Distance and Path Oracles in Spatial Databases

Jagan Sankaranarayanan*, Hanan Samet

jagan,hjs@cs.umd.edu

Department of Computer Science

Institute for Advanced Computer Studies

University of Maryland

College Park, MD 20742, USA

*Now at Google

Overview

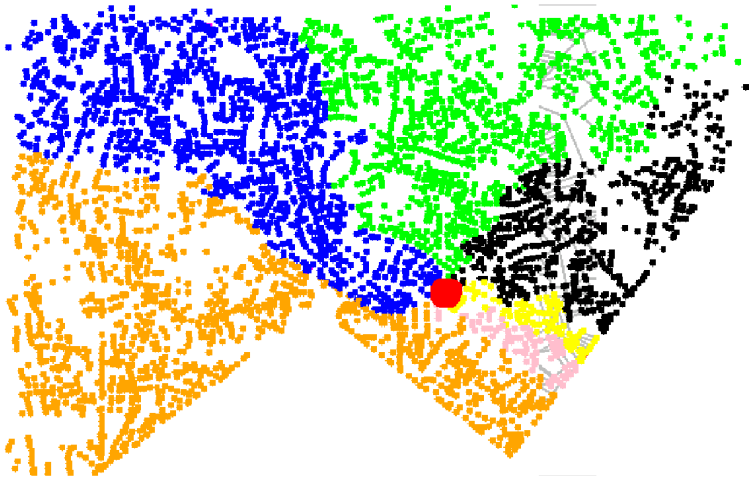
1. Motivation: Want to find the shortest path and network distance between points A and B without having to store the n^2 shortest paths and network distances between all pairs
   - Willing to expend a little bit of time like $O(\log n)$
   - Willing to accept ϵ error in accuracy of distance
2. Goal: Efficient shortest path and distance computation in a spatial network
   - Avoid applying Dijkstra's algorithm for each query which visits almost all of the vertices
3. Strategy: Determine groups of source and destination vertices that share common vertices in their shortest paths and precompute it
4. Result:
   - Can determine an intermediate vertex between any pair of vertices in a spatial network in $O(\log n)$ time and $O(n)$ storage
   - Can determine ϵ-approximate distance between any pair of vertices in spatial network in $O(\log n)$ time and $O(n/\epsilon^d)$ storage
   - Enable expressing operations using relational operators

Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths

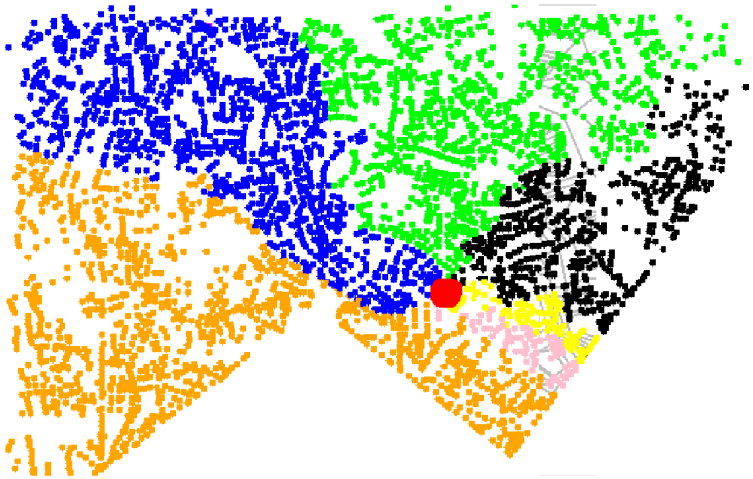
Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths
  - single source vertex to multiple destination vertices



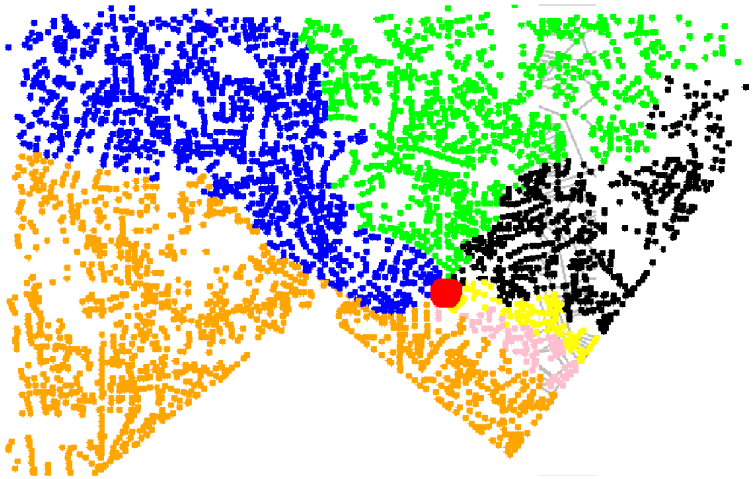
Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths
  - single source vertex to multiple destination vertices
  - Not captured: multiple source vertices to multiple destination vertices



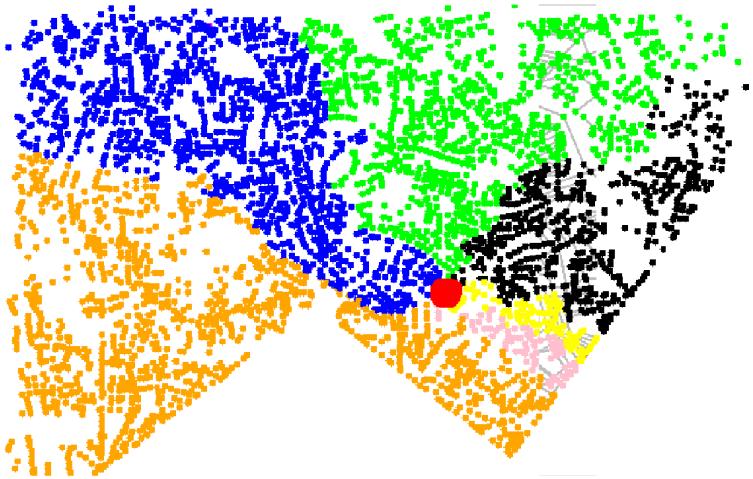
Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths
  - single source vertex to multiple destination vertices
  - Not captured: multiple source vertices to multiple destination vertices
- A new framework: Path Coherent Pairs (PCP)



