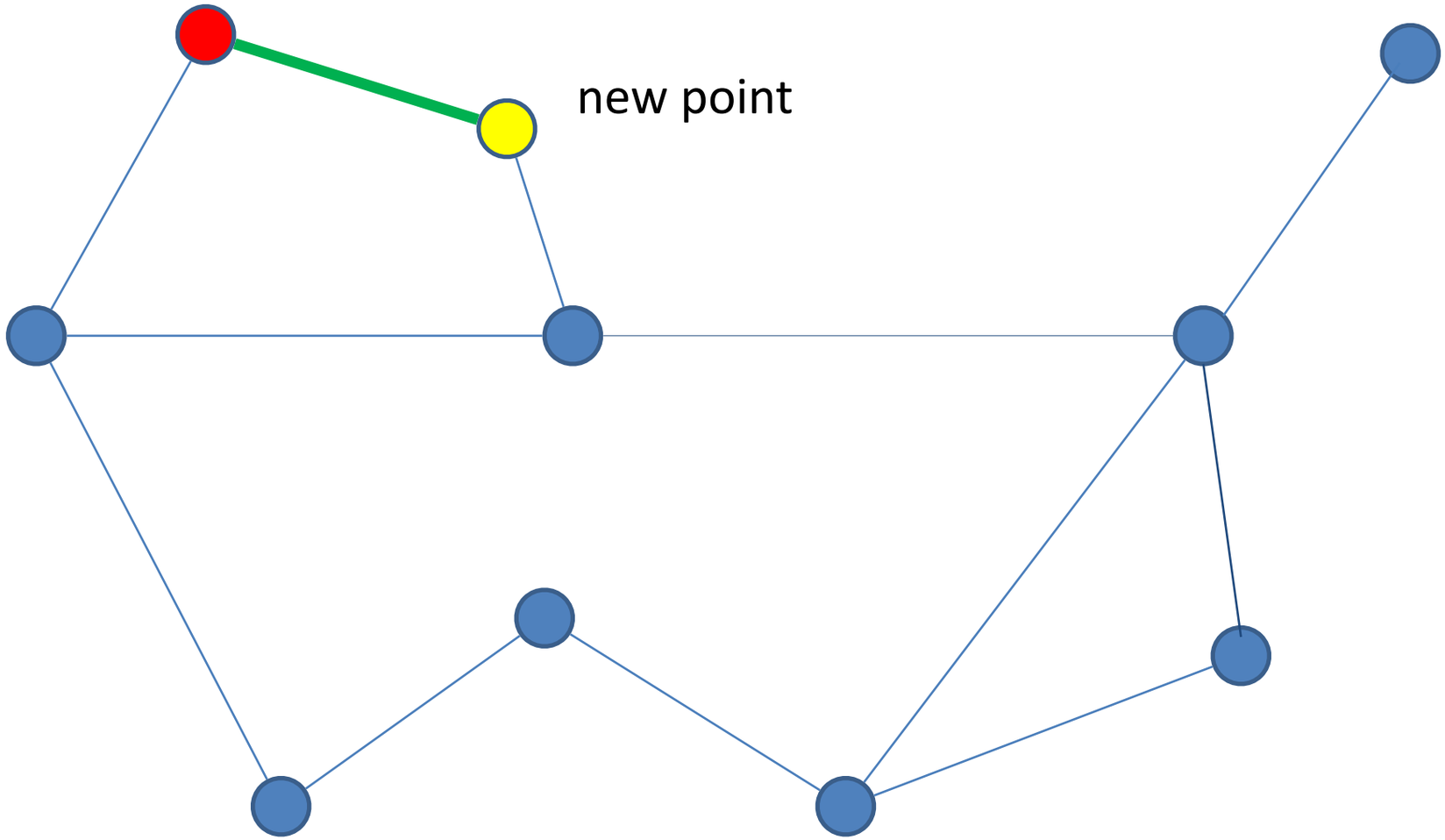
Local minimum



Fail
Success
Fail

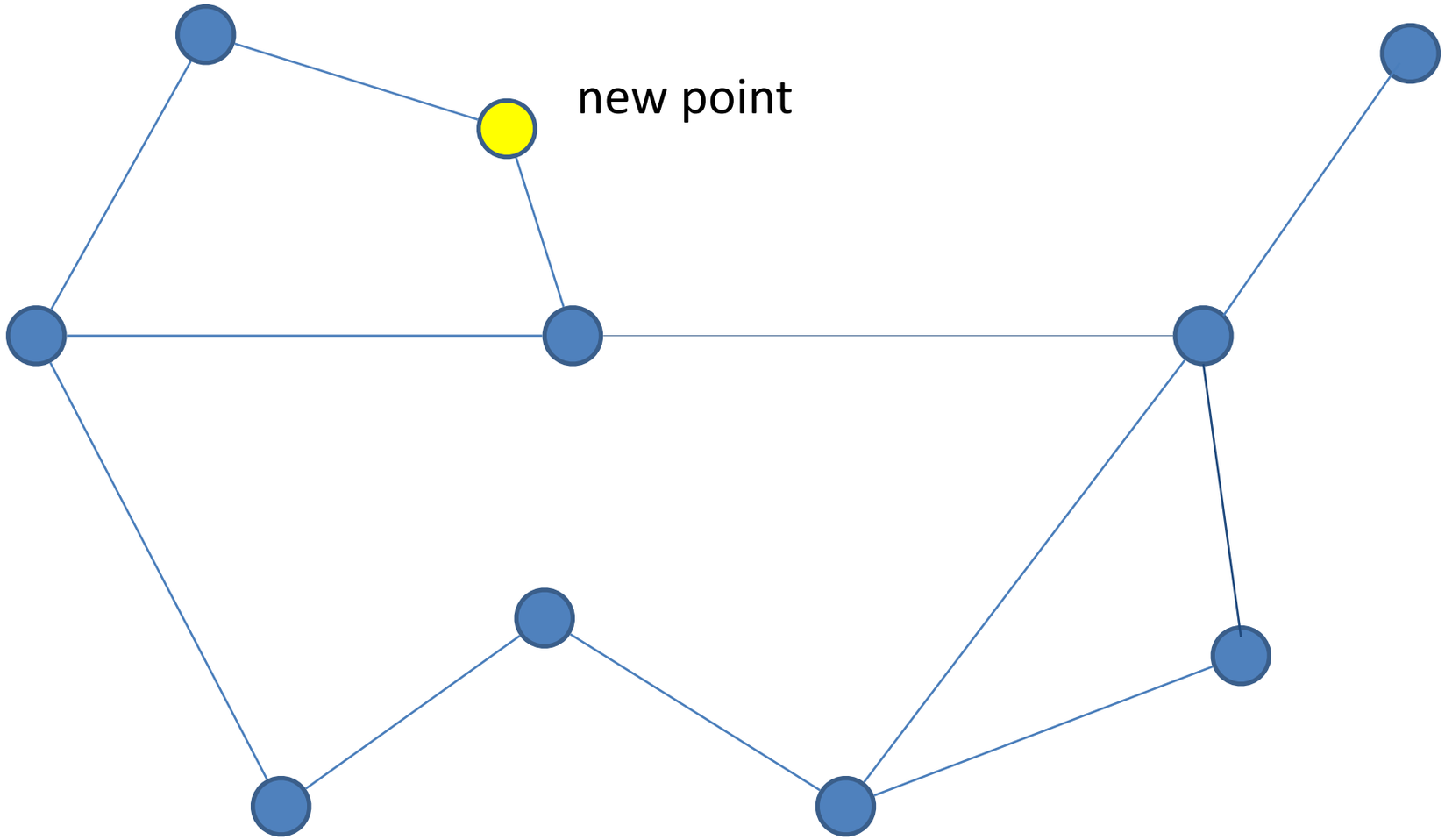
Remove local minimum by adding new edge

Local minimum



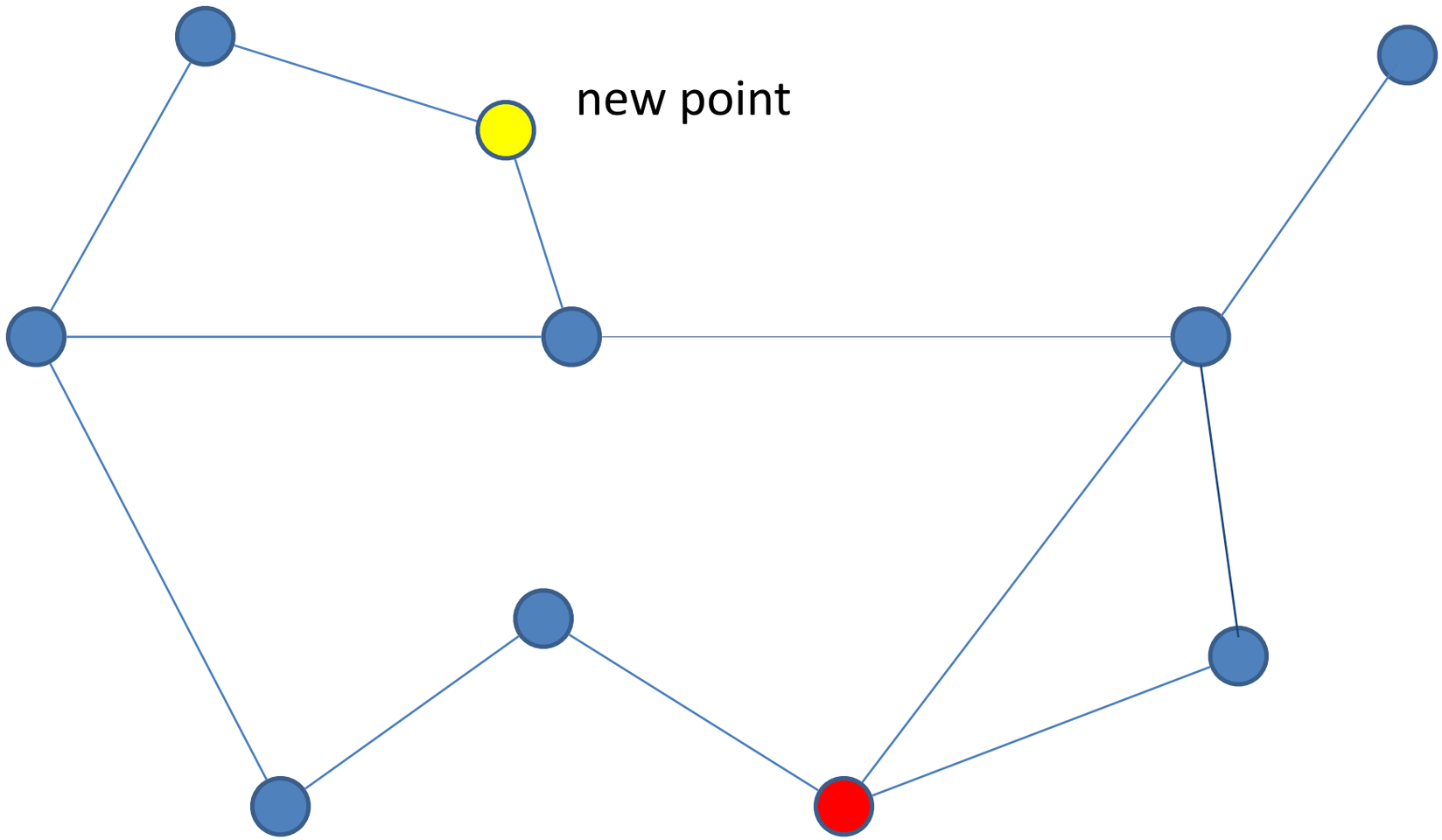
fail
success
fail

Lets do another attempt!



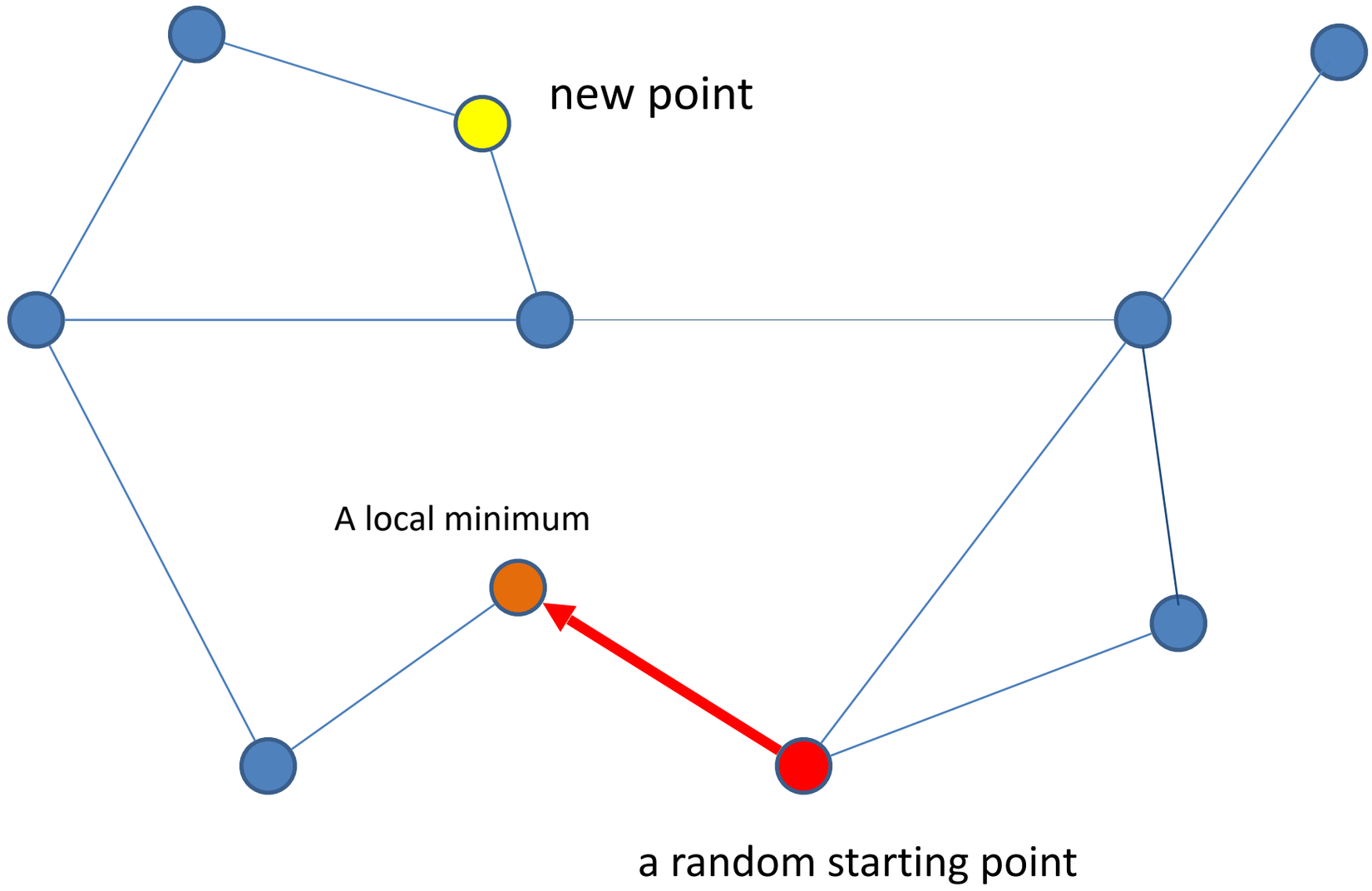
fail
success
fail

Select a random point again!