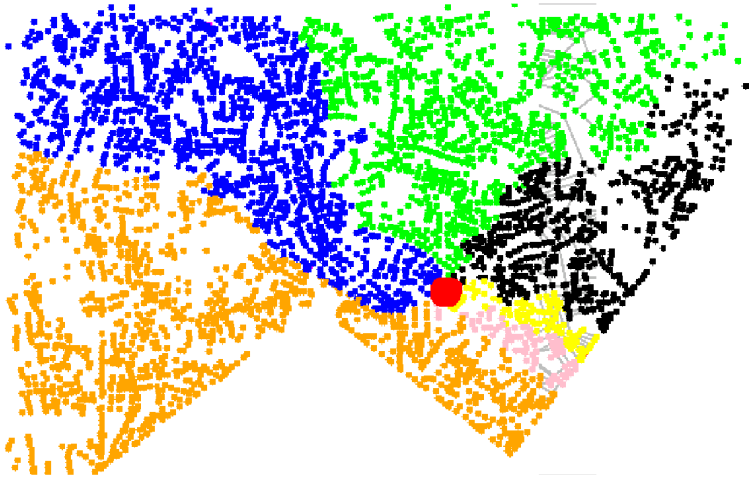
Path Coherence Beyond SILC

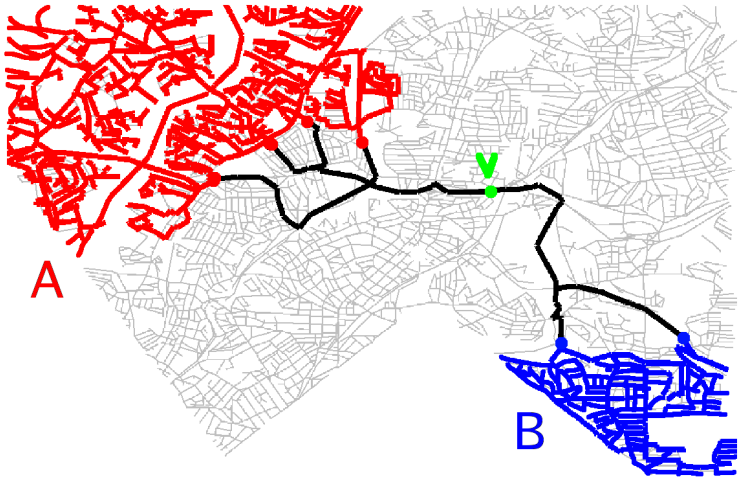
- The SILC framework captures the path coherence in the shortest paths
  - single source vertex to multiple destination vertices
  - Not captured: multiple source vertices to multiple destination vertices
- A new framework: Path Coherent Pairs (PCP)
  - A PCP is denoted by: (A,B,v)
    - A is a set of source vertices
    - B is a set of destination vertices
    - v is a common vertex to all pairs of shortest paths



Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths
  - single source vertex to multiple destination vertices
  - Not captured: multiple source vertices to multiple destination vertices
- A new framework: Path Coherent Pairs (PCP)
  - A PCP is denoted by: (A,B,v)
    - A is a set of source vertices
    - B is a set of destination vertices
    - v is a common vertex to all pairs of shortest paths
  - Example of a path coherent pair





Finding Path Coherent Pairs in Spatial Networks

Finding Path Coherent Pairs in Spatial Networks

- Source Vertices:

Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D)

Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N)

Finding Path Coherent Pairs in Spatial Networks

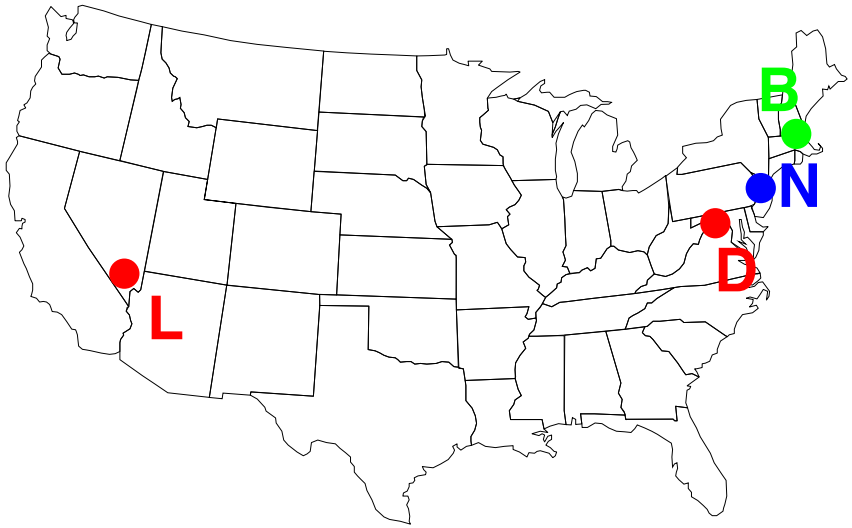
- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)

Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices:

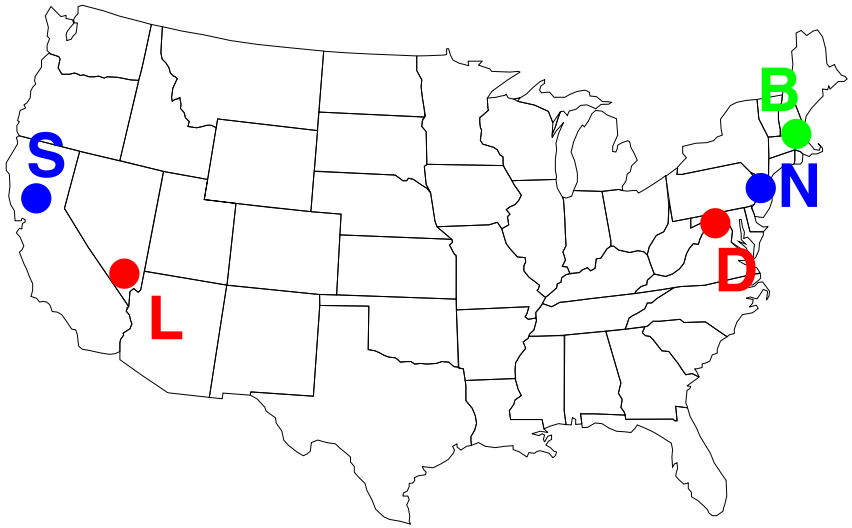
Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L)



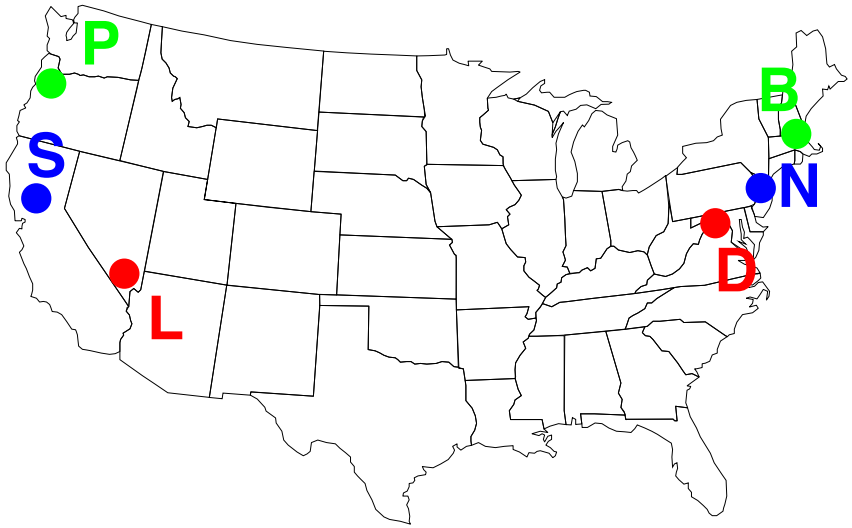
Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S)



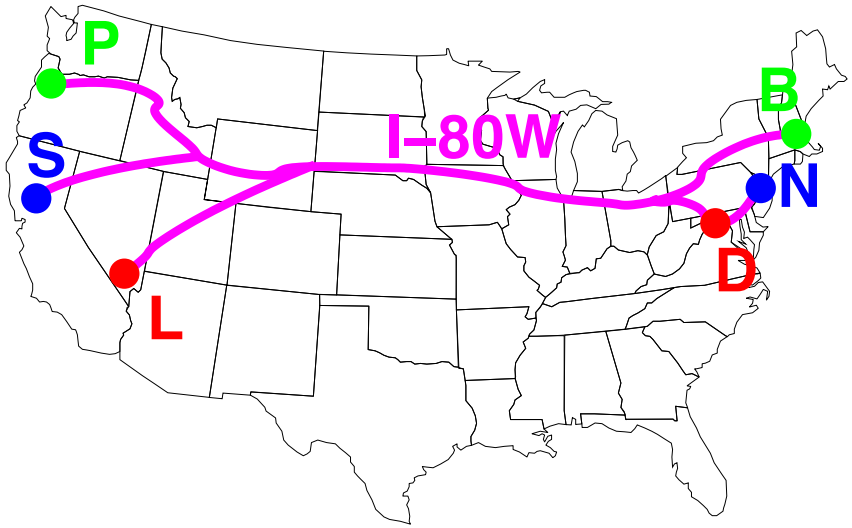
Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)



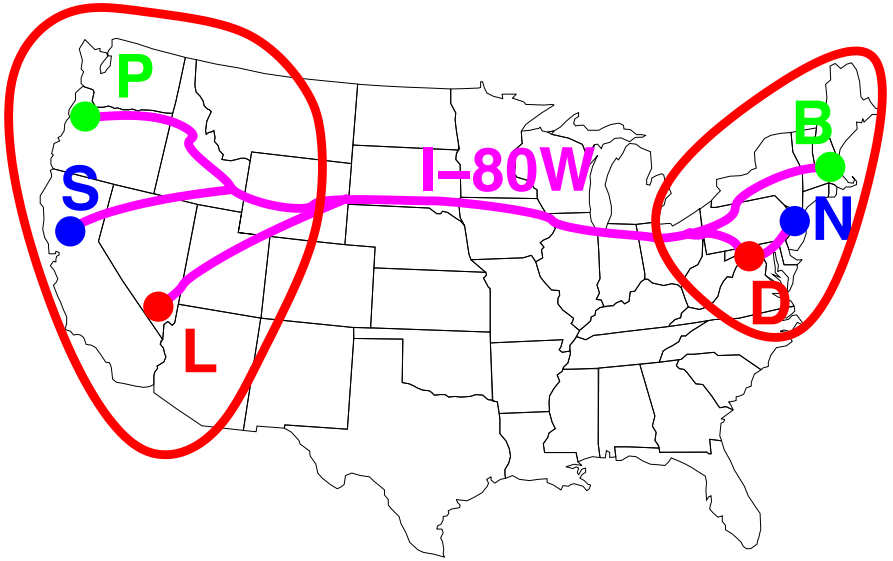
Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W



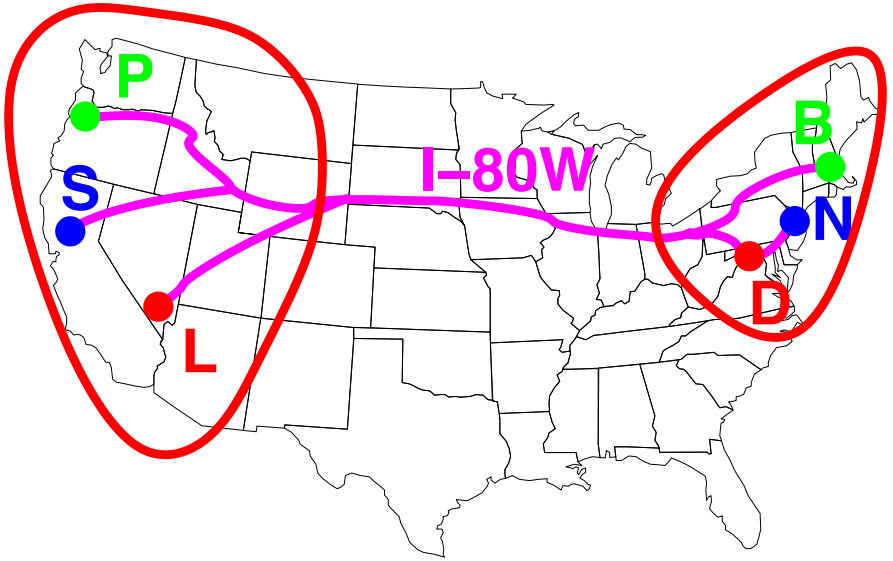
Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W
- Capture shortest paths from one million (say) sources in “North-East” to one million (say) destinations in “North-West” using $O(1)$ storage