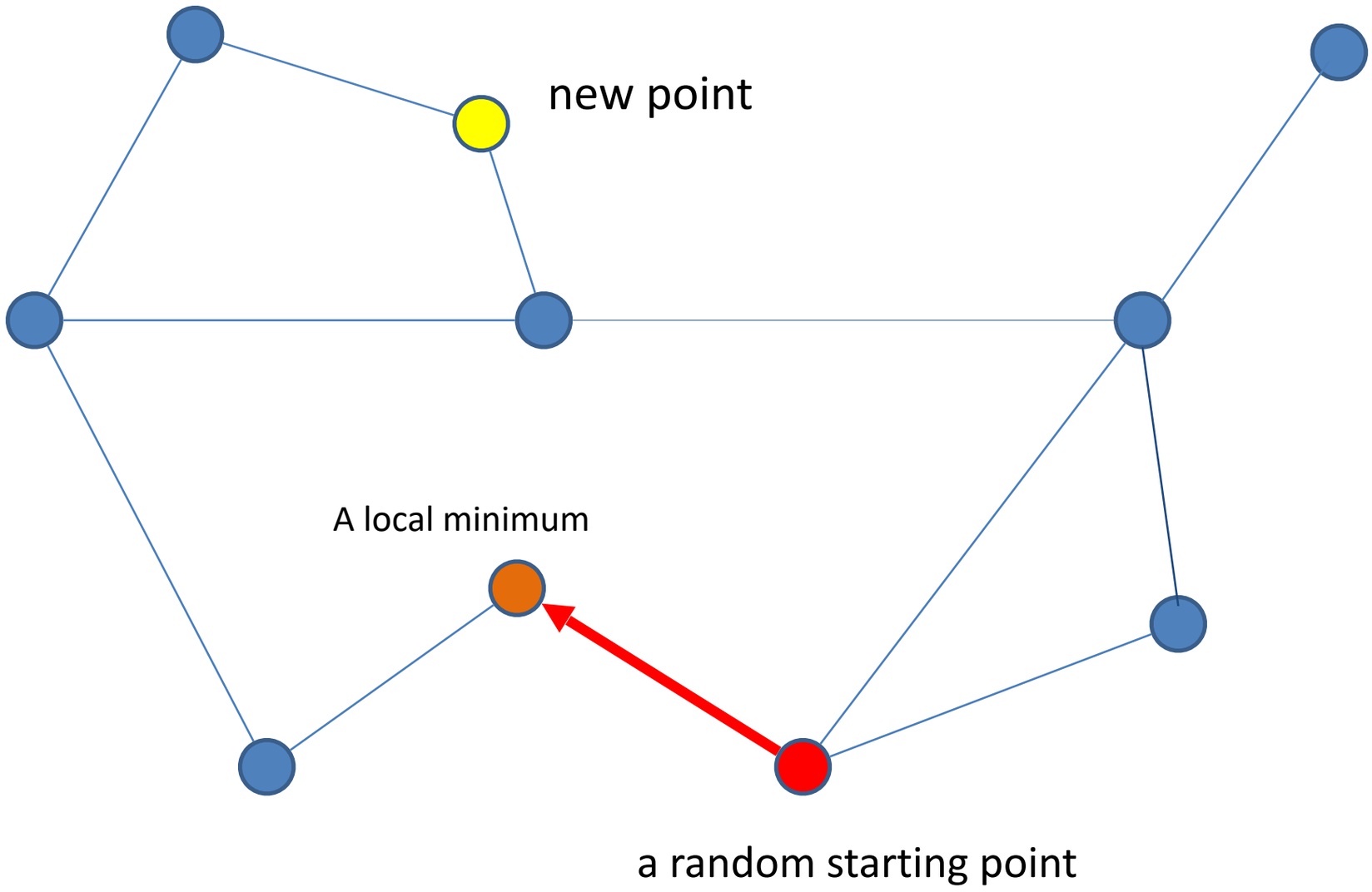
fail
success
fail

Perform a greedy walk



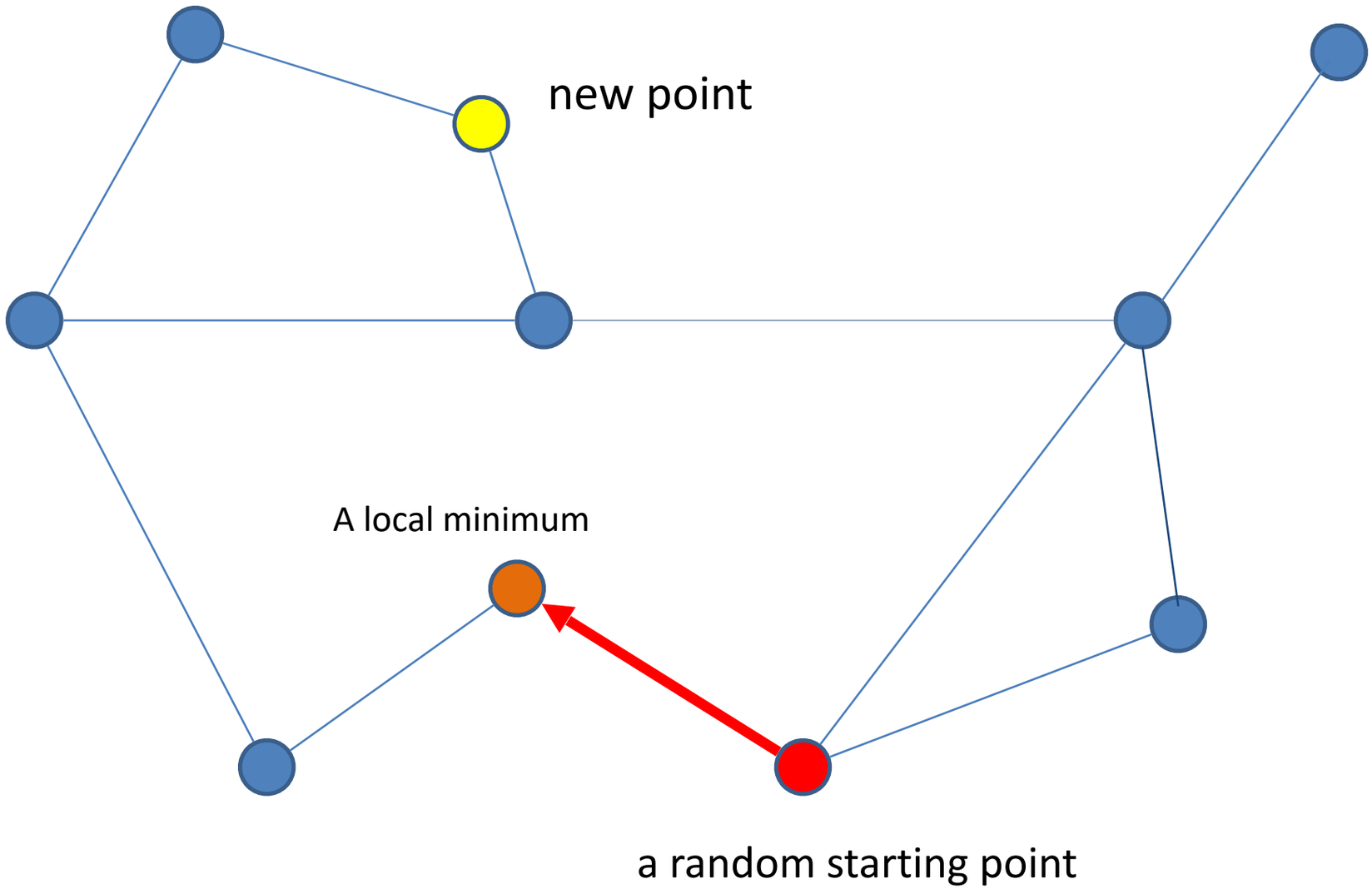
fail
success
fail

Mark that have fail



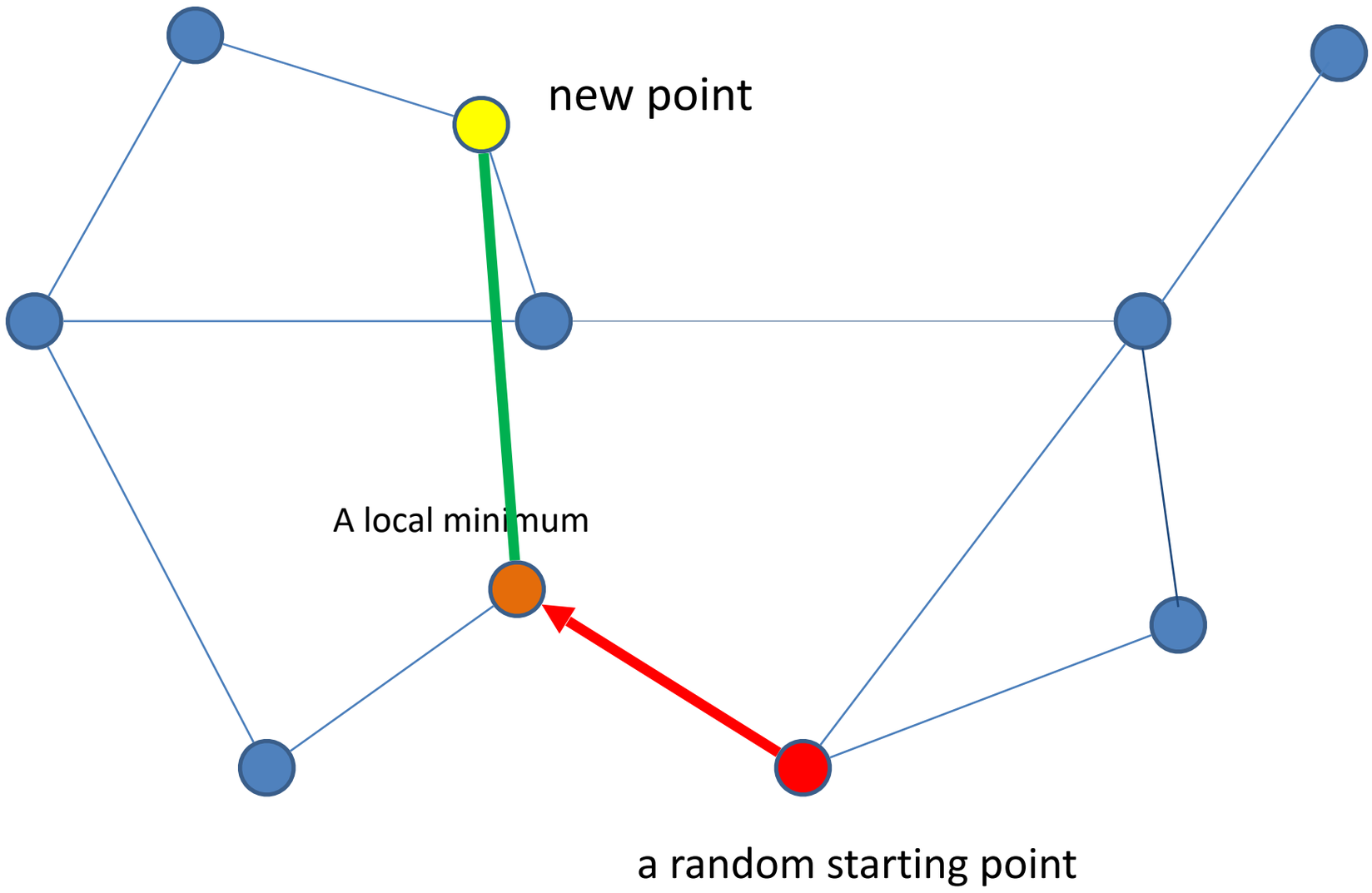
fail
success
fail
fail

Adding new edge to remove the local minimum



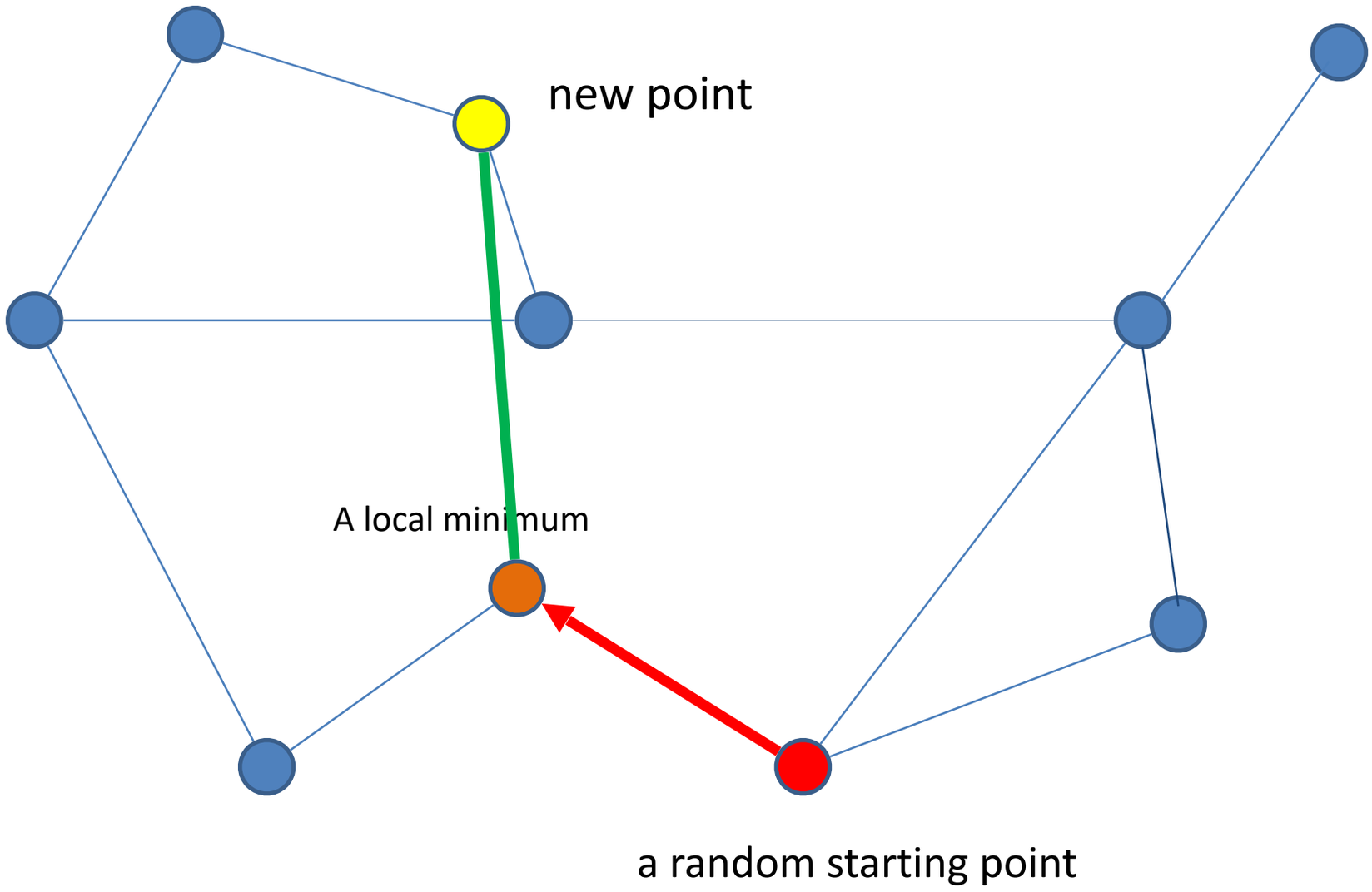
fail
success
fail
fail

This one?



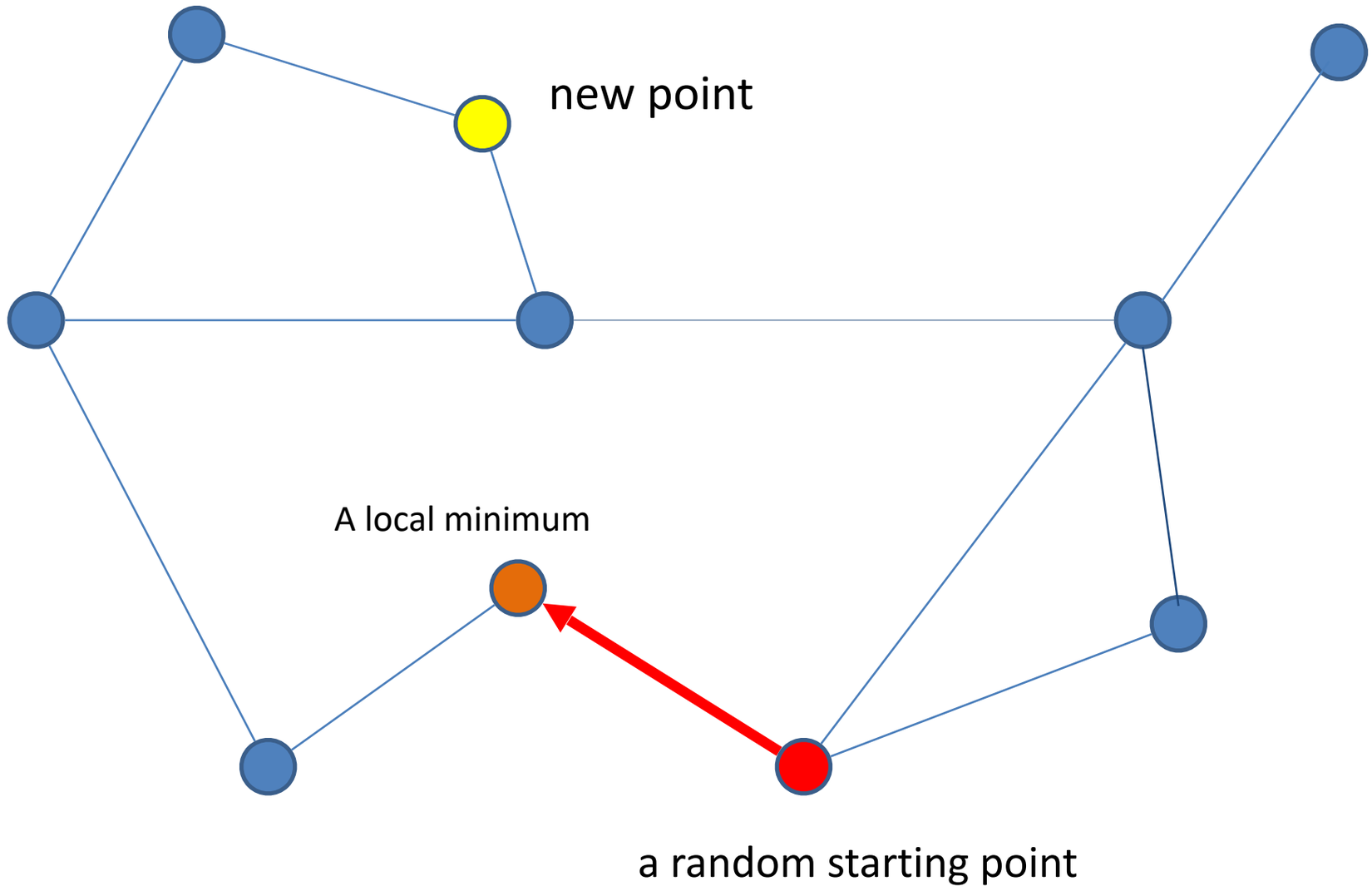
fail
success
fail
fail

Bad idea to add this edge!



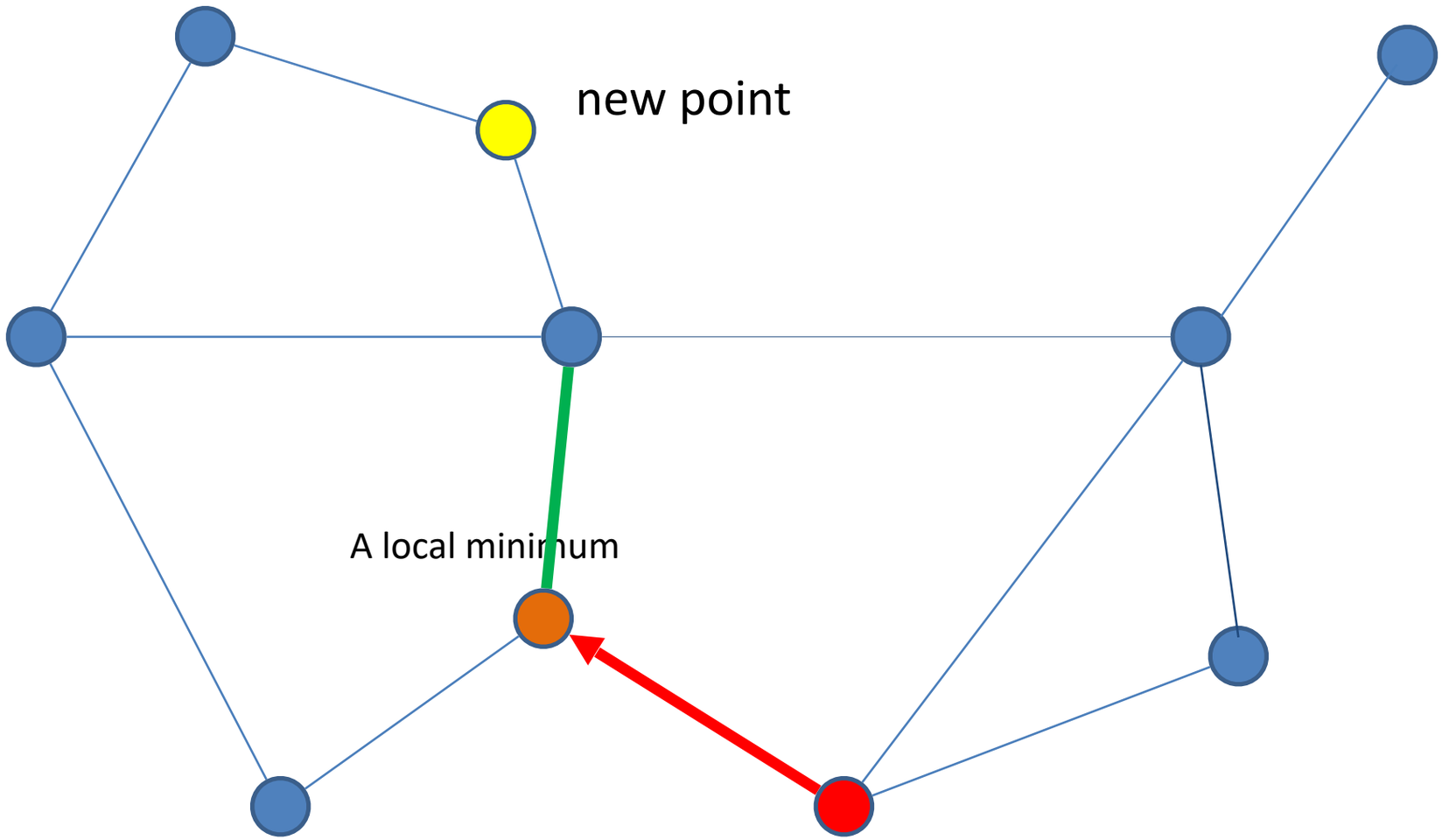
fail
success
fail
fail

Lets explore the neighborhood and the shortest edge that will remove the local minimum



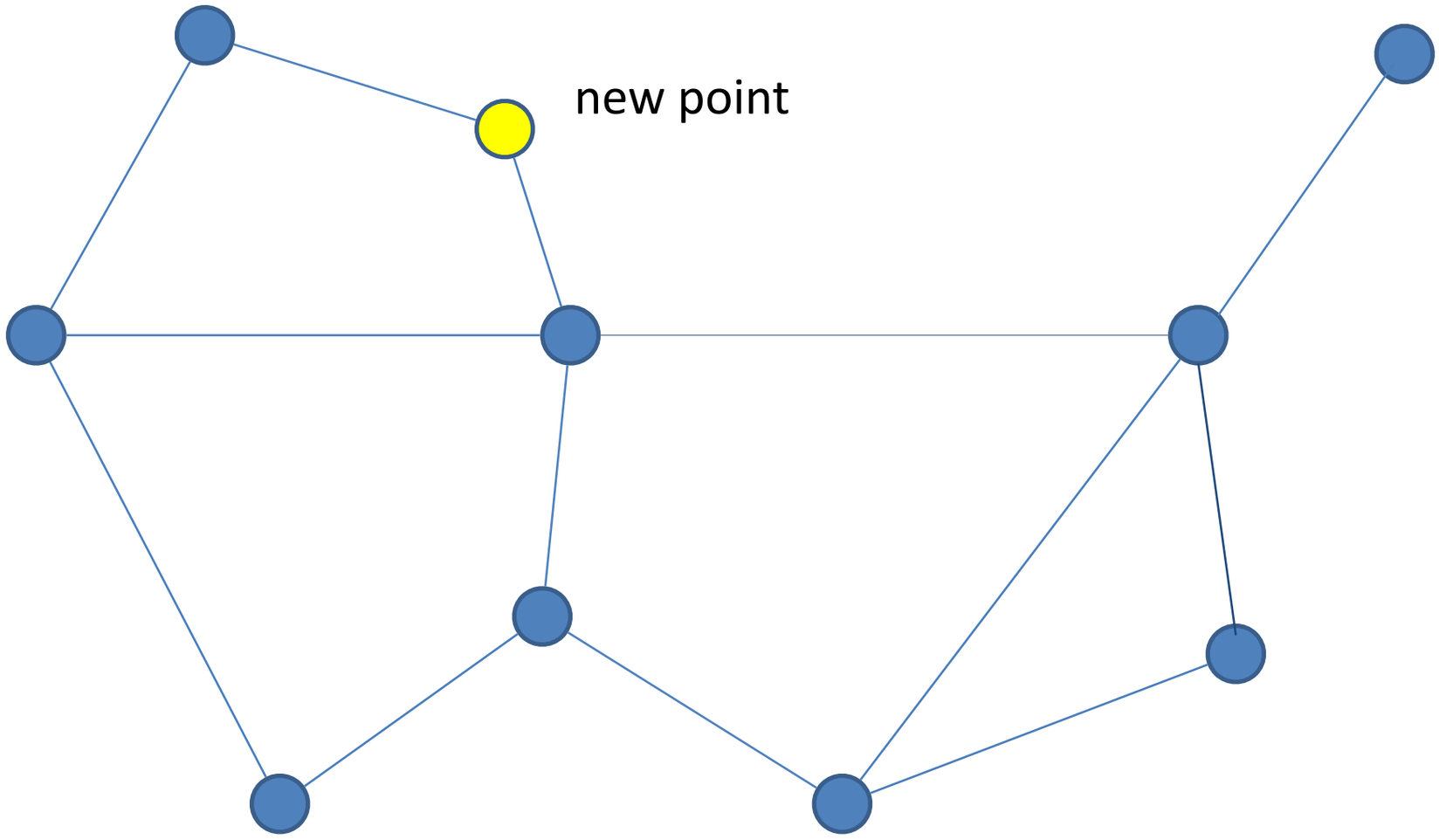
fail
success
fail
fail

This one!