Finding Path Coherent Pairs in Spatial Networks

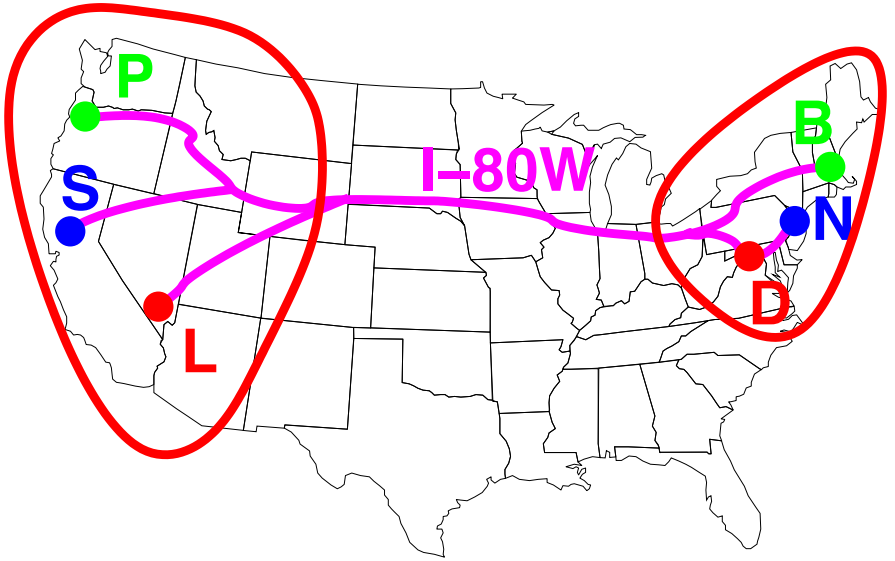
- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W
- Capture shortest paths from one million (say) sources in “North-East” to one million (say) destinations in “North-West” using $O(1)$ storage
- Intuition: Sources “sufficiently far” from destinations share common vertices in their shortest paths



Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W
- Capture shortest paths from one million (say) sources in “North-East” to one million (say) destinations in “North-West” using $O(1)$ storage
- Intuition: Sources “sufficiently far” from destinations share common vertices in their shortest paths

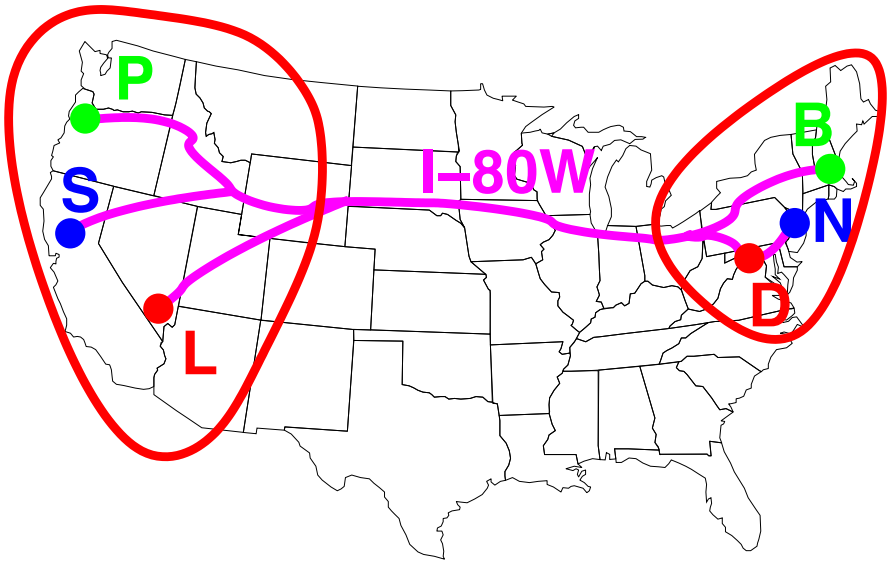
- Decompose road network into PCPs:
  - Any vertex pair is contained in exactly one PCP
  - All n^2 shortest paths are captured



Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W
- Capture shortest paths from one million (say) sources in “North-East” to one million (say) destinations in “North-West” using $O(1)$ storage
- Intuition: Sources “sufficiently far” from destinations share common vertices in their shortest paths

- Decompose road network into PCPs:
  - Any vertex pair is contained in exactly one PCP
  - All n^2 shortest paths are captured



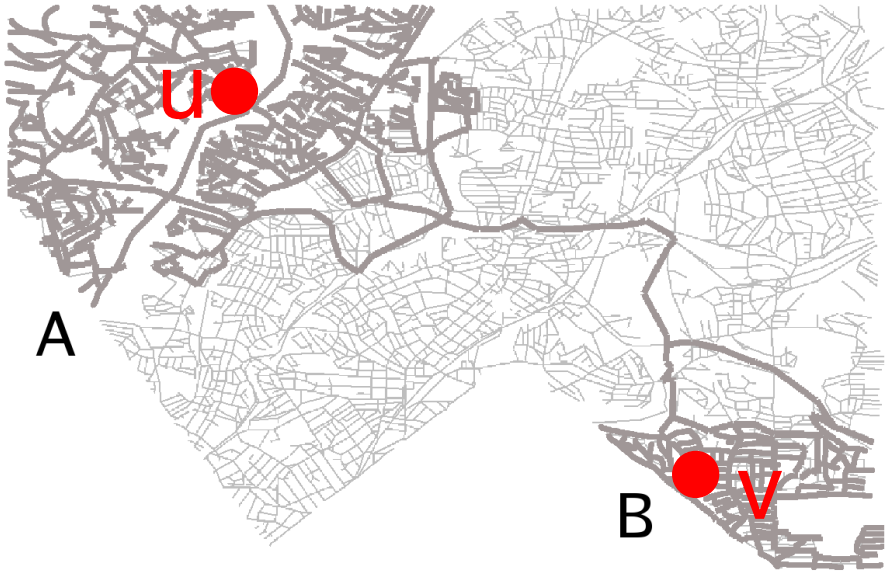
- Key idea is the analogy to the well-separated pairs in computational geometry

Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between

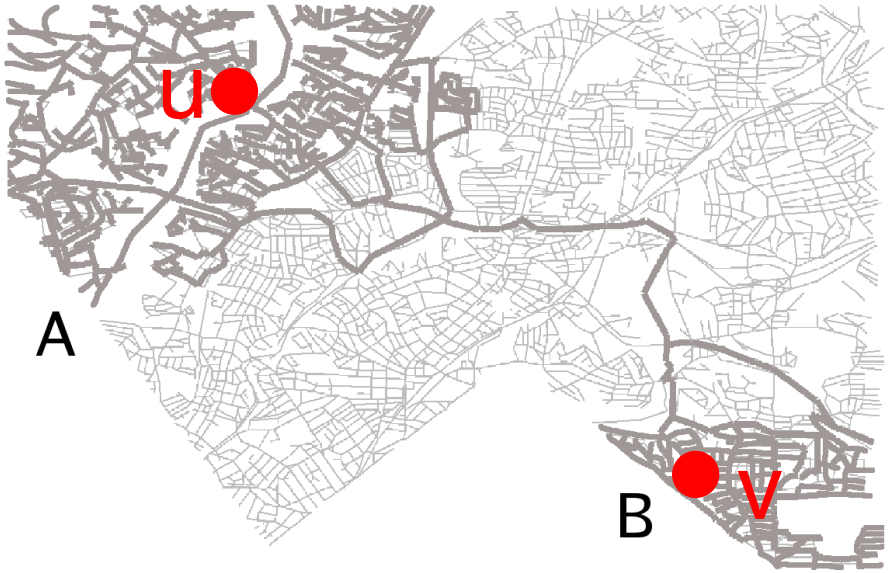
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v



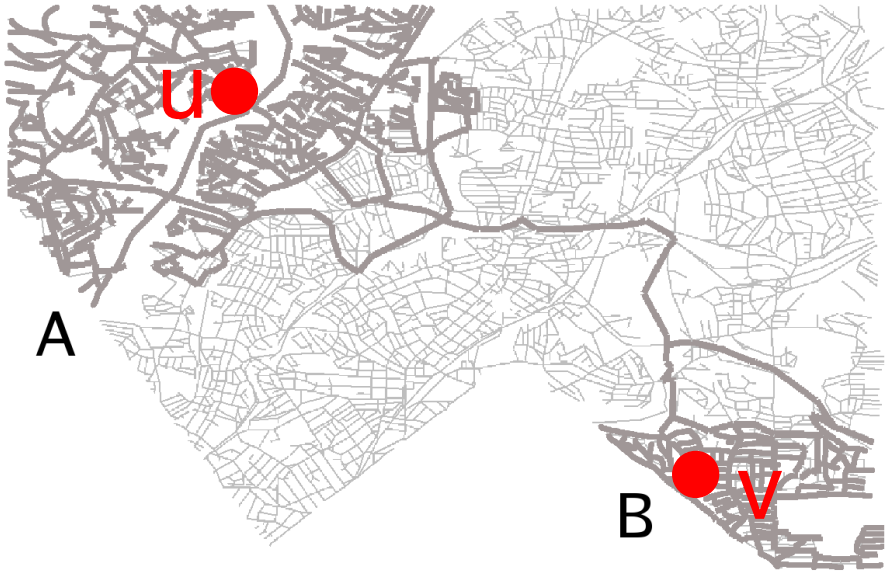
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?



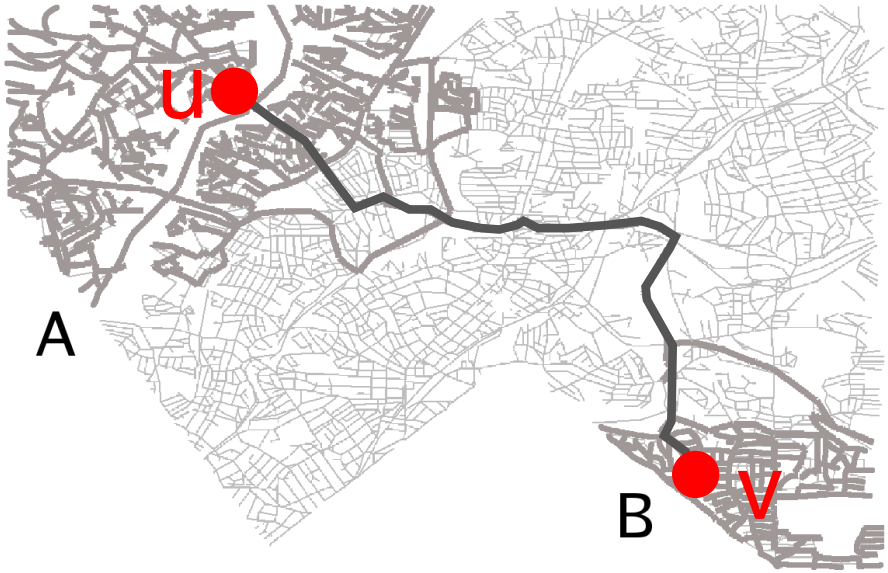
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths



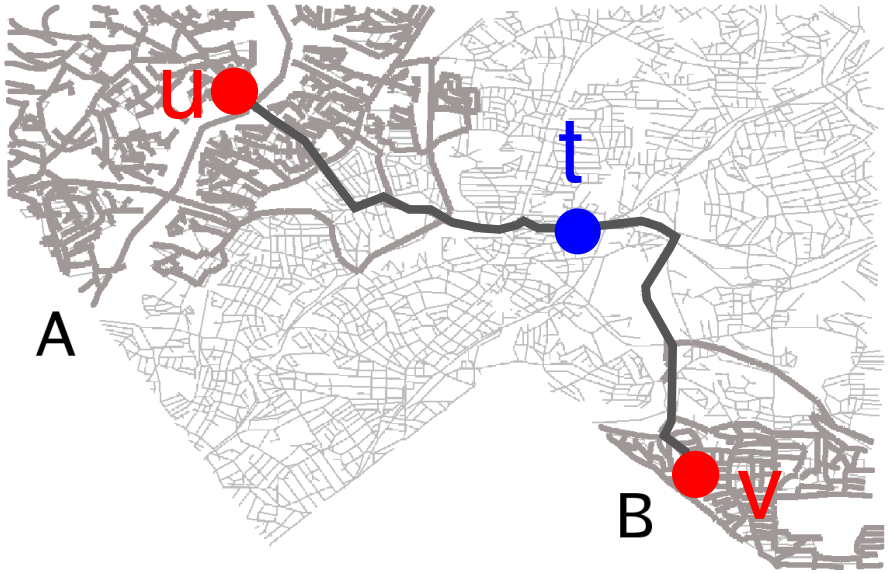
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured



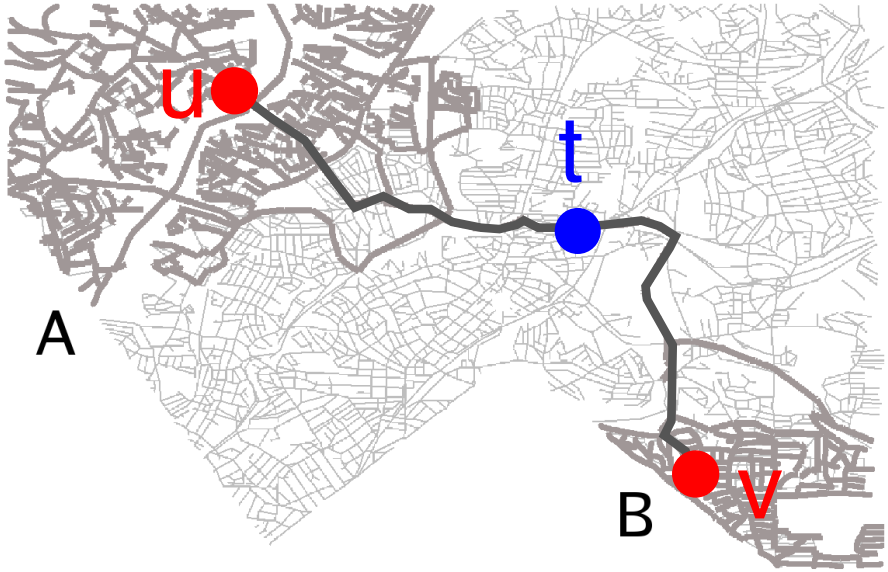
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B