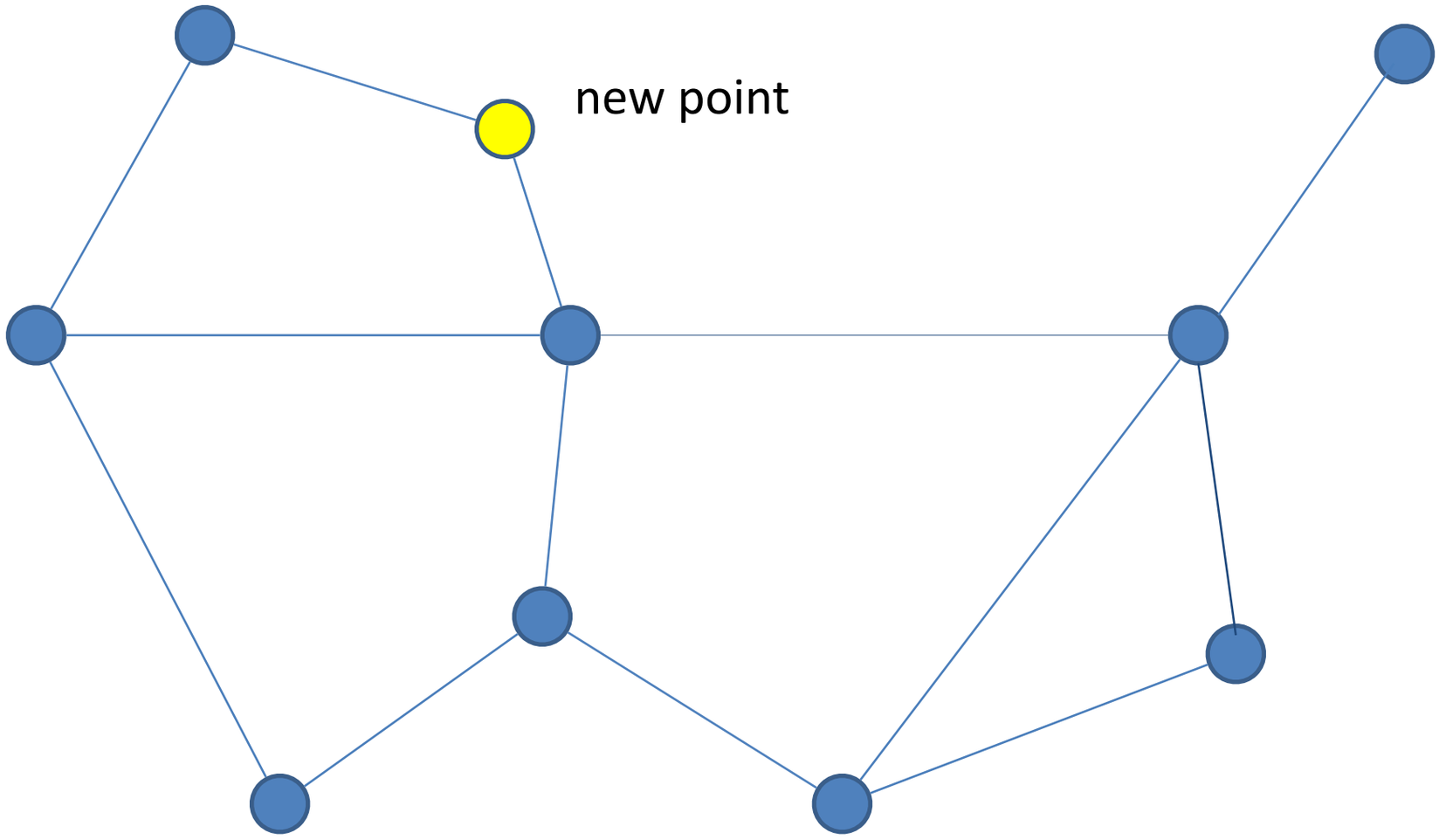
fail
success
fail
fail

Now our graph is much more better, isn’t it?



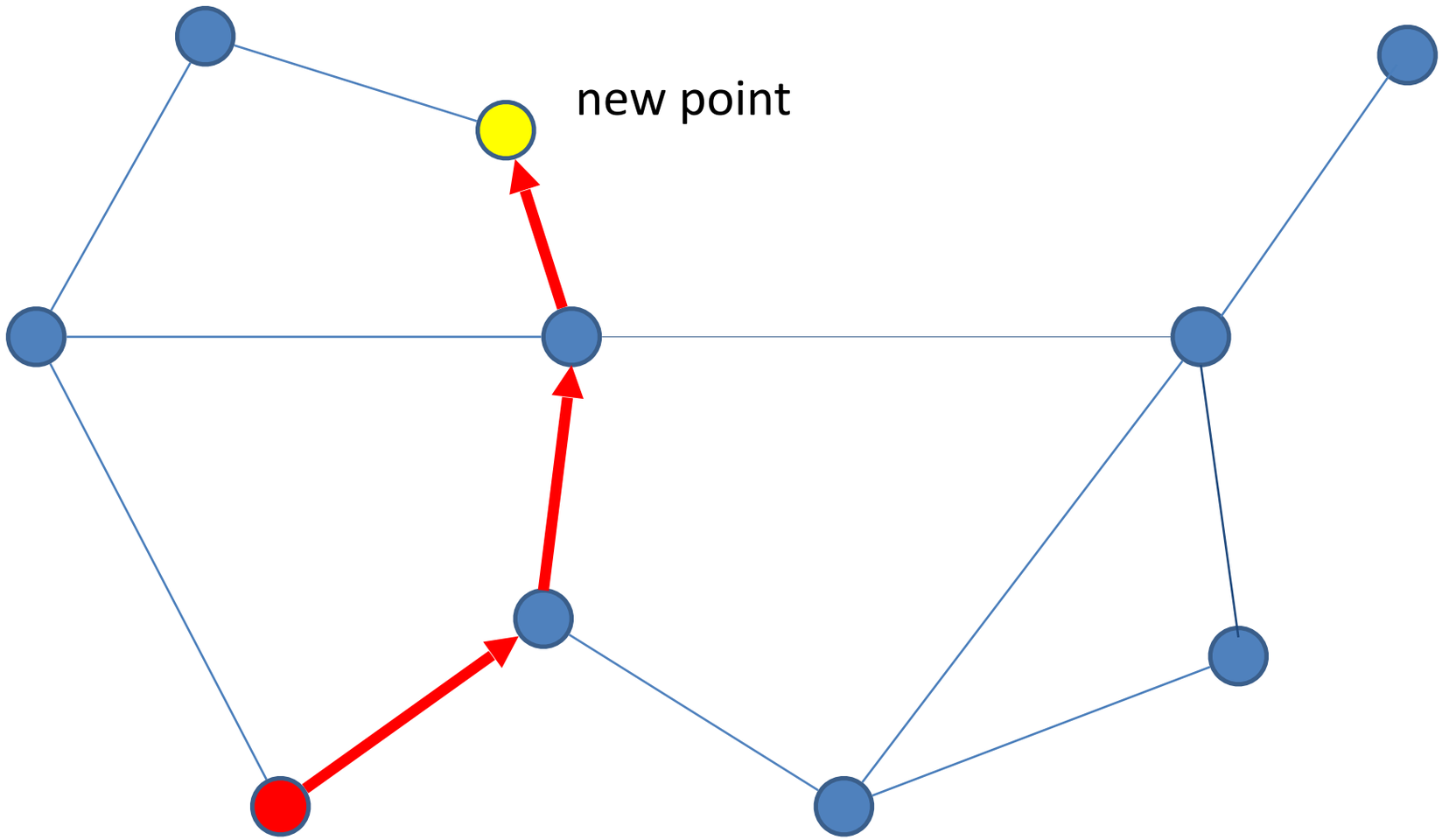
fail
success
fail
fail

Lets try to find a new point again!



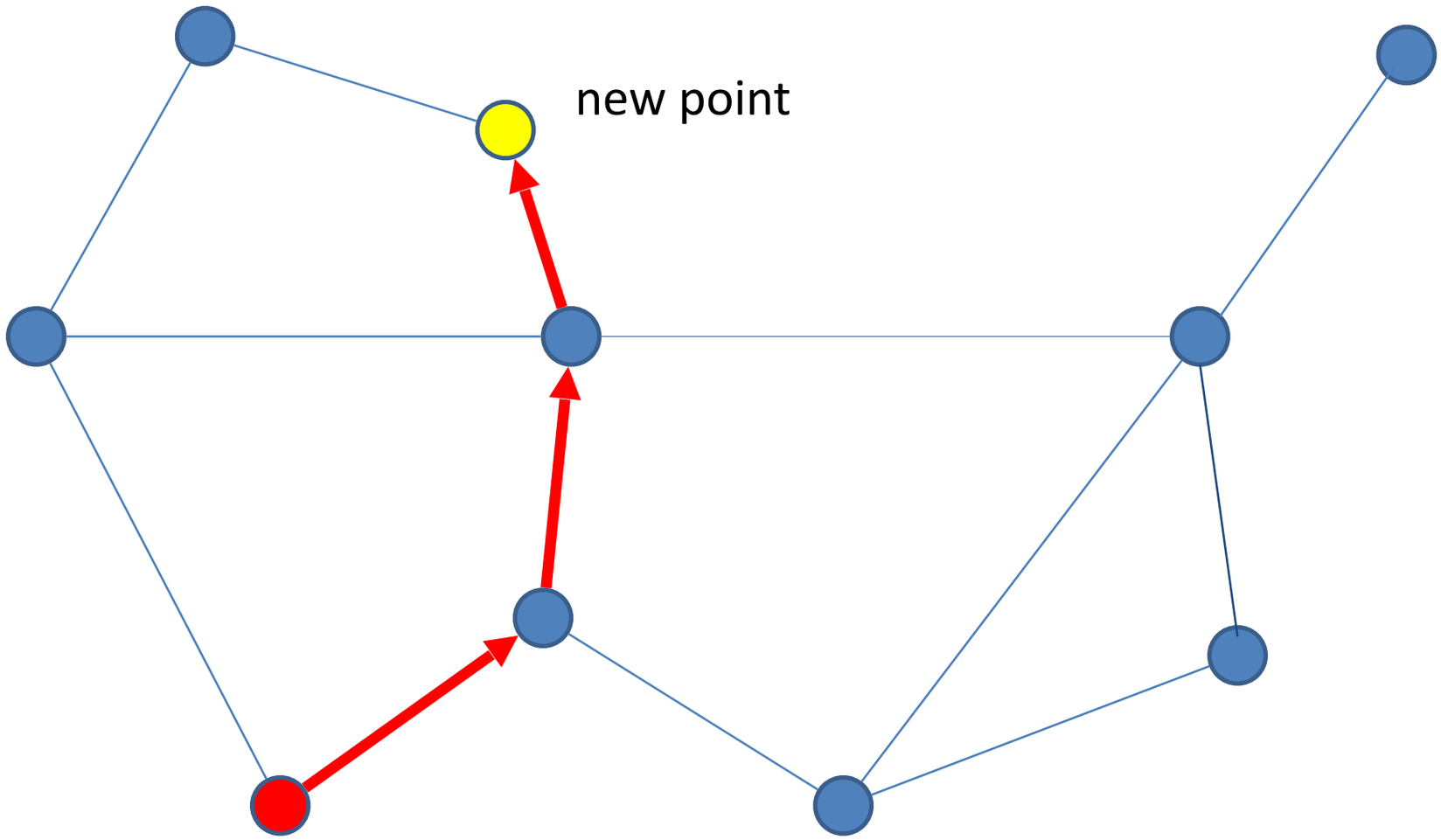
fail
success
fail
fail

Success!



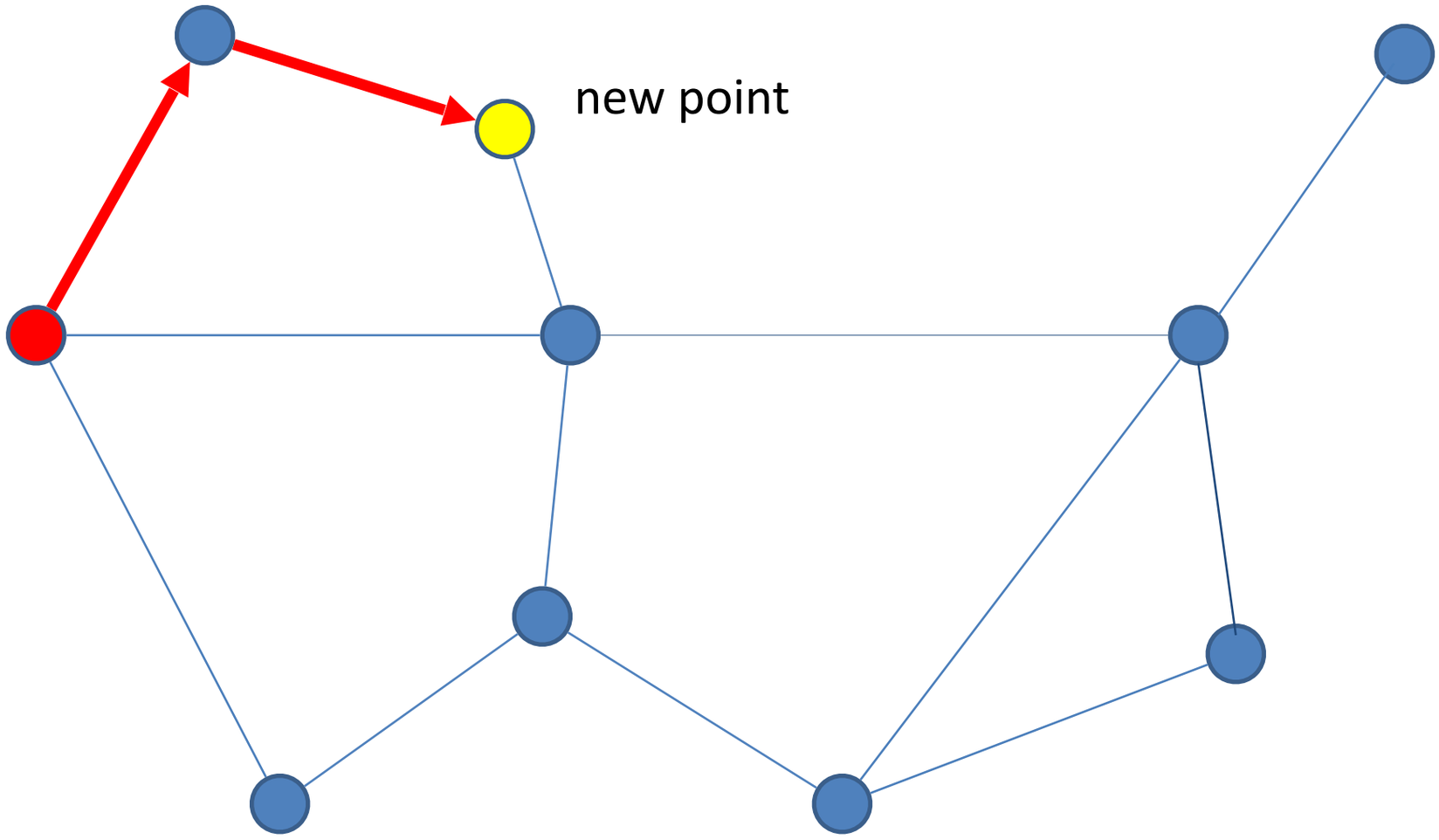
fail
success
fail
fail

Add new mark to the “success” table



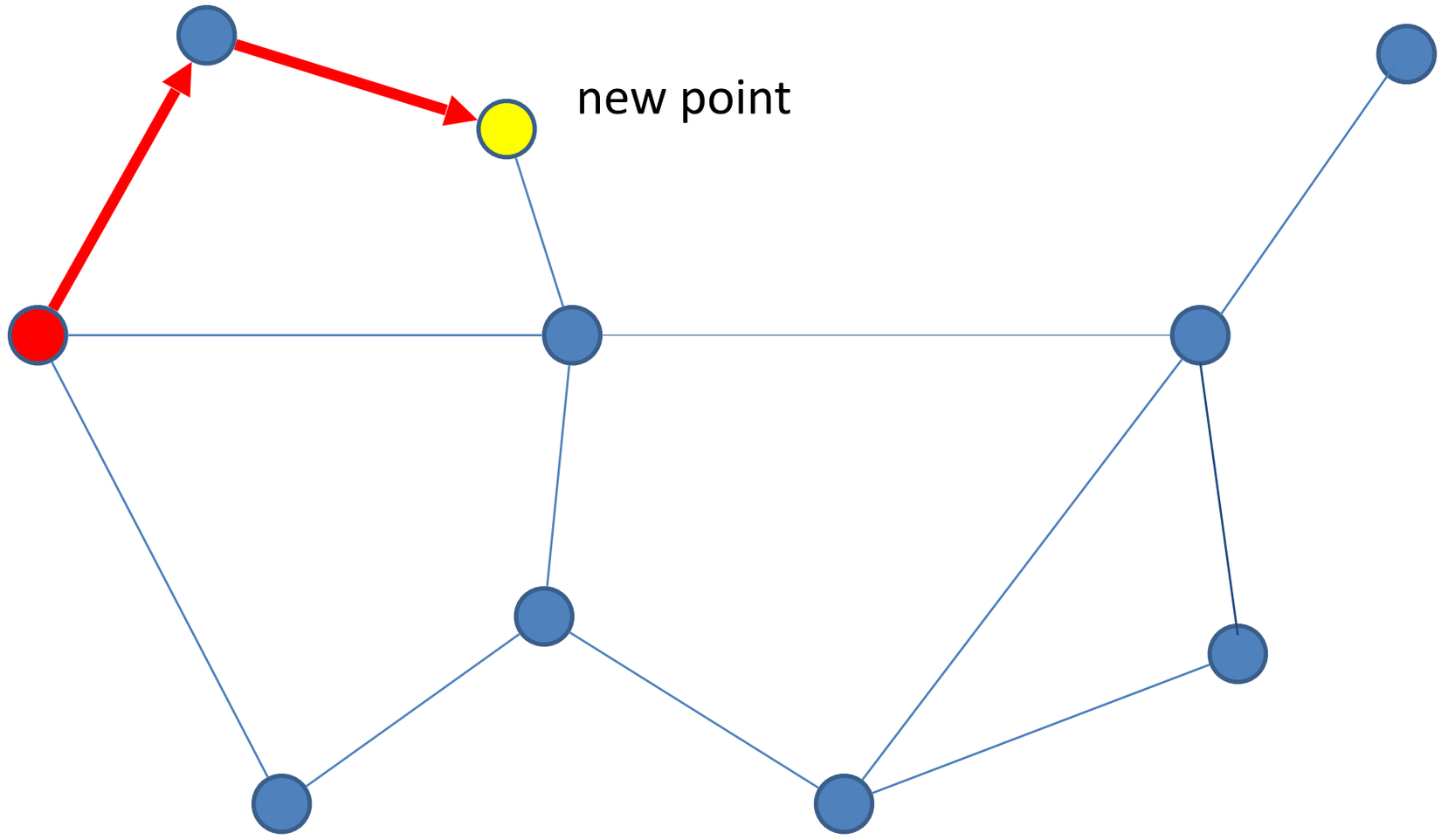
fail
success
fail
fail
success

Another attempt and success again!



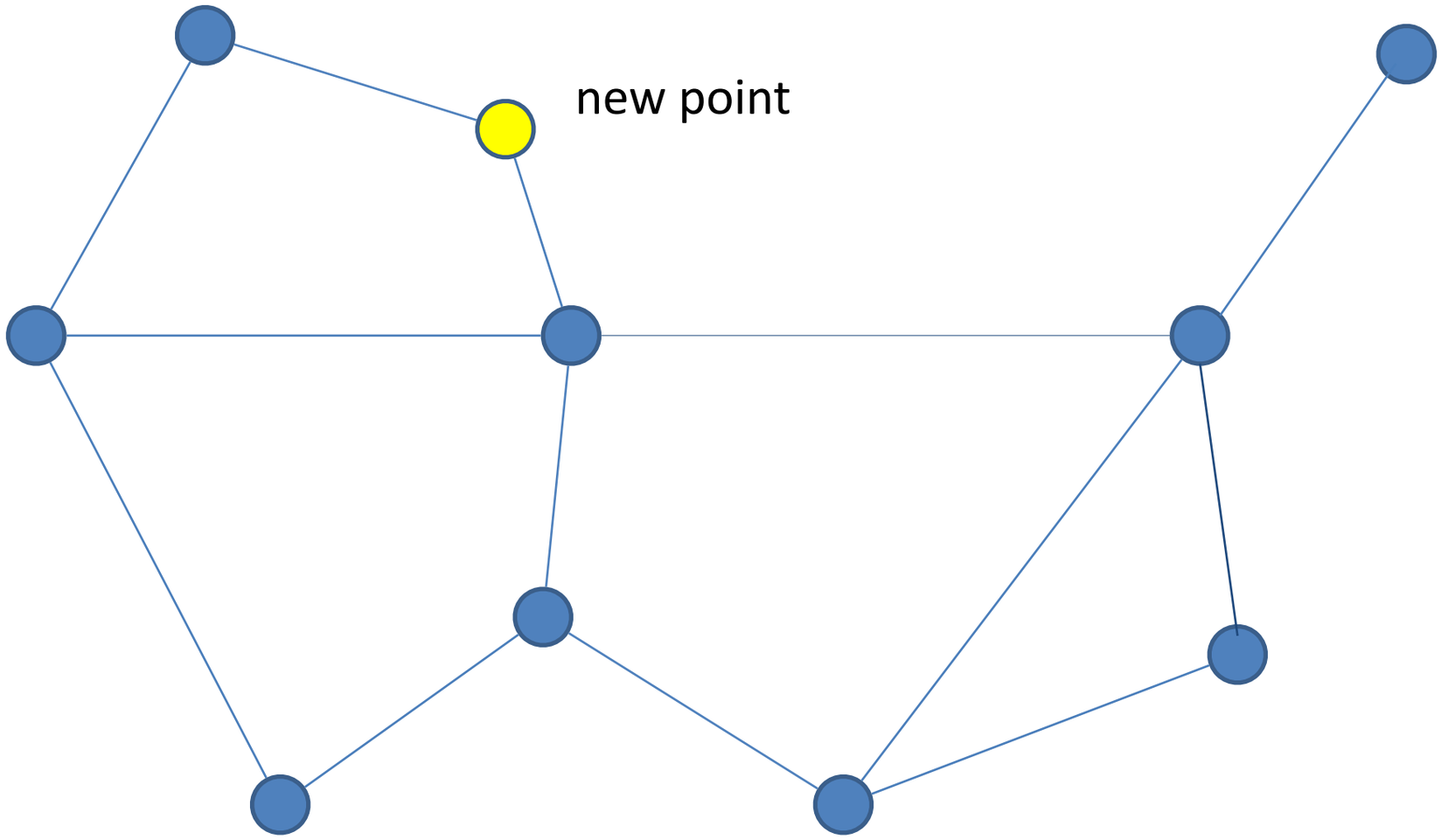
fail
success
fail
fail
success

Add record to the table



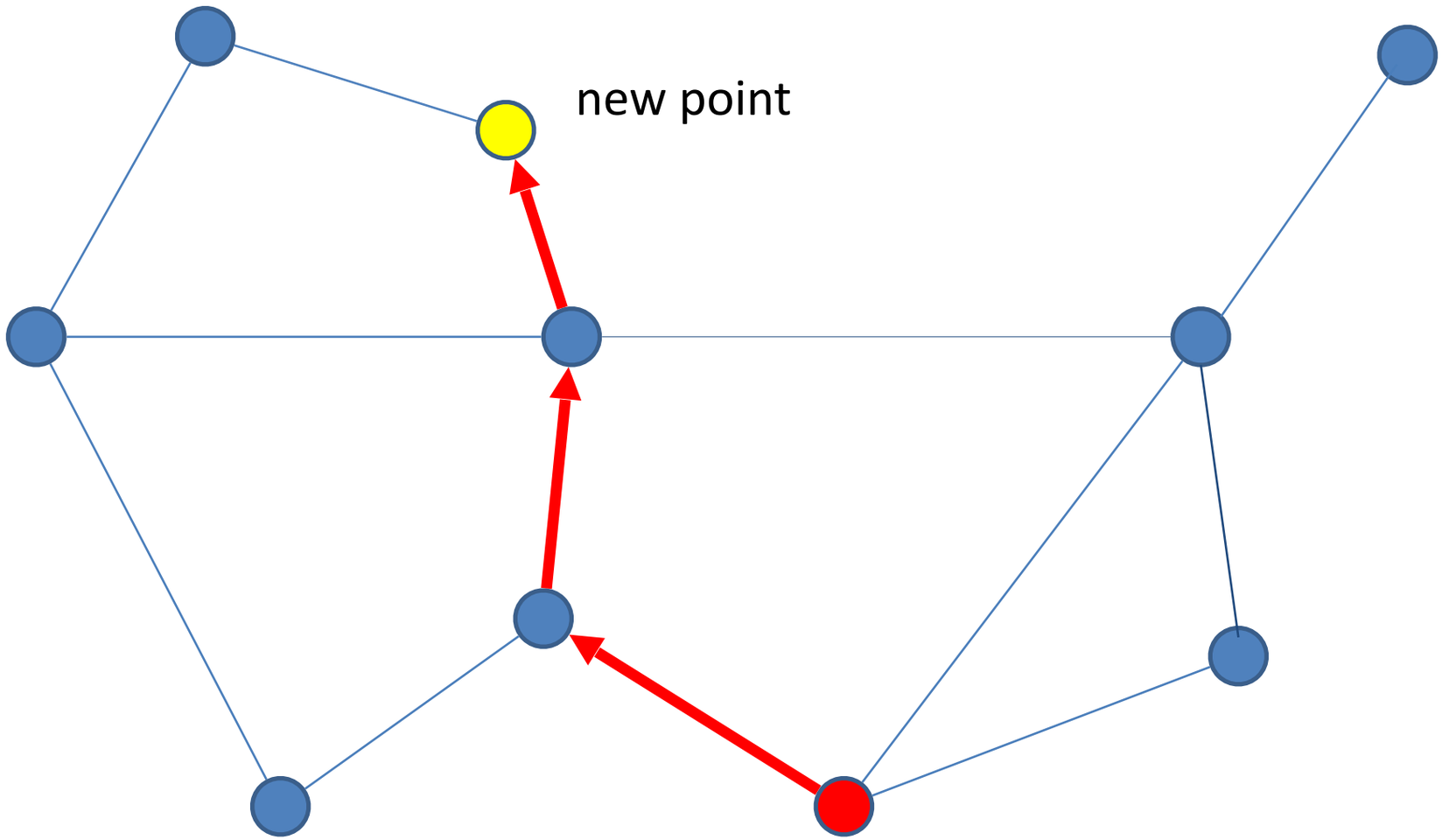
fail
success
fail
fail
success
success

One more new attempt so search a new point…



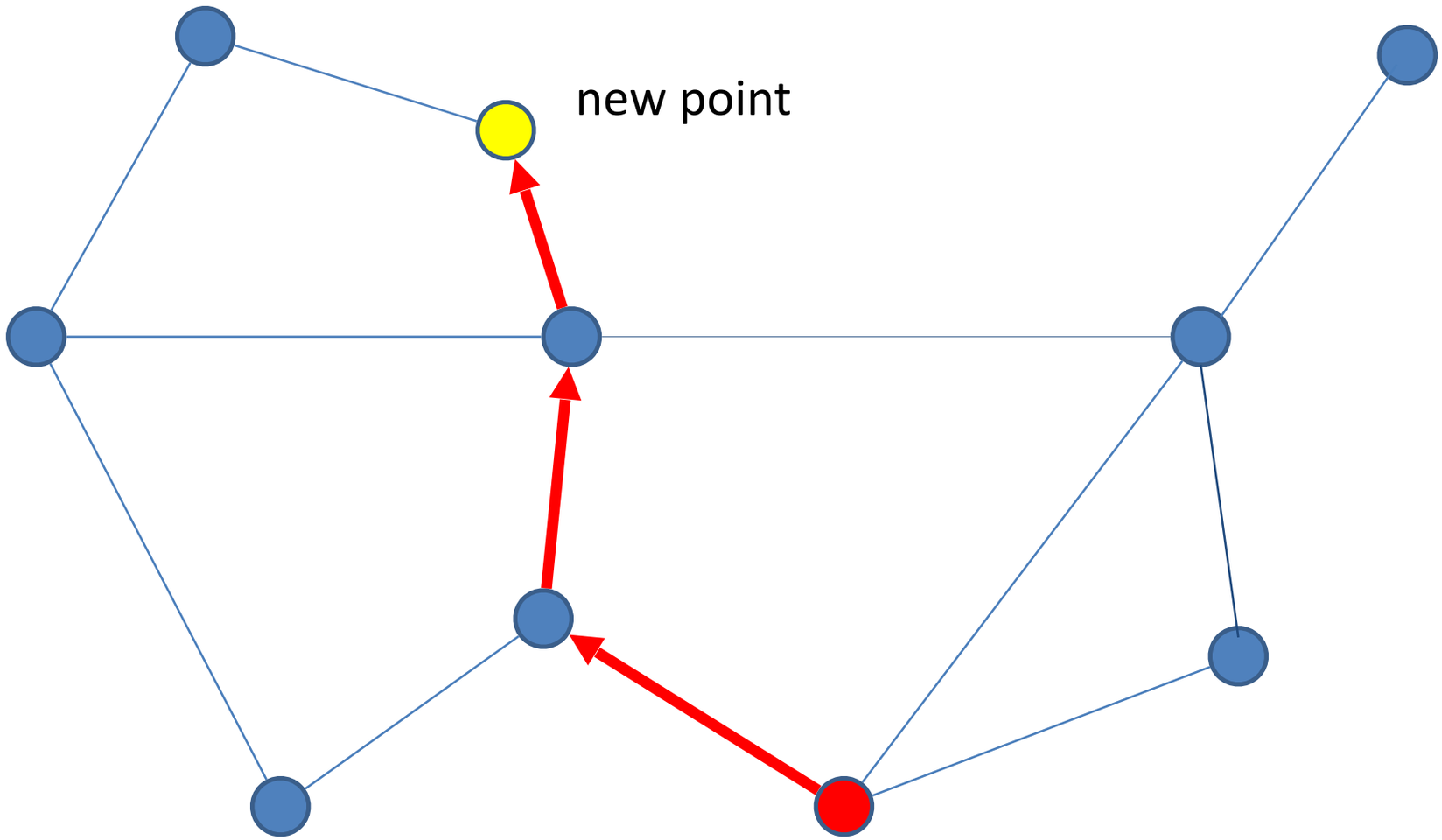
fail
success
fail
fail
success
success

Success!!!



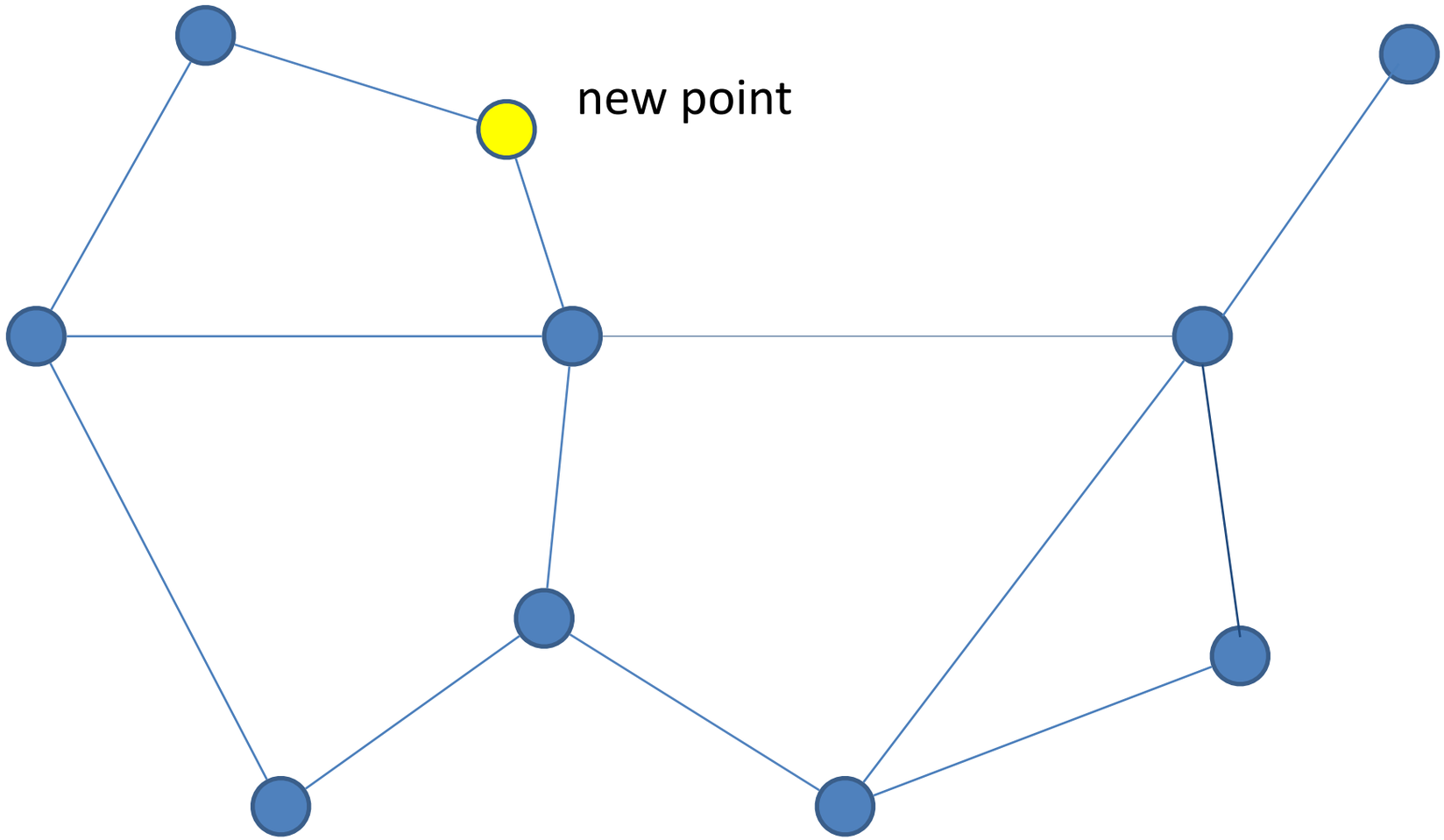
fail
success
fail
fail
success
success

Add record to “success” table



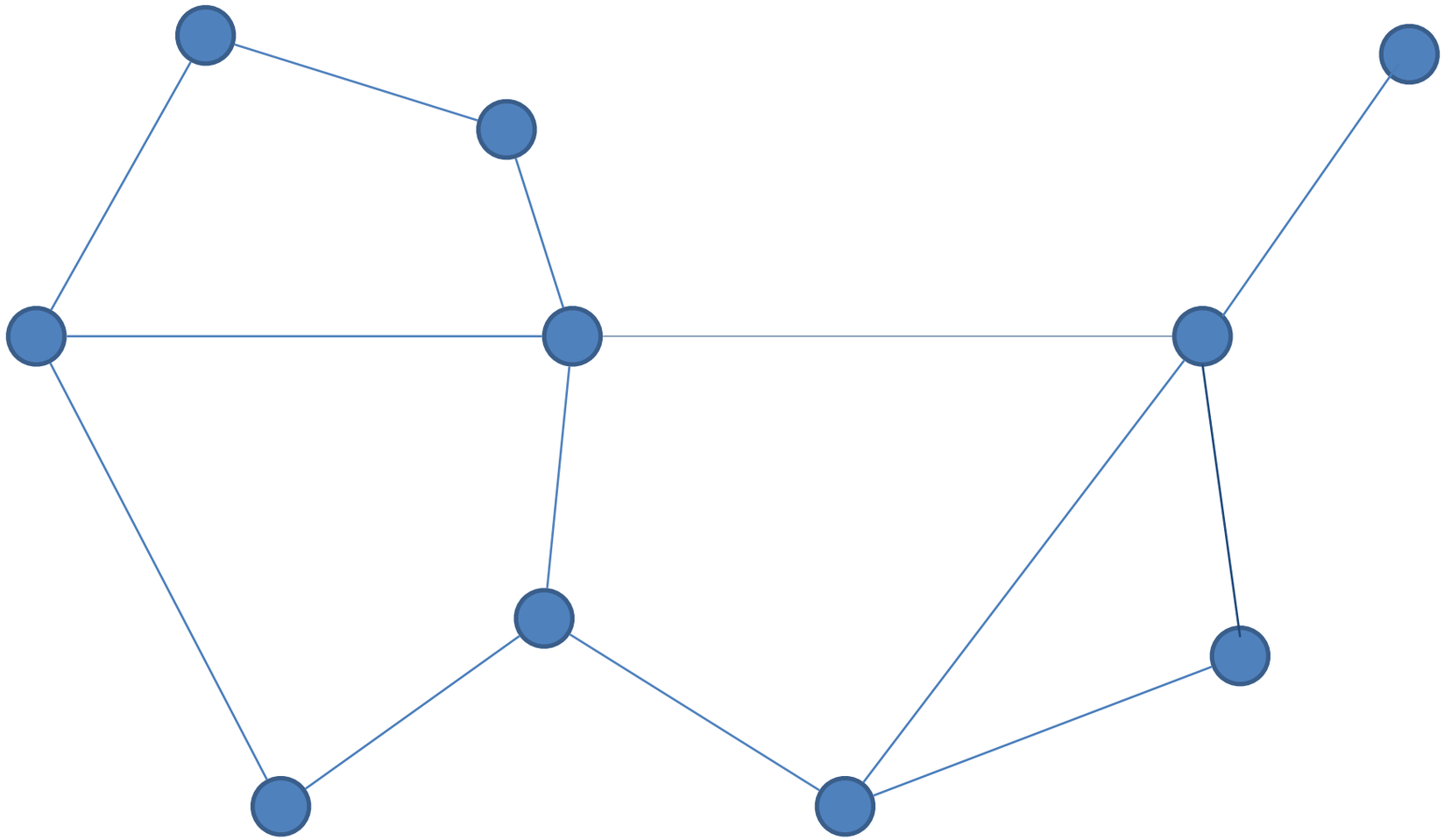
fail
success
fail
fail
success
success
success

3 successes in a row!



fail
success
fail
fail
success
success
success

Stop insertion procedure.