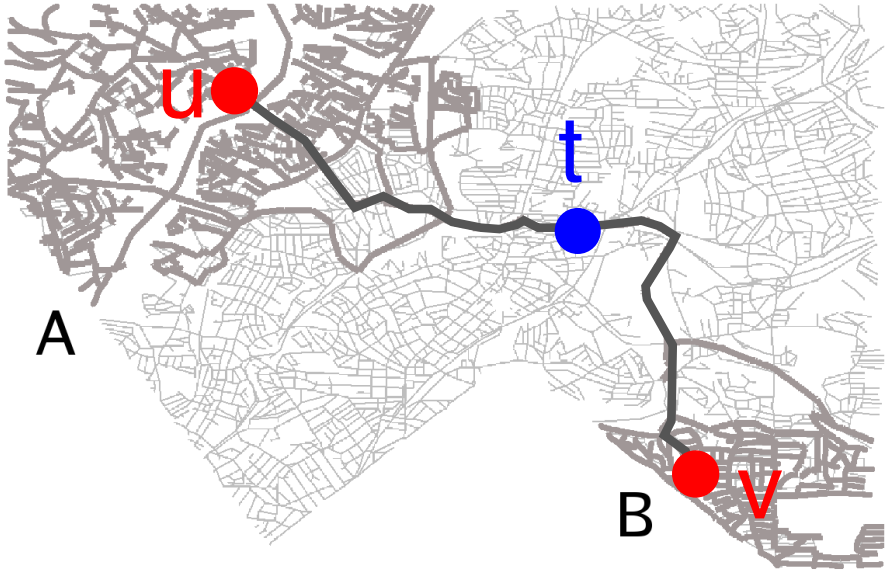
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B
- A path oracle is a PCP decomposition of a spatial network



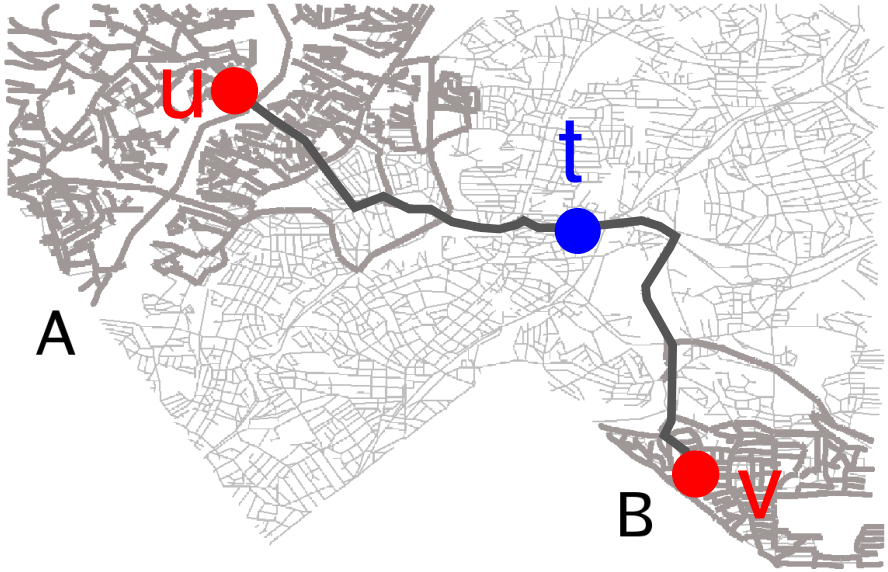
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B
- A path oracle is a PCP decomposition of a spatial network
  - Size of PCP decomposition should be linear in n



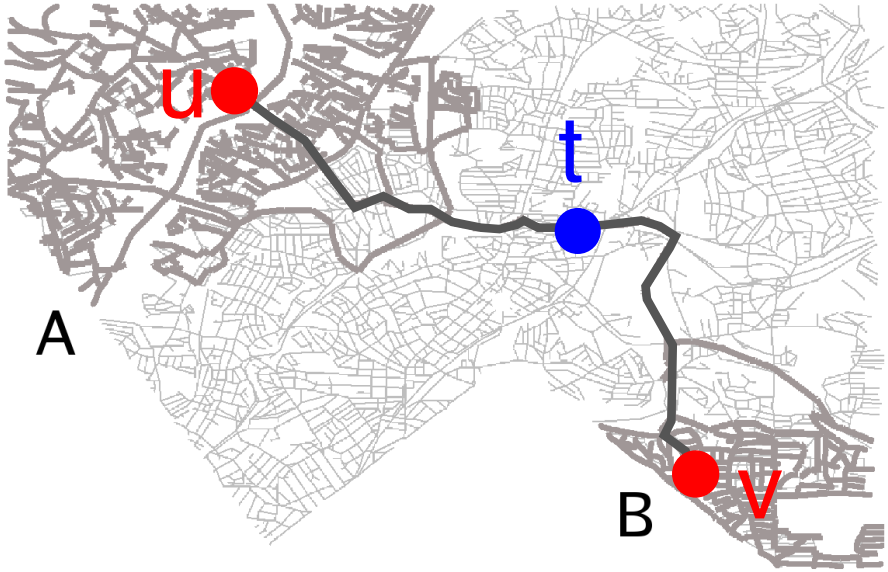
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B
- A path oracle is a PCP decomposition of a spatial network
  - Size of PCP decomposition should be linear in n
- Propose two path oracles



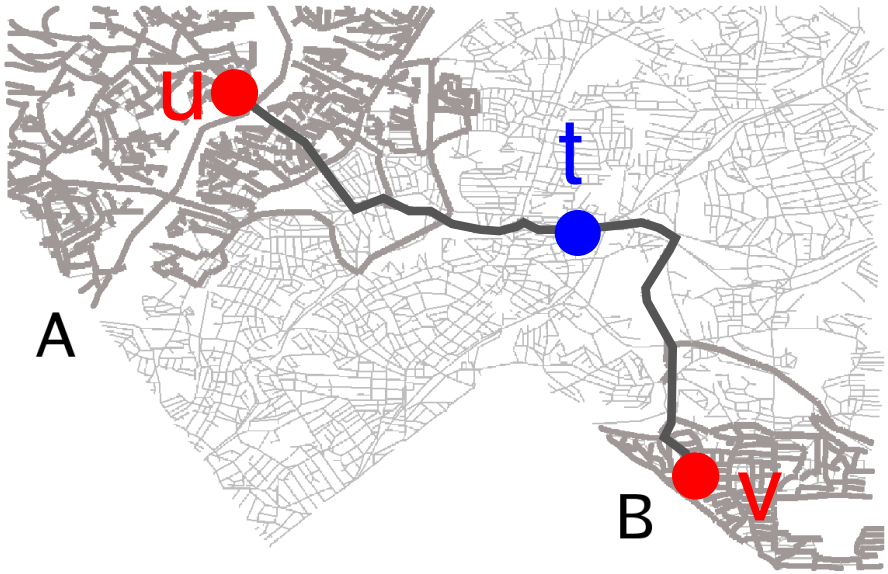
Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B
- A path oracle is a PCP decomposition of a spatial network
  - Size of PCP decomposition should be linear in n
- Propose two path oracles
  - Size $O(s^2 n)$; empirically $s \approx 12$
    - Using a B-tree, access in $O(\log n)$ time

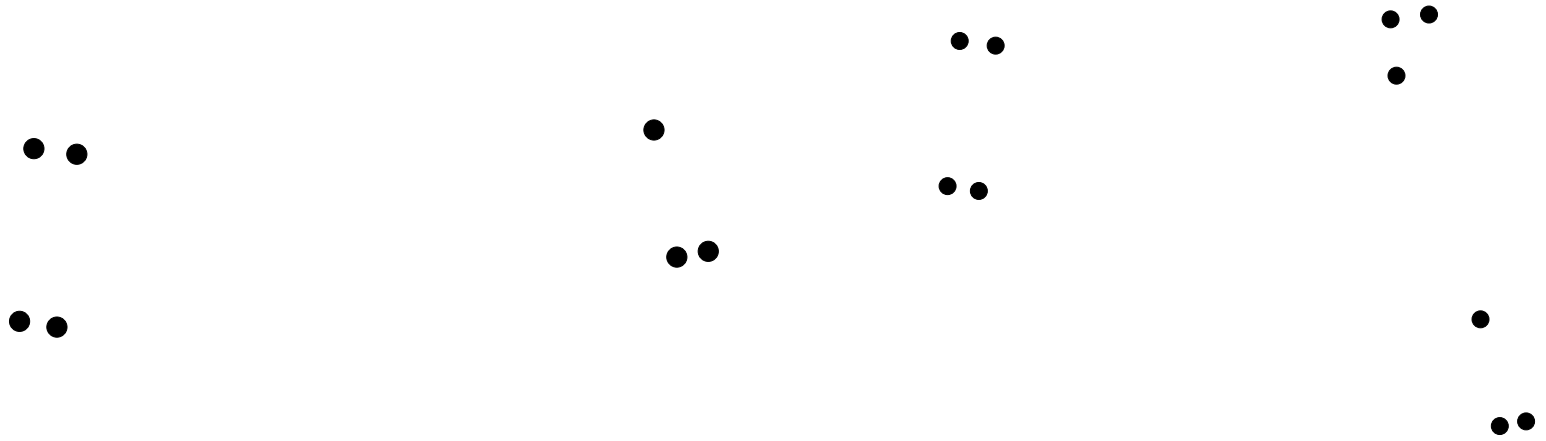


Definition: Path Oracles [VLDB 2009]

- Provide near instantly an intermediate vertex between a source u and a destination v
- How to take advantage of *path coherence*?
  - Intuition: Path coherent vertices have similar shortest paths
  - Part of the shortest path between between u and v can be captured by recording the common vertex t between A and B
- A path oracle is a PCP decomposition of a spatial network
  - Size of PCP decomposition should be linear in n
- Propose two path oracles
  - Size $O(s^2 n)$; empirically $s \approx 12$
    - Using a B-tree, access in $O(\log n)$ time
  - Size $O(s^2 n \log n)$
    - Using a hash table, access in $O(1)$ time

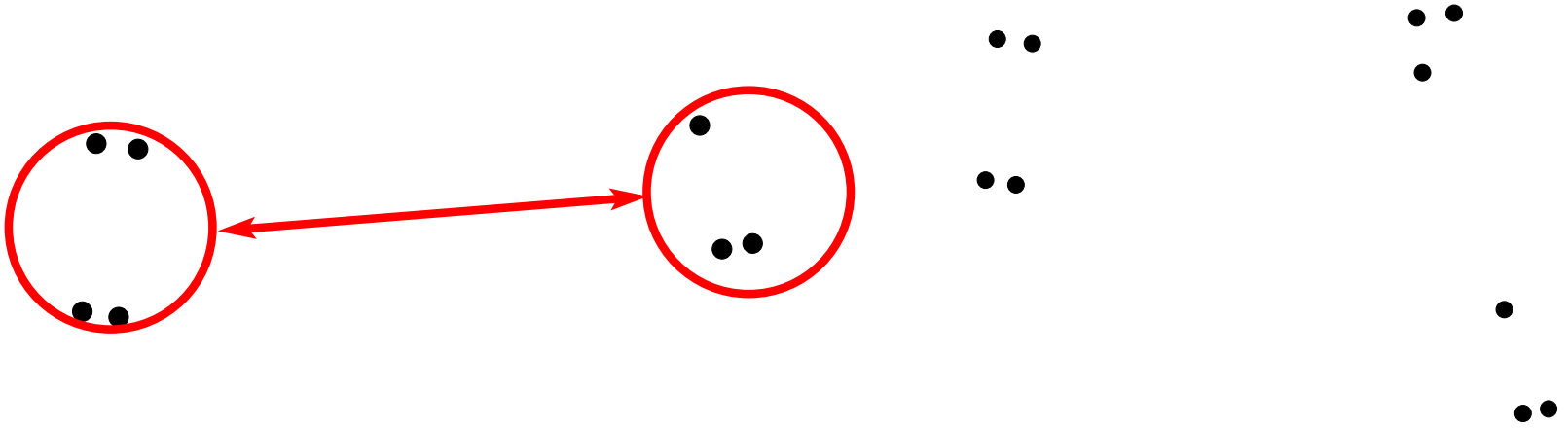


Definition of Well-Separated Pairs (WSPs)