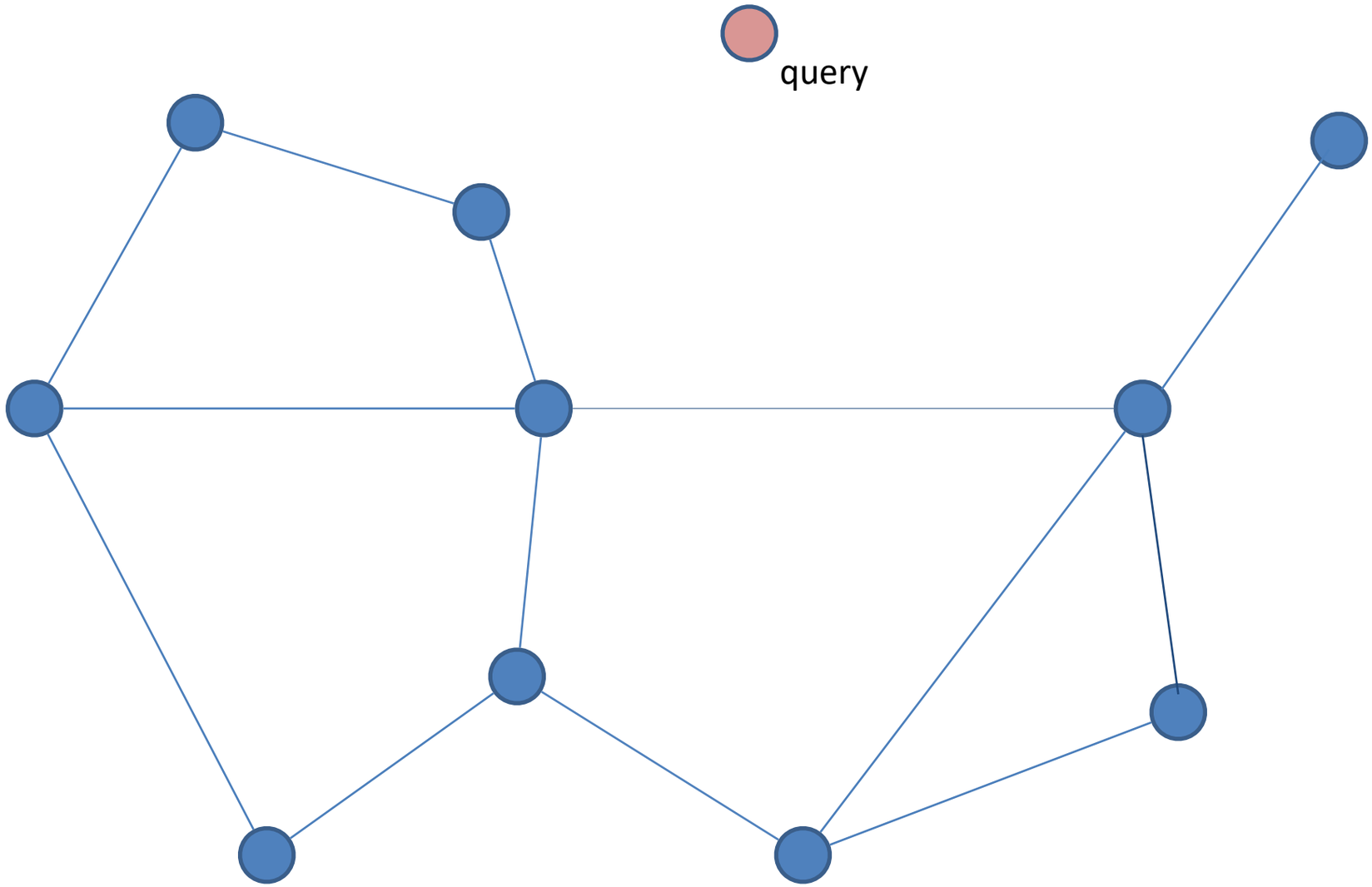
fail
success
fail
fail
success
success
success

Self-adaptive Graph Construction Algorithm

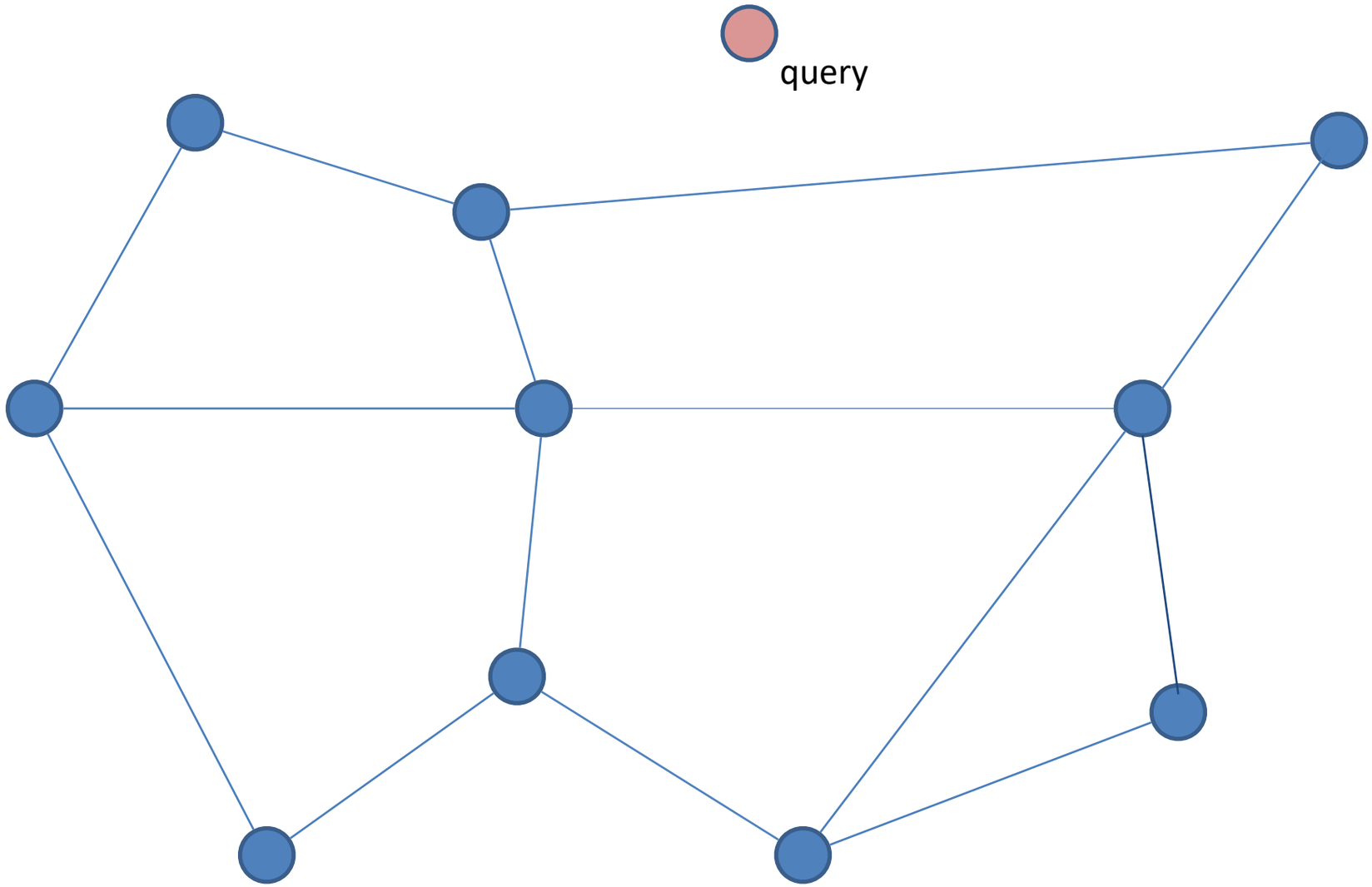
```
Get_Local_Minimums(q ∈ D, G(V,E), τ ∈ N)
01 L ← {}; τ' ← 0
02  while τ' < τ do
03    v_start ← Random(V) //put to v_start random vertex from V
04    v, P ← Greedy_Walk(q, G, v_start)
05    if v ∉ L then τ' ← 0; L ← L ∪ v
08    else τ' ← τ'+1
09  end while
11  return L, P
```

```
Insert_By_Repairing(x ∈ X, G(V,E), τ ∈ N)
01   V ← V ∪ x; P ← {}
02   L, P ← Get_Local_Minimums(x, G, τ)
03   P ← P ∪ x
04   for each z ∈ L do
05     P' ← {y ∈ P : d(y,x) < d(z,x)}
06     x' ← arg min_{y∈P'} (d(z,y))
07     E ← E ∪ (x',z) ∪ (z,x')
08   end for
```

The same we can do with queries, but without embedding query to the graph



This is the way how we can improve the data structure by queries



Query-based Improvement Procedure

Repair_By_Query($q \in D$, $G(V,E)$, $\tau \in \mathbb{N}$)

01 $L, P \leftarrow$ Get_Local_Minimums(q, G, τ)

02 $x \leftarrow \arg\min_{y \in L}(d(y,q))$

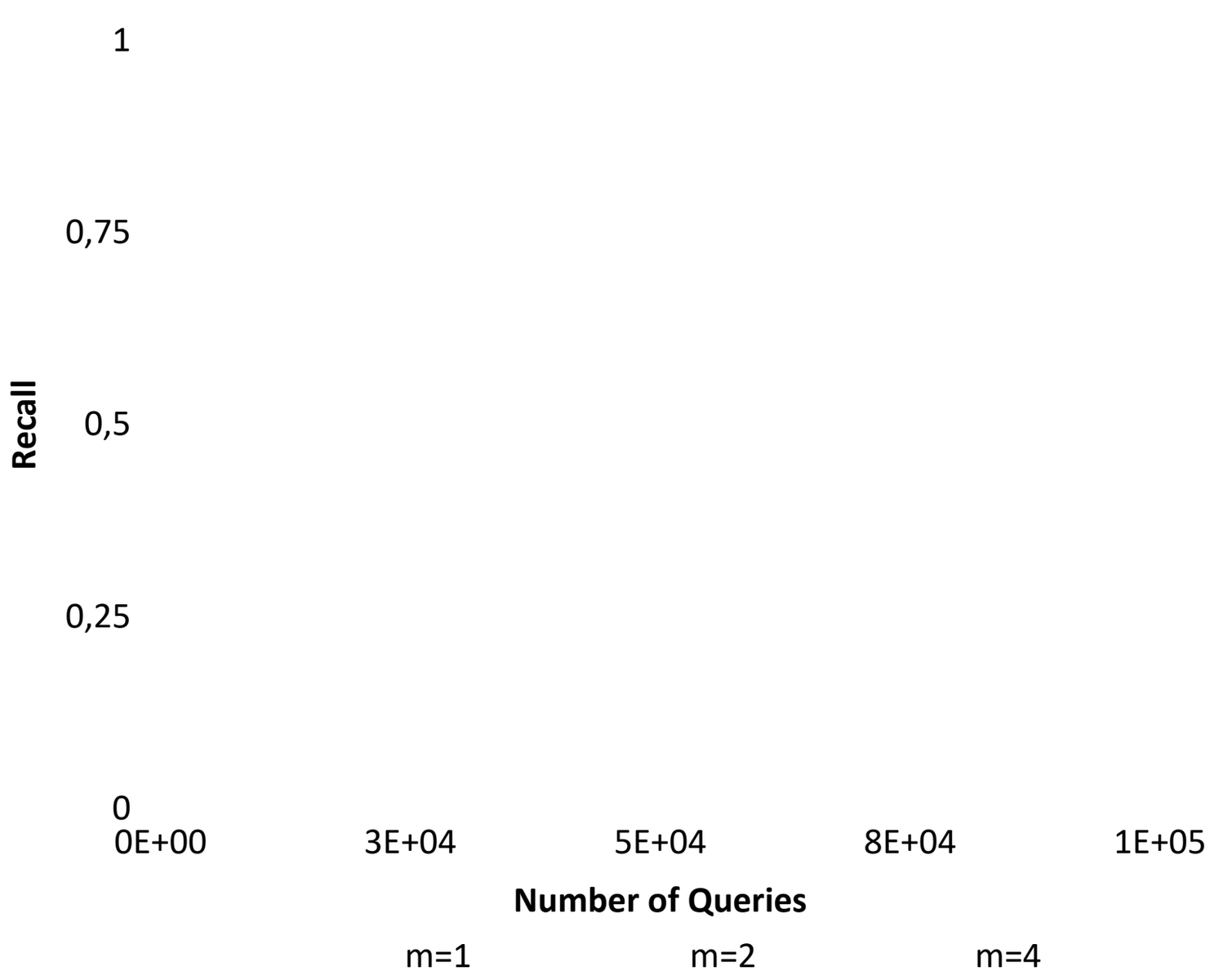
03 for each $z \in L \setminus x$ do

04 $P' \leftarrow \{y \in P : d(y,q) < d(z,q)\}$

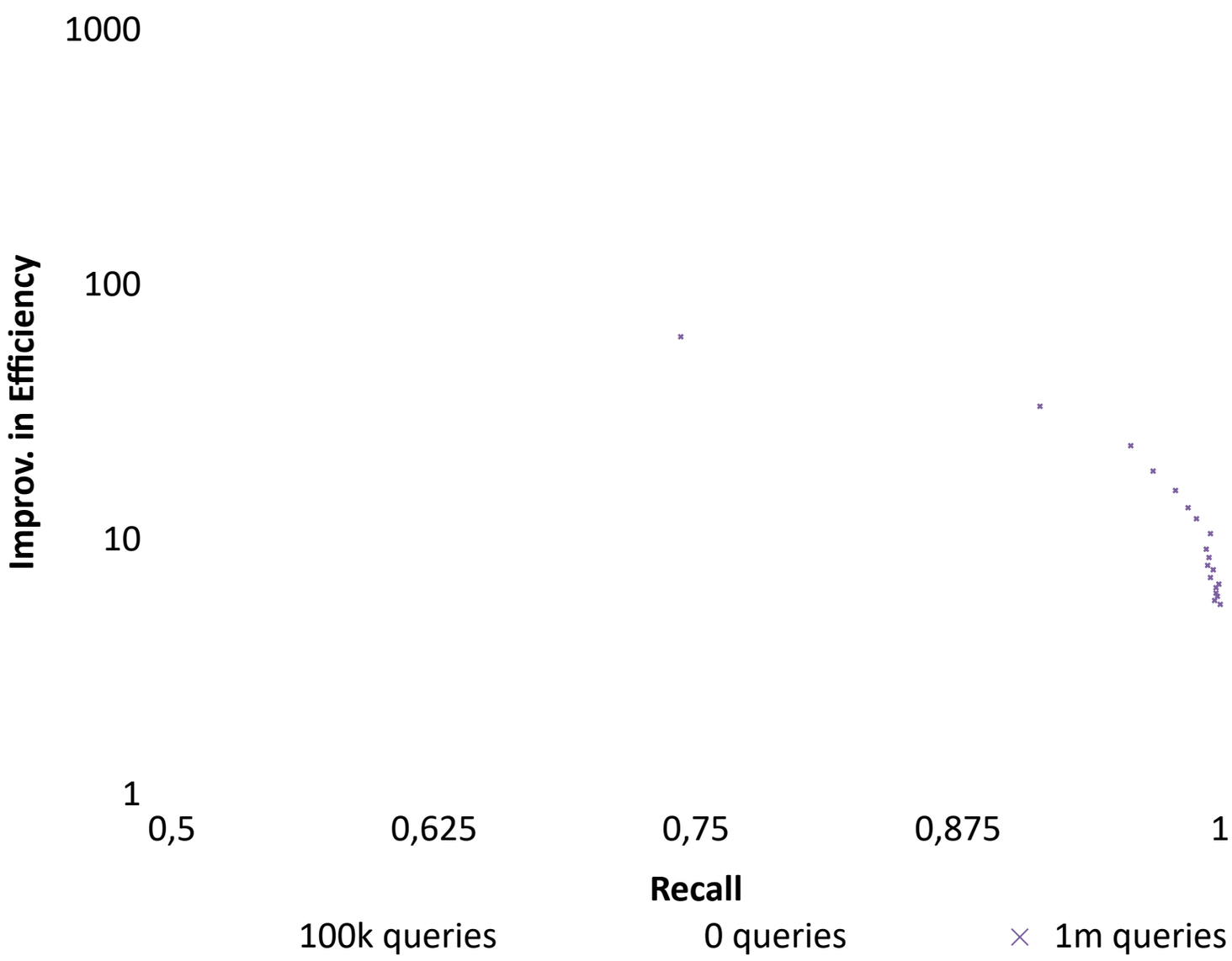
05 $x' \leftarrow \arg\min_{y \in P'}(d(z,y))$

06 $E \leftarrow E \cup (x',z) \cup (z,x')$

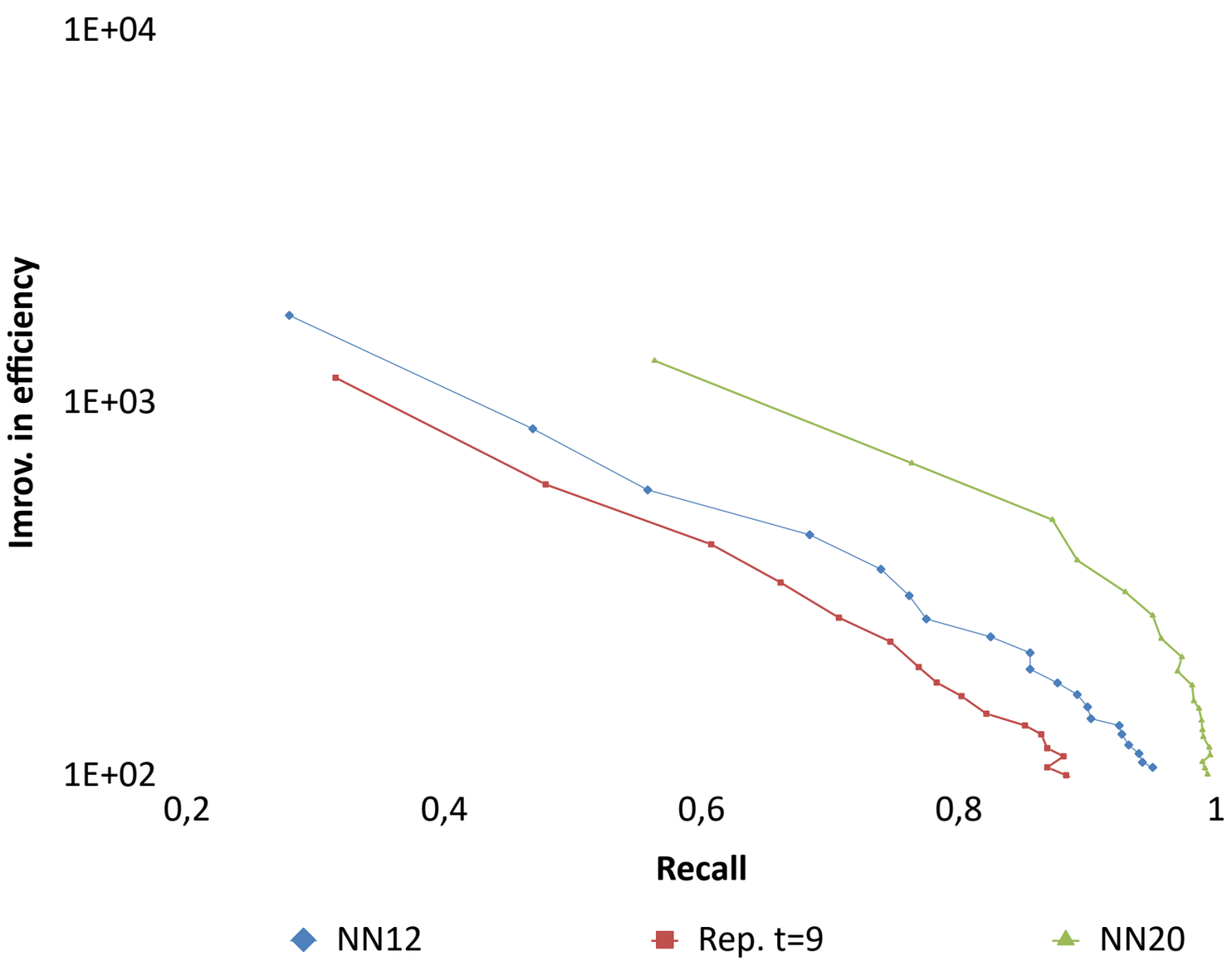
07 end for



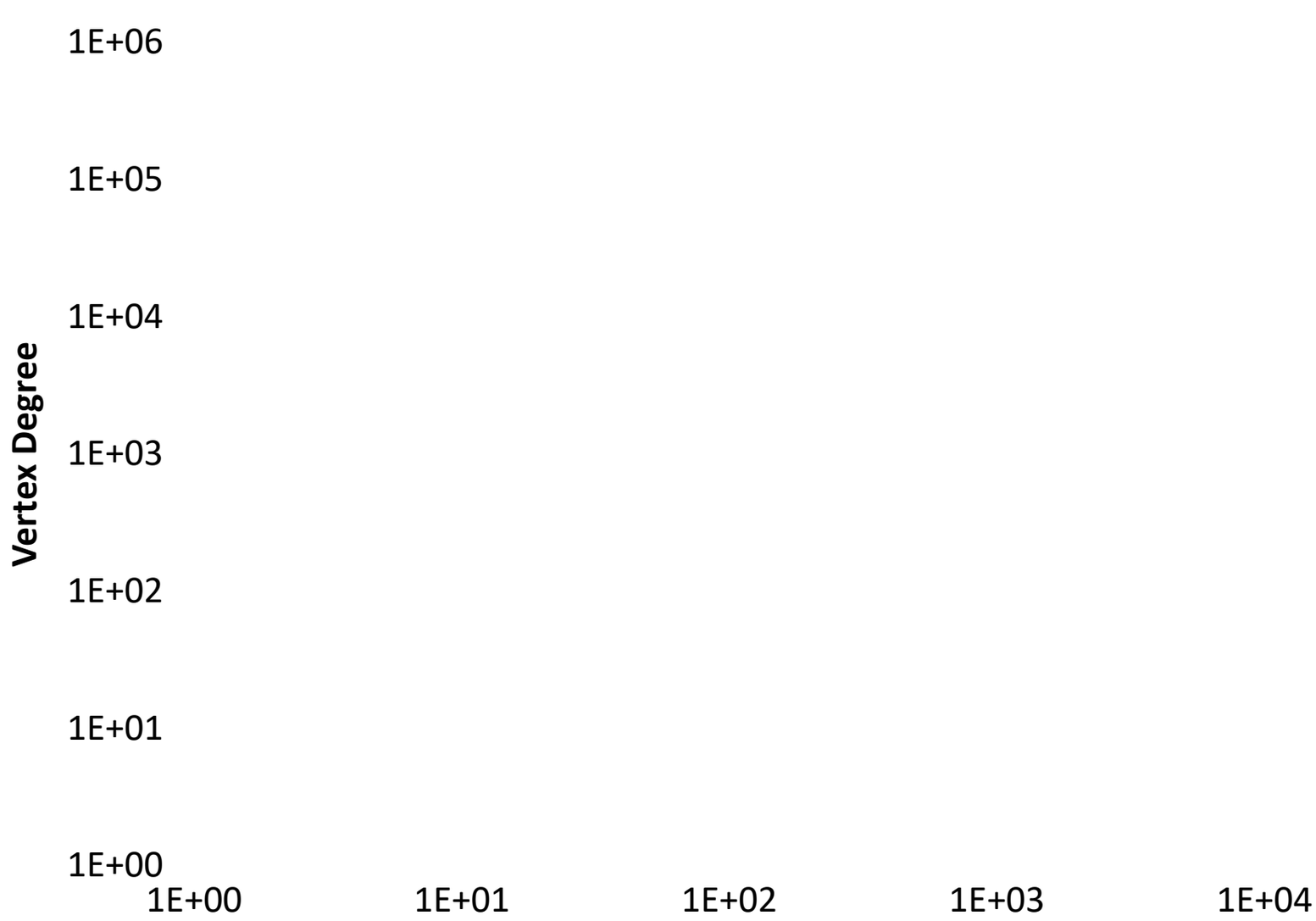
Recall after Improving by queries applying Repair_By_Query procedure



The overall efficiency of the structure after applying Repair_By_Query procedure



Comparison with NN

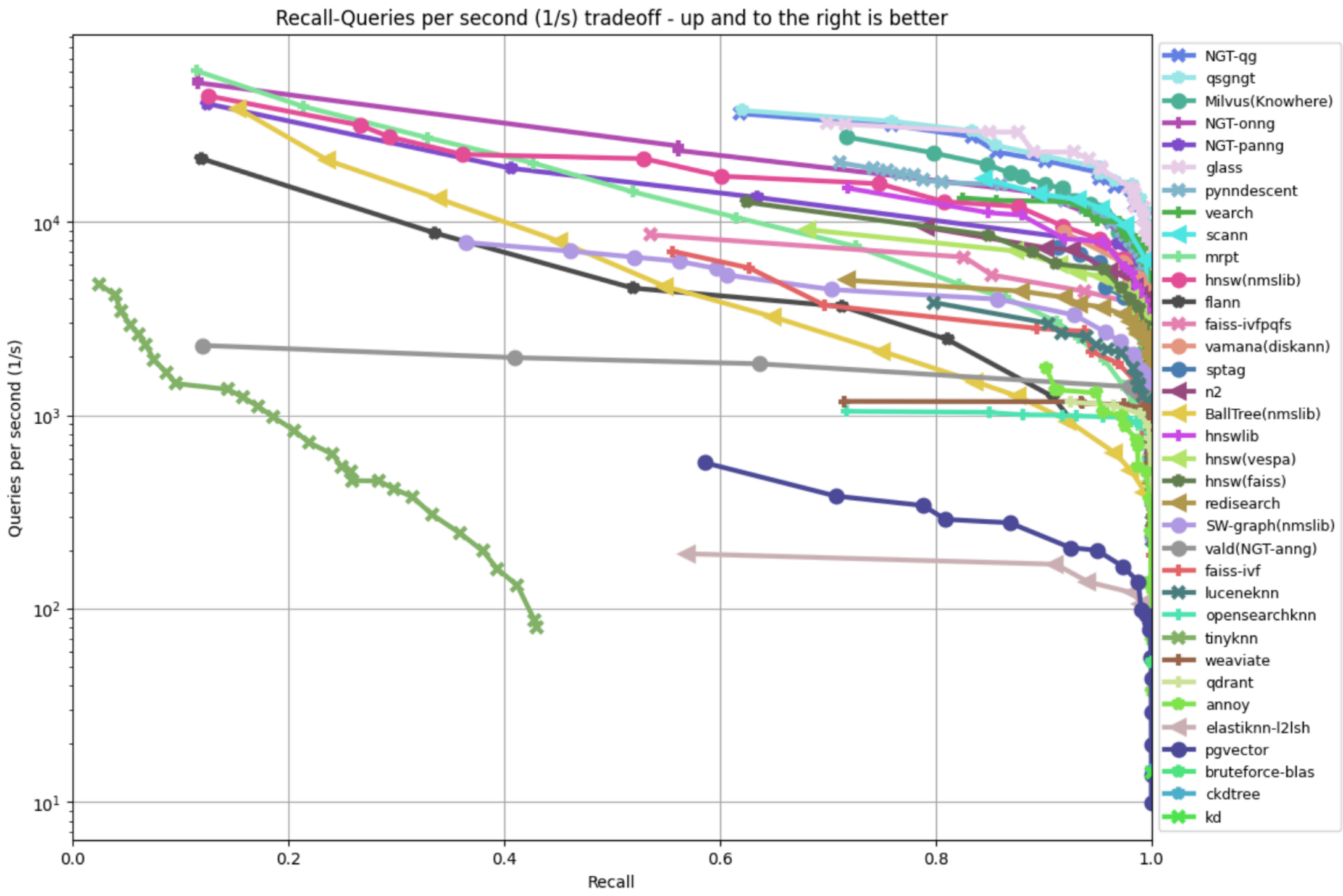


Vertex Degree Distribution

Summary

Algorithm	Base Graph	Edge	Build Complexity	Search Complexity
KGraph [31]	KNNG	directed	$O(\|S\|^{1.14})$	$O(\|S\|^{0.54})^{\ddagger}$
NGT [46]	KNNG+DG+RNG	directed	$O(\|S\|^{1.14})^{\ddagger}$	$O(\|S\|^{0.59})^{\ddagger}$
SPTAG [27]	KNNG+RNG	directed	$O(\|S\| \cdot \log(\|S\|^c + t^t))^{\dagger}$	$O(\|S\|^{0.68})^{\ddagger}$
NSW [65]	DG	undirected	$O(\|S\| \cdot \log^2(\|S\|))^{\ddagger}$	$O(\log^2(\|S\|))^{\dagger}$
IEH [54]	KNNG	directed	$O(\|S\|^2 \cdot \log(\|S\|) + \|S\|^2)^{\ddagger}$	$O(\|S\|^{0.52})^{\ddagger}$
FANNG [43]	RNG	directed	$O(\|S\|^2 \cdot \log(\|S\|))$	$O(\|S\|^{0.2})$
HNSW [67]	DG+RNG	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\log(\|S\|))$
EFANNA [36]	KNNG	directed	$O(\|S\|^{1.13})^{\ddagger}$	$O(\|S\|^{0.55})^{\ddagger}$
DPG [61]	KNNG+RNG	undirected	$O(\|S\|^{1.14} + \|S\|)^{\ddagger}$	$O(\|S\|^{0.28})^{\ddagger}$
NSG [38]	KNNG+RNG	directed	$O(\|S\|^{\frac{1+c}{c}} \cdot \log(\|S\|) + \|S\|^{1.14})^{\dagger}$	$O(\log(\|S\|))$
HCNNG [72]	MST	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\|S\|^{0.4})^{\ddagger}$
Vamana [88]	RNG	directed	$O(\|S\|^{1.16})^{\ddagger}$	$O(\|S\|^{0.75})^{\ddagger}$
NSSG [37]	KNNG+RNG	directed	$O(\|S\| + \|S\|^{1.14})$	$O(\log(\|S\|))$

† c, t are the constants. ‡ Complexity is not informed by the authors; we derive it based on the related papers' descriptions and experimental estimates. See Appendix D for deatils.



Distance: Euclidean

Dataset: fashion-mnist-784

https://ann-benchmarks.com/index.html#datasets

Effect of Beam Search

dim	steps	degree	Recall@1	Beam
2	200	20	0.998	1
4	15	60	0.999	1
8	5	300	0.998	1
16	4	2000	0.99	1
16	106	20	0.995	100

Graph Based ANN Recipe:

1. Delone graph approximation for local traversing

+

2. The way to move fast into the neighbourhood of query (long links, tree structure, random jumps)

- In a plane KNN-graph (no long range links), when $d \ll \log(n)$, the number of steps of greedy algorithm is $\Theta(n^{1/d})$

- When $d > \log(n)$ in a plane KNN-graph, the greedy algorithm converges in at most two steps

Theorem 1. *Assume that* $d \gg \log\log n$ *and we are given some constant* $c \geq 1$. *Let* M *be a constant such that* $M > \sqrt{\frac{4c^2}{3c^2-1}}$, *then, with probability* $1-o(1)$, $G(M)$*-based NNS solves* c, *R-ANN for any* R *(or the exact NN problem if* $c = 1$*); time complexity is* $\Theta\left(d^{1/2} \cdot n^{1/d} \cdot M^d\right) = n^{o(1)}$; *space complexity is* $\Theta\left(n \cdot d^{-1/2} \cdot M^d \cdot \log n\right) = n^{1+o(1)}$.

Theorem 2. *For any* $c > 1$ *let* $\alpha_c = \cos\left(\frac{\pi}{2c}\right)$ *and let* M *be any constant such that* $M < \frac{\alpha_c^2}{\alpha_c^2+1}$. *Then, with probability* $1-o(1)$, $G(M)$*-based NNS solves* c, *R-ANN (for any* R *and for spherical distance); time complexity of the procedure is* $\Theta\left(n^{1-M+o(1)}\right)$; *space complexity is* $\Theta\left(n^{2-M+o(1)}\right)$.

[Liudmila Prokhorenkova, Aleksandr Shekhovtsov "*Graph-based Nearest Neighbor Search: From Practice to Theory*"]

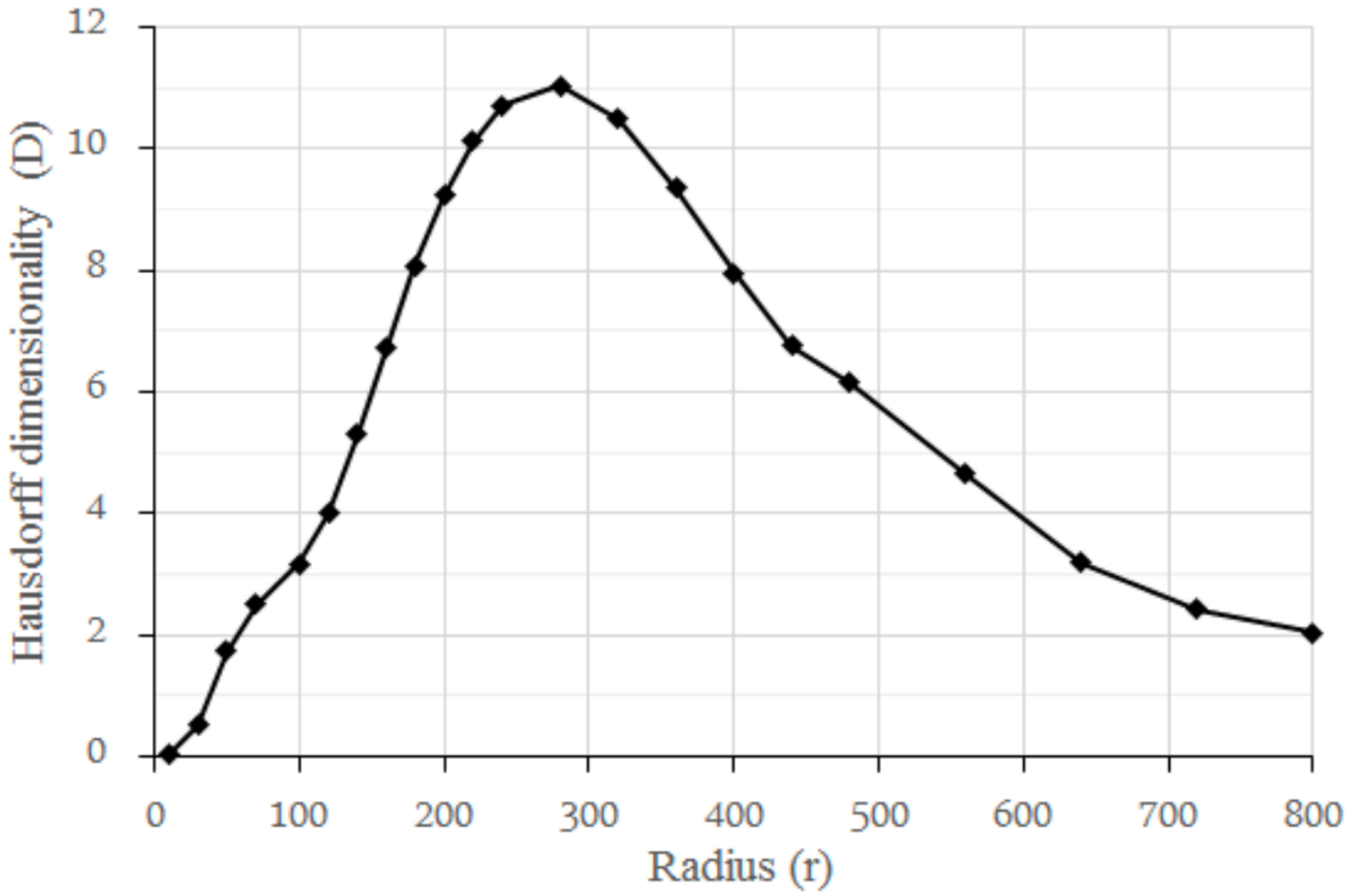
Intrisict in real world dataset often is not high

Hausdorff Dimensionality

$$D(r_1, r_2) = \frac{\log\left(\frac{n(r_1)}{n(r_2)}\right)}{\log\left(\frac{r_1}{r_2}\right)},$$

where

$$n(r) = |\{(a, b) \mid a, b \in X, \rho(a, b) < r\}|$$



Hausdorff dimensionality of SIFT data measured at varying distance scales. The maximum dimensionality is shown to be approximately 11

[Harwood, Ben, and Tom Drummond. "Fanng: Fast approximate nearest neighbour graphs." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016]

Optimal Network Structure

Decision variables

$$x_{ij} = \begin{cases} 1, & \text{if edge } (i, j) \text{ belongs to the solution} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$y_{ij}^{k} = \begin{cases} 1, & \text{if vertex } k \text{ belongs to the greedy walk from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Objective function

$$\min \frac{1}{n} \sum_{i=1}^{n} \sum_{q \in D} O(i, j_q) f_q \text{ (discrete domain)} \tag{3a}$$

$$\min \frac{1}{n} \sum_{i=1}^{n} \int_{D} O(i, j_q) f(q) dq, \text{ (continuous domain),} \tag{3b}$$

$$\text{where } j_q = \arg\min_{j=\overline{1,n}} d(j, q) \tag{4}$$

$$O(i, j_q) = \left| \left\{ l \in V : \exists k \, x_{lk} = 1 \text{and} \, y_{ij_q}^{k} = 1 \right\} \right| \tag{5}$$

Constraints

$$x_{ii} = 0 \quad \forall i \in V \tag{6}$$

$$y_{ij}^{i} = y_{ij_q}^{j} = 1 \quad \forall i, j_q \in V \tag{7}$$

$$\sum_{k=1}^{n} x_{lk} y_{ij_q}^{k} \geq y_{ij_q}^{l} \quad \forall i, j_q, l \in V \tag{8}$$

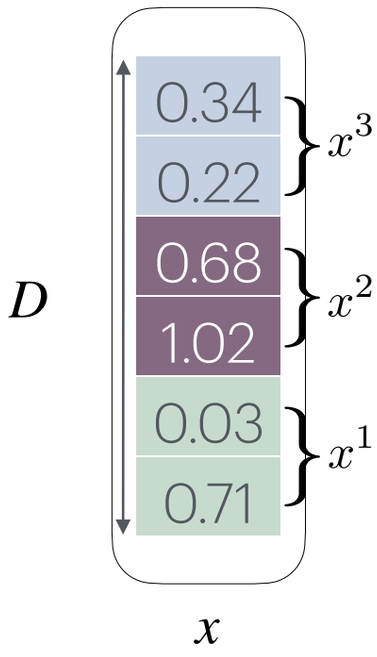
$$l^{*} = \arg\min_{l \in V : \, x_{kl} = 1} (d(l, q)) \Rightarrow y_{ij_q}^{l^{*}} \geq y_{ij_q}^{k} \quad \forall q \in D \quad \forall i, k \in V \tag{9}$$

Product Quantisation Codes (PQ-Codes)

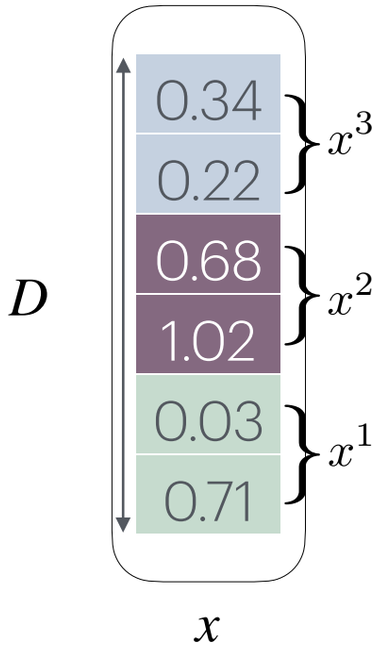
Let x is D-dimensional vector

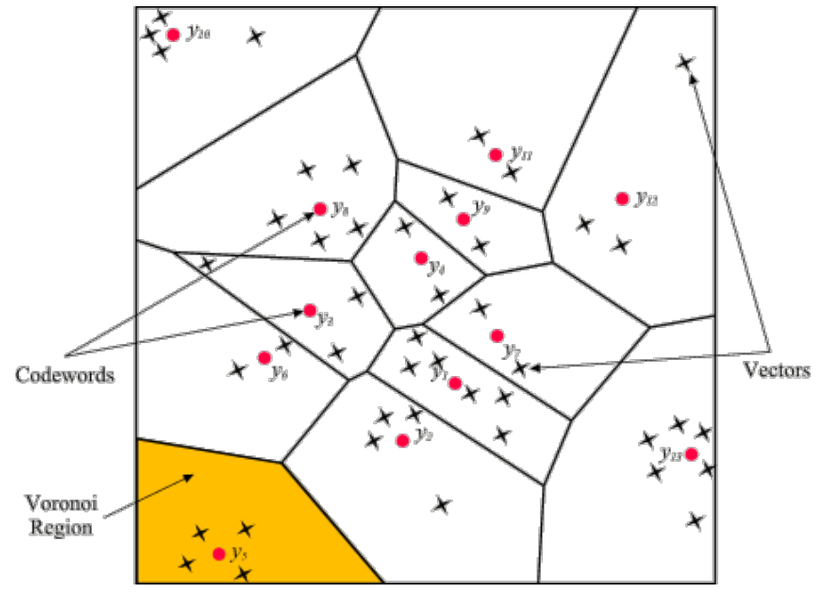
Split x to M=3 parts

Split x to M=3 parts

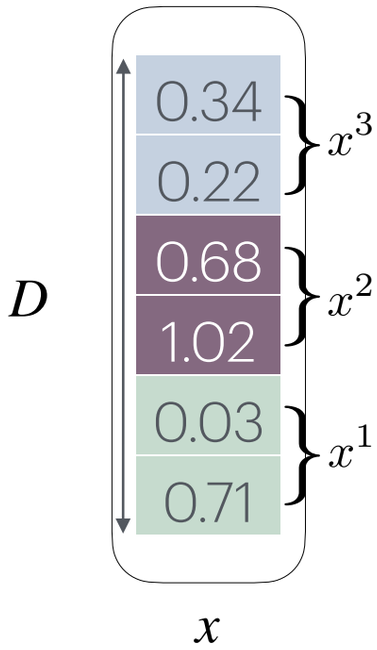


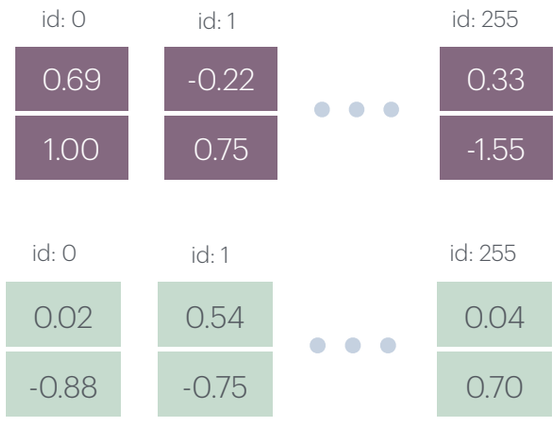
For green component perform k-means to find K=256 centroids

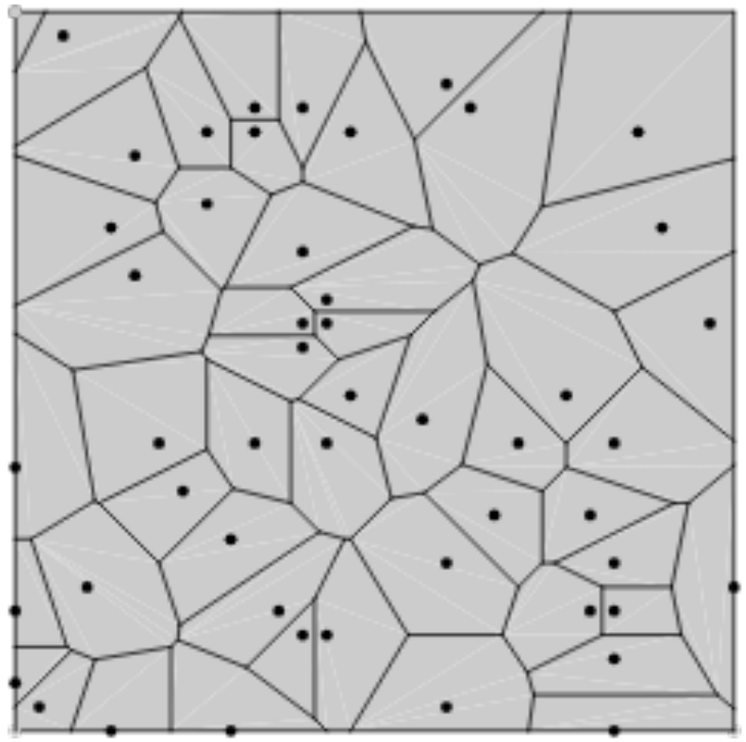




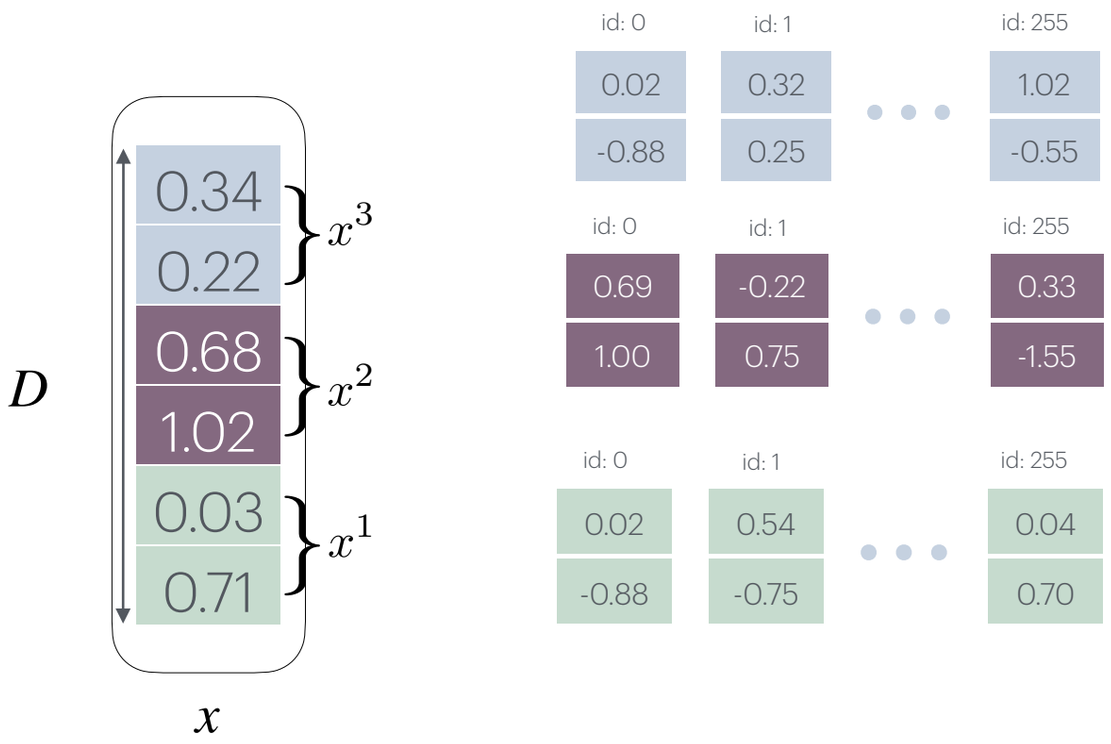
Perform k-means to find K=256 centroids for violet component



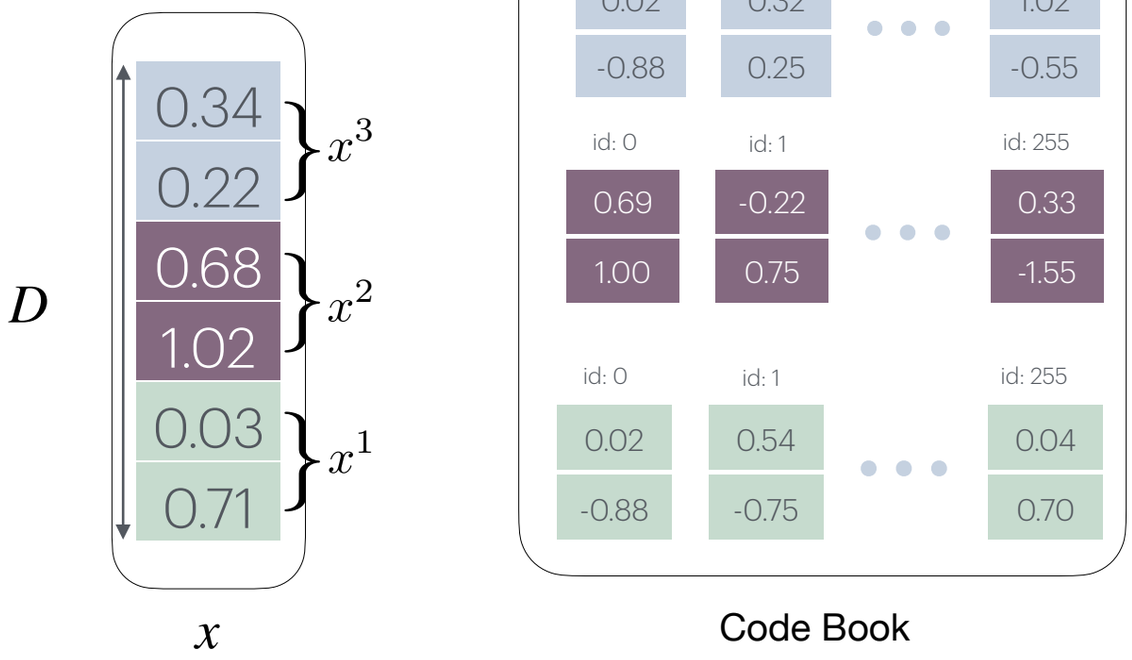




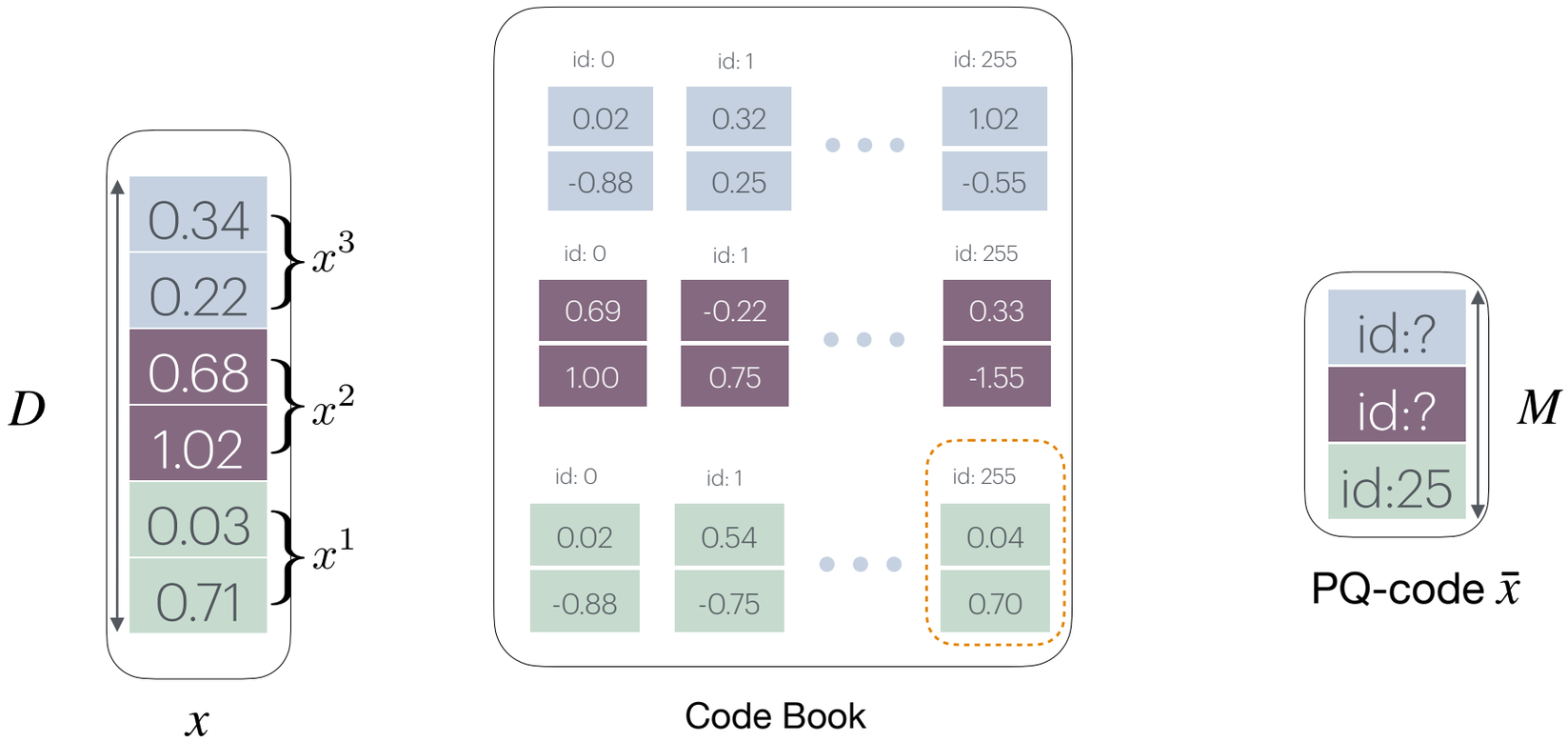
For each component perform k-means to find K=256 centroids



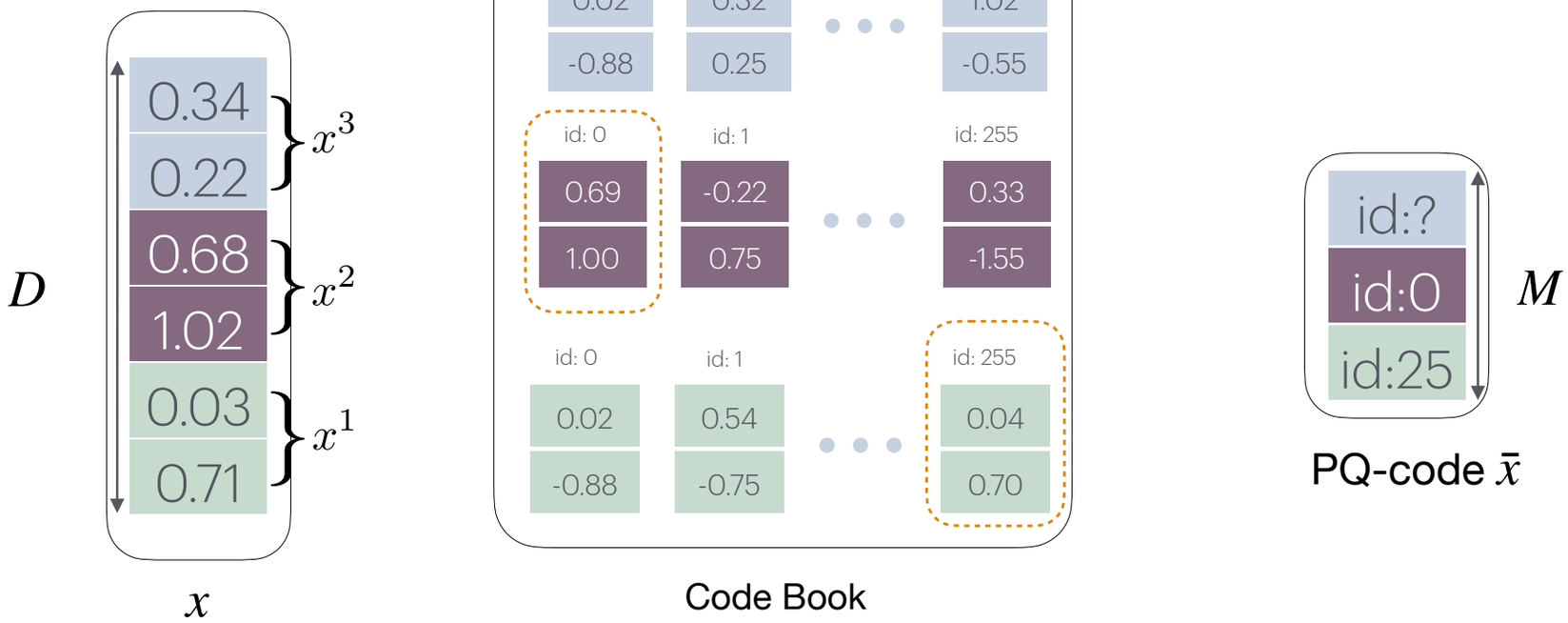
It is our code book



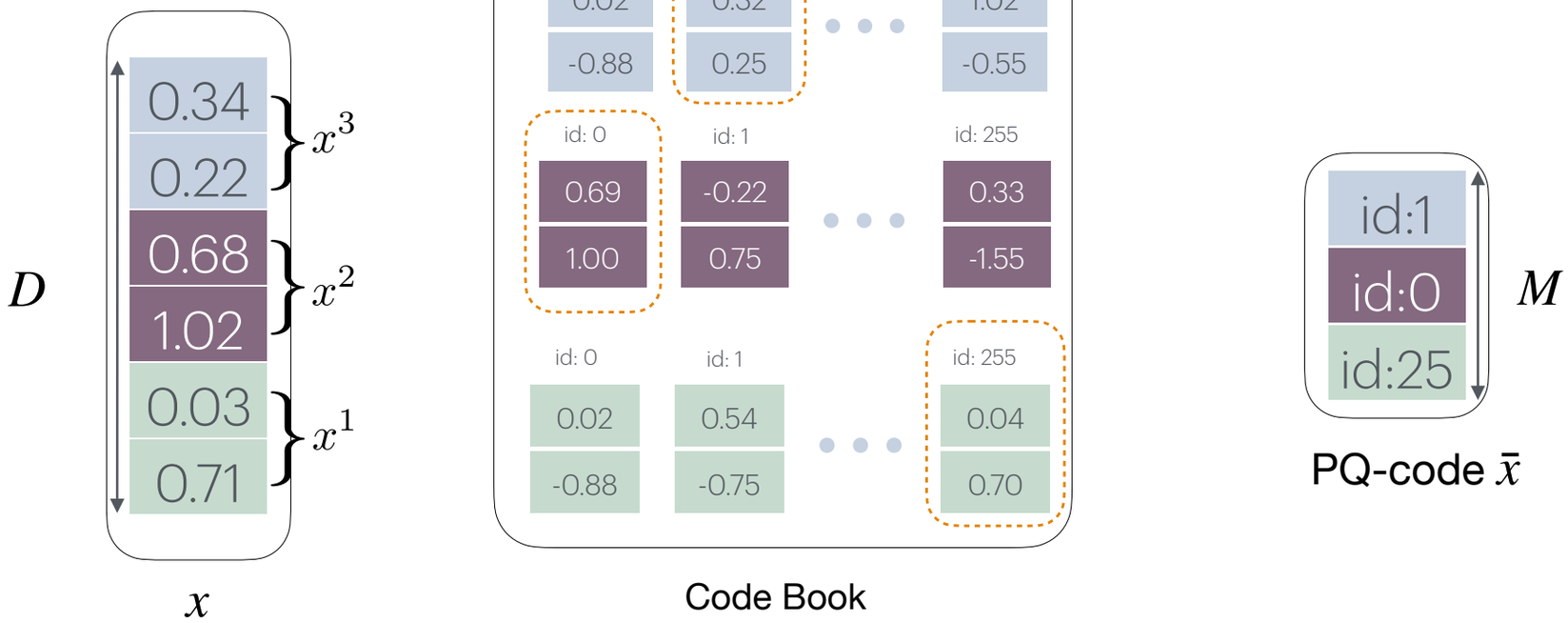
Assign id of the closest centroid for green component

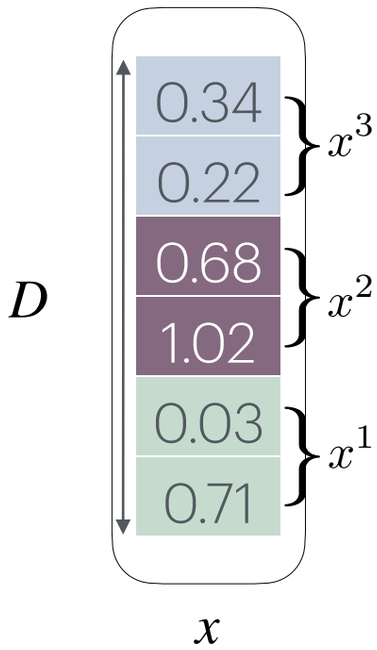


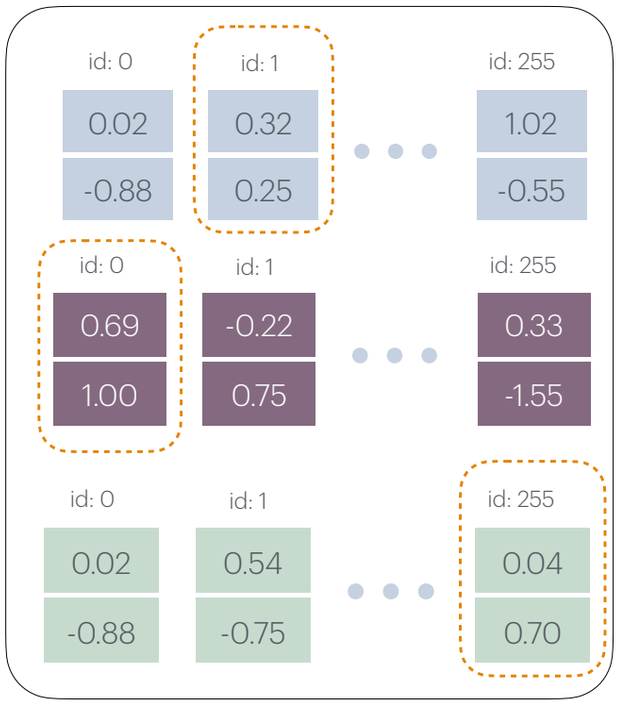
Assign id of the closest centroid for violet component



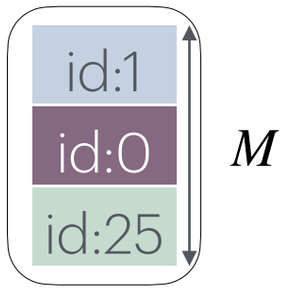
Assign id of the closest centroid for grey component



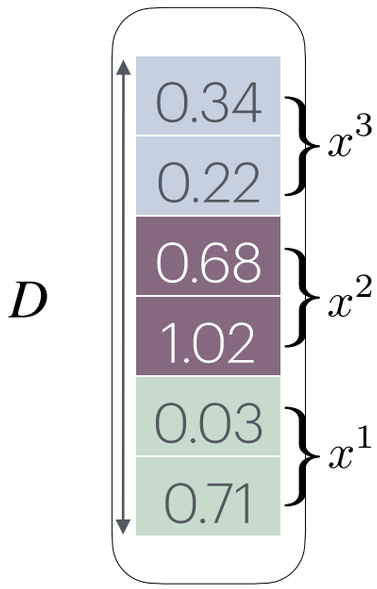




Code Book



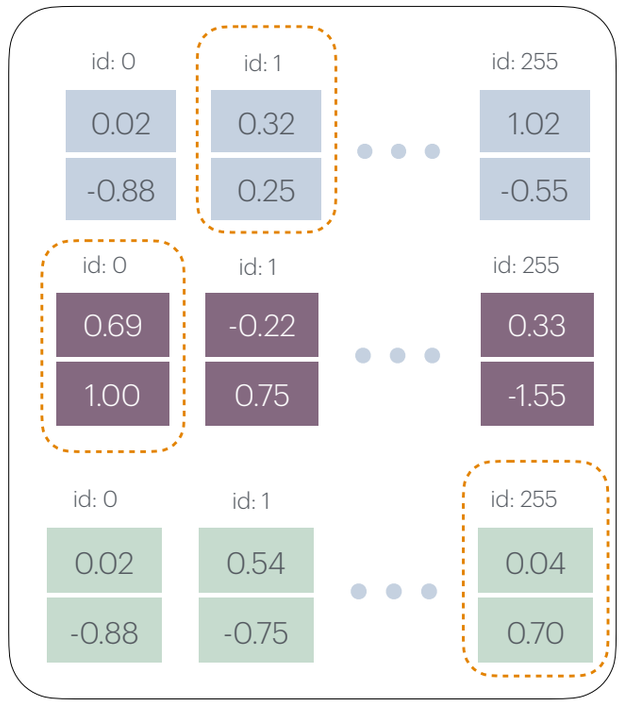
PQ-code $\bar{x}$



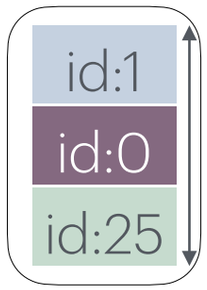
D

x

float: 32 bit



Code Book



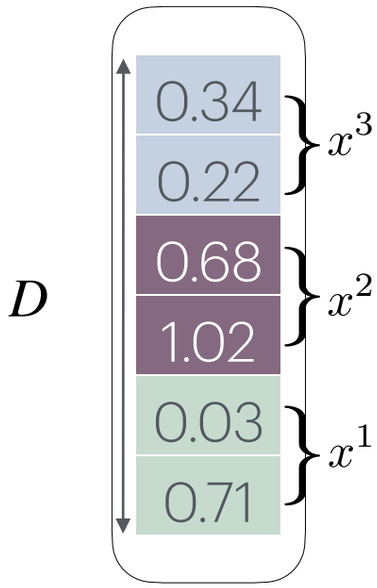
M

PQ-code $\bar{x}$

e.g., D = 128

128 × 32 = 4096 [bit]

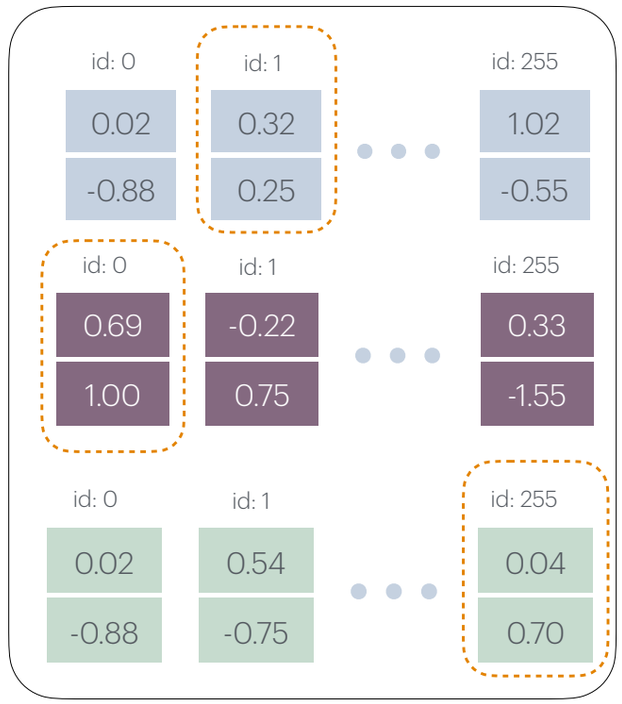
[Jegou, Herve, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search." *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010): 117-128.]



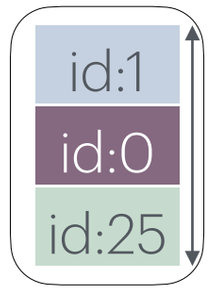
D

x

float: 32 bit



Code Book



M

PQ-code

denoting: $\bar{x}$ or $q(x)$

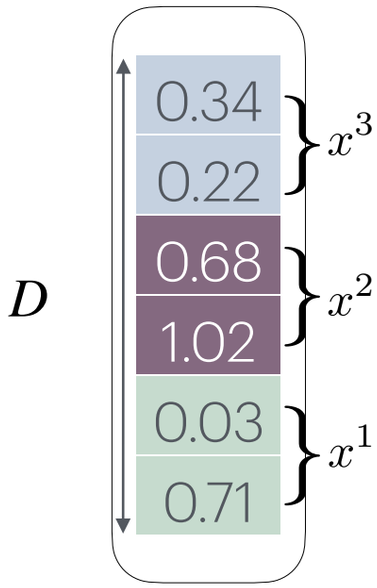
e.g., D = 128

128 × 32 = 4096 [bit]

e.g., M=8

8 × 8 = 64 [bit]

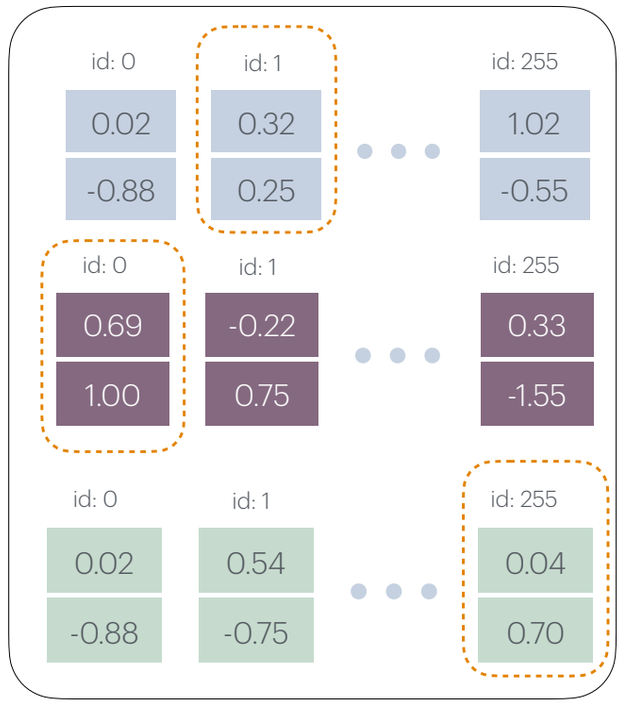
[Jegou, Herve, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search." *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010): 117-128.]



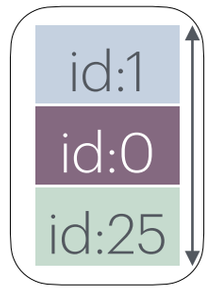
D

x

float: 32 bit



Code Book



M

PQ-code

denoting: $\bar{x}$ or $q(x)$

e.g., D = 128

$128 \times 32 = 4096$ [bit]

64 times less!

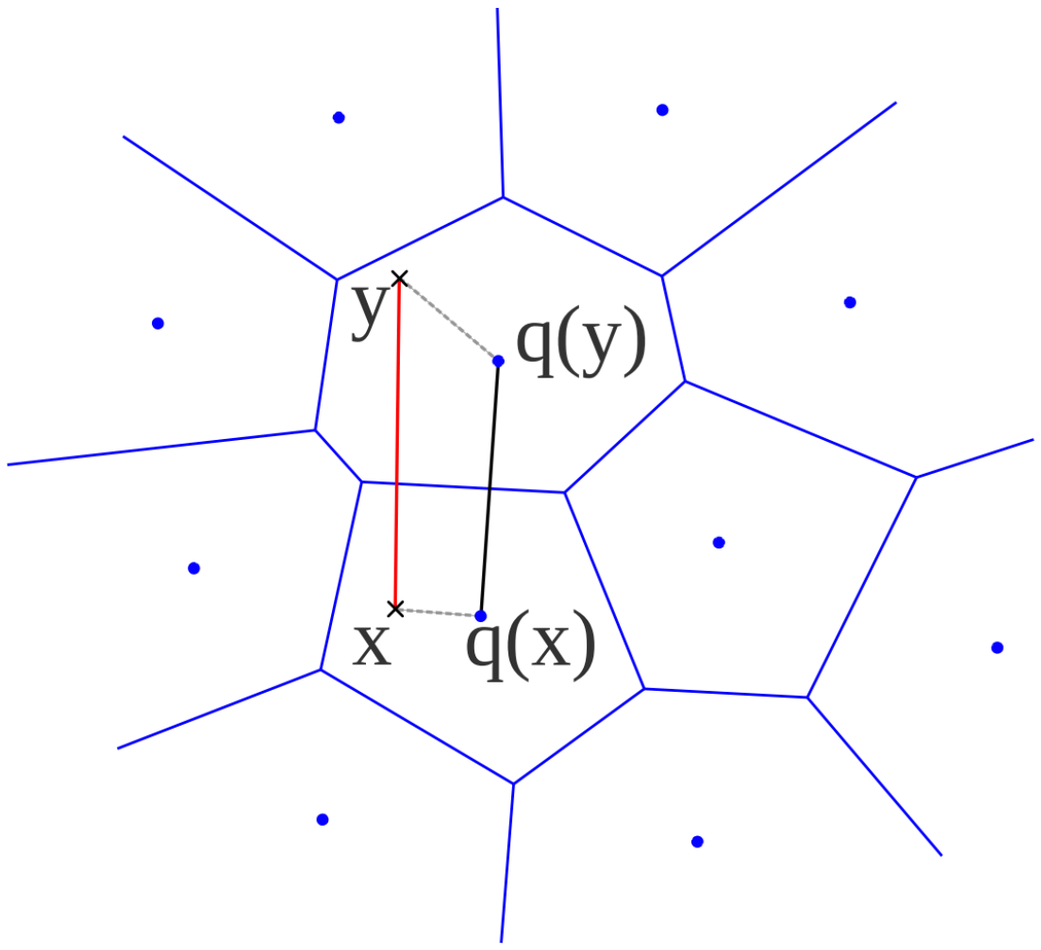
e.g., M=8

$8 \times 8 = 64$ [bit]

[Jegou, Herve, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search." *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010): 117-128.]

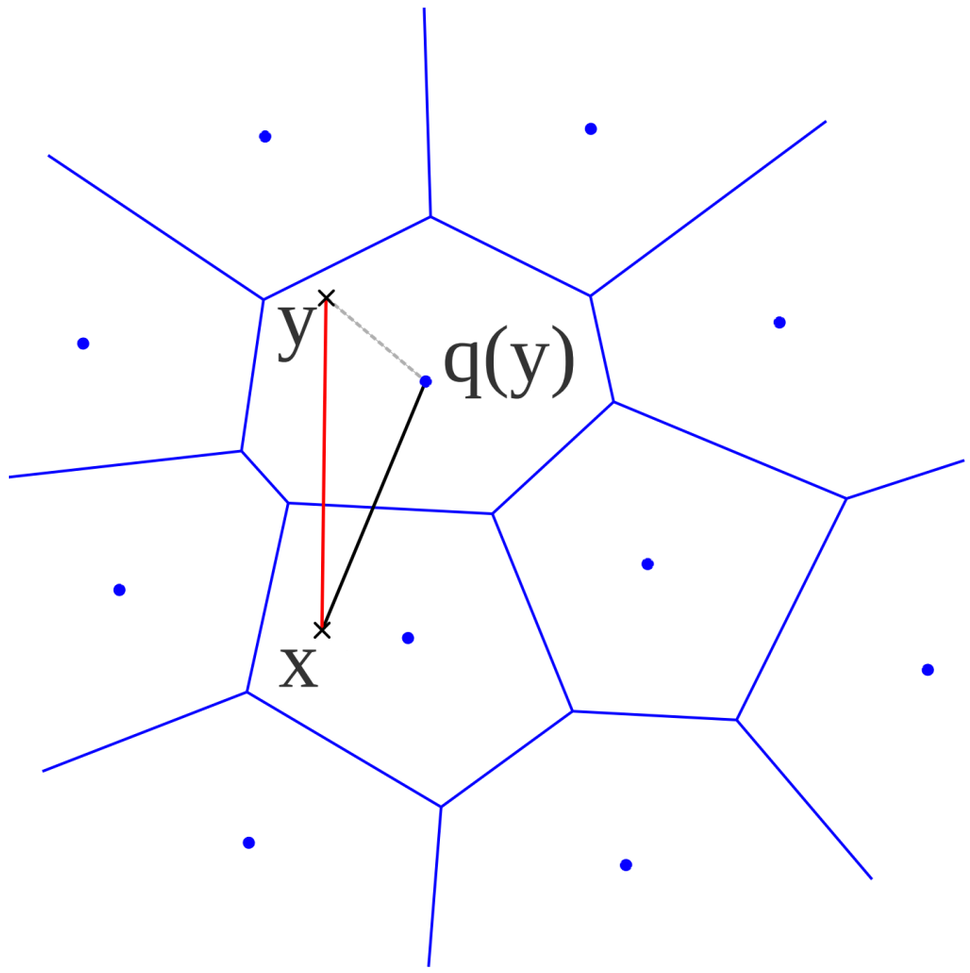
Asymmetric Distance Search

Symmetric case

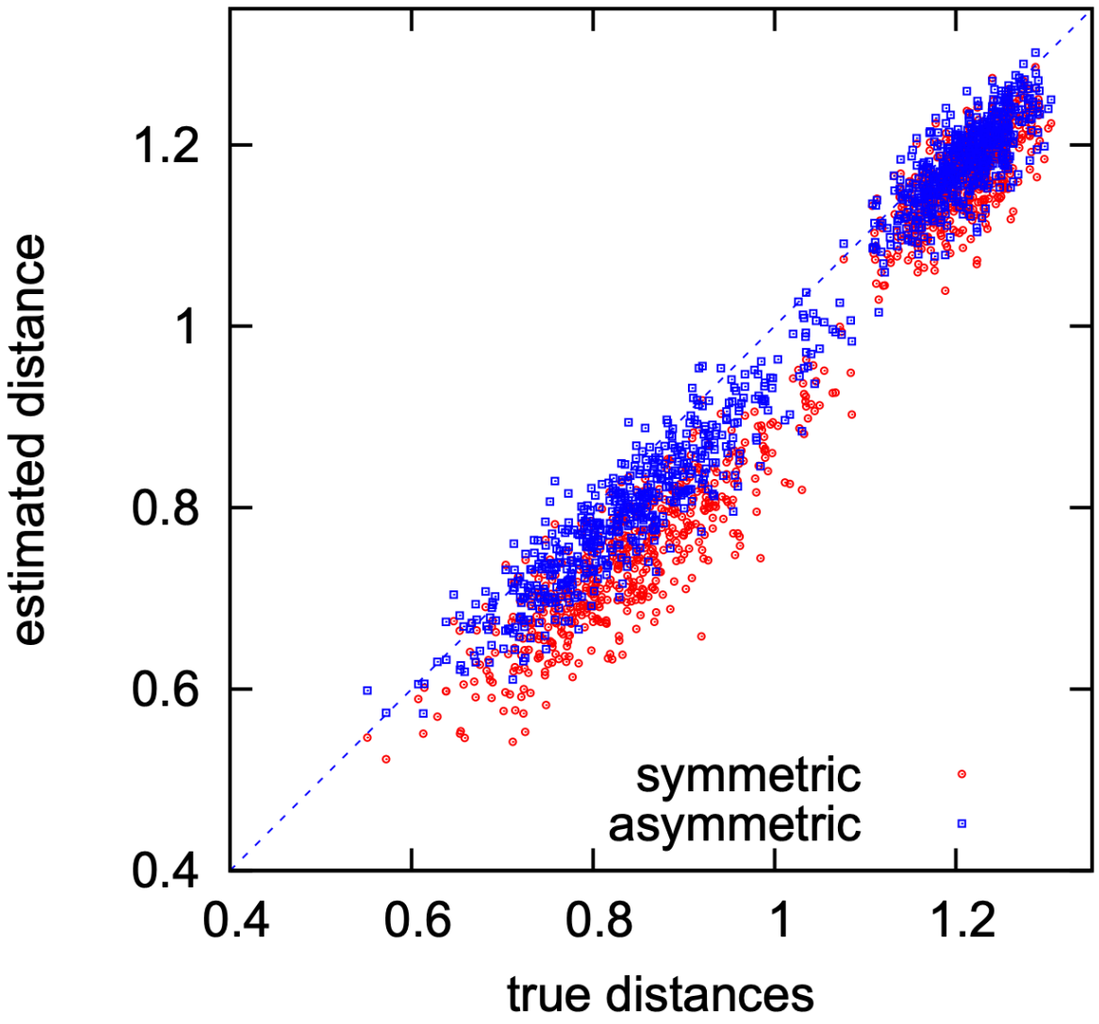


$$d_{SD}(x,y)^2 = \sum_{m=1}^{M} d(q(x^m), q(y^m))^2$$

Asymmetric case



$$d_{AD}(\boldsymbol{x}, \boldsymbol{y})^2 = \sum_{m=1}^{M} d\left(\boldsymbol{x}^m, \boldsymbol{q}^m\left(\boldsymbol{y}^m\right)\right)^2$$



Typical query of a SIFT vector (D=128) in a set of 1000 vectors: comparison of the distance $d(x, y)$ obtained with the SD and AD estimators. M = 8 and K = 256, i.e., 64-bit code vector

[https://inria.hal.science/inria-00514462/file/jegou_pq_postprint.pdf]