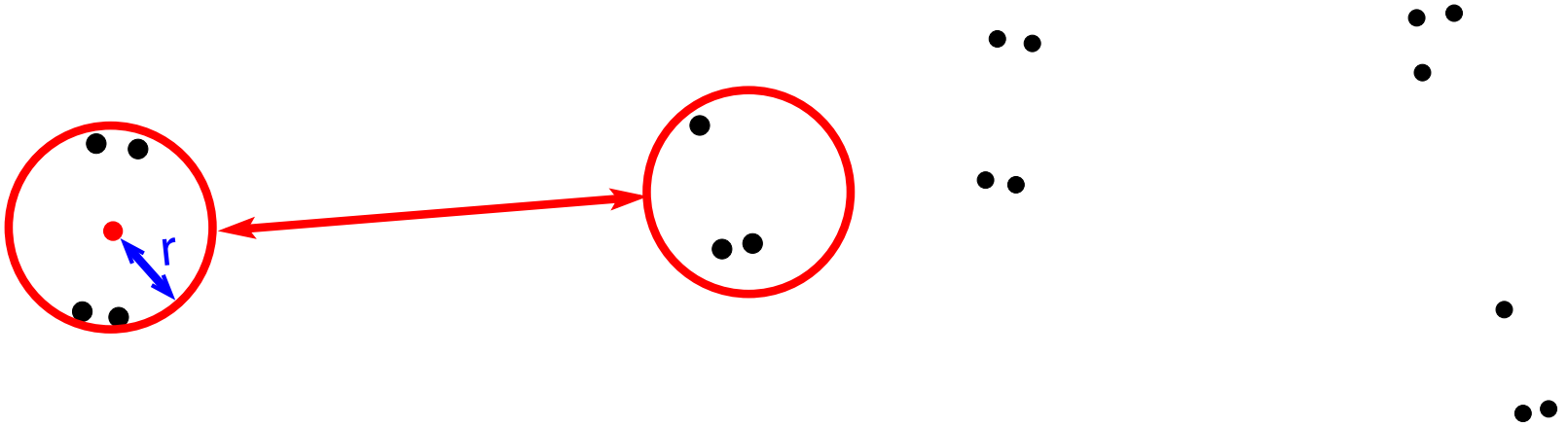
- Let $s > 0$ be the separation factor

Definition of Well-Separated Pairs (WSPs)



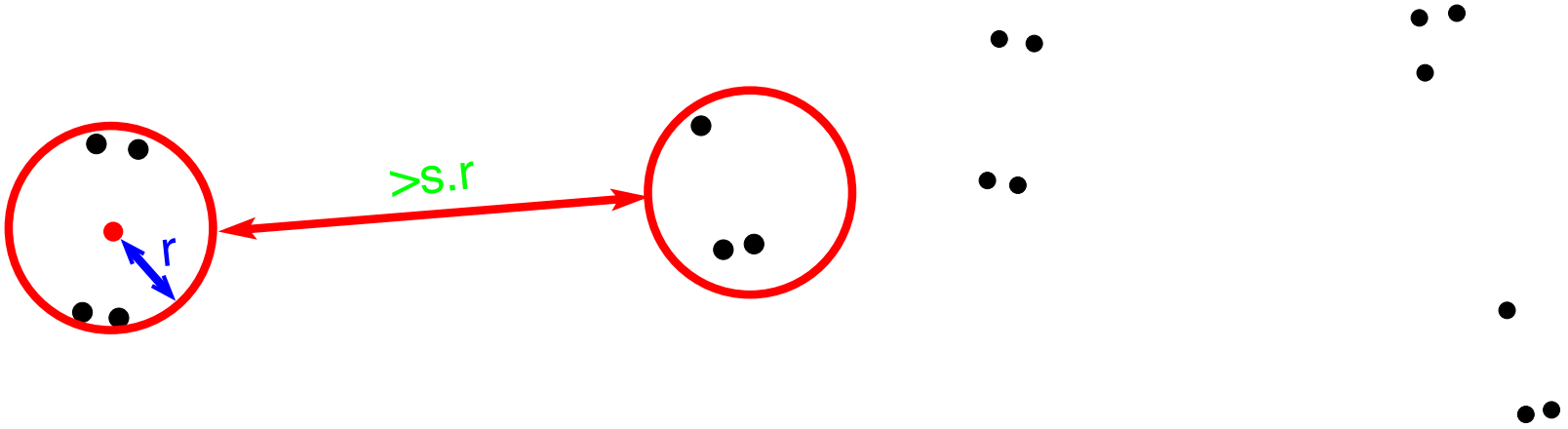
- Let $s > 0$ be the separation factor
- Two sets of points A and B

Definition of Well-Separated Pairs (WSPs)



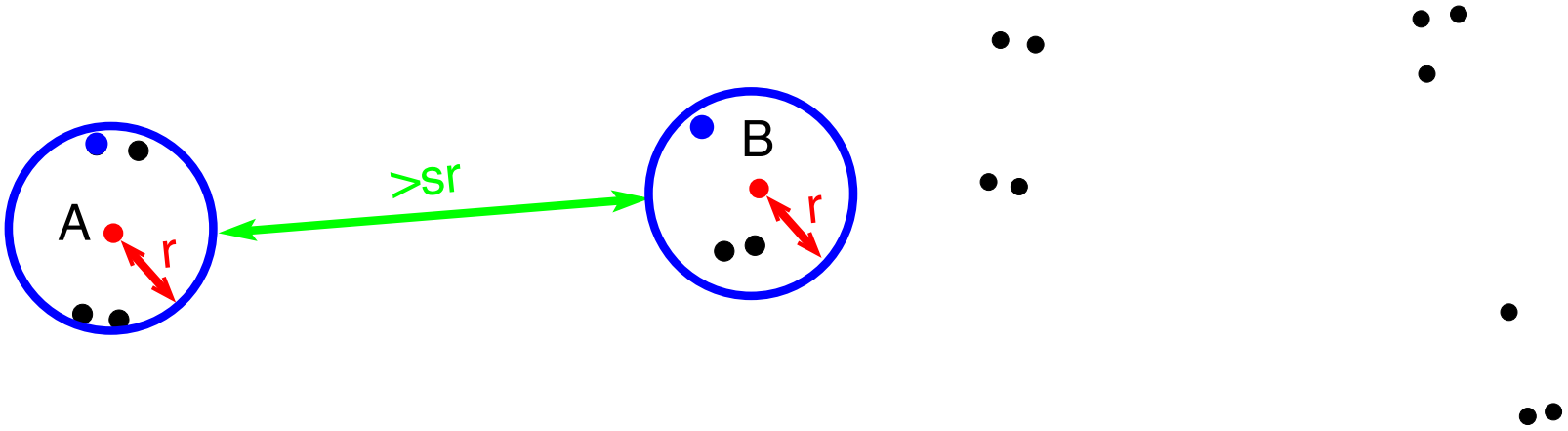
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r

Definition of Well-Separated Pairs (WSPs)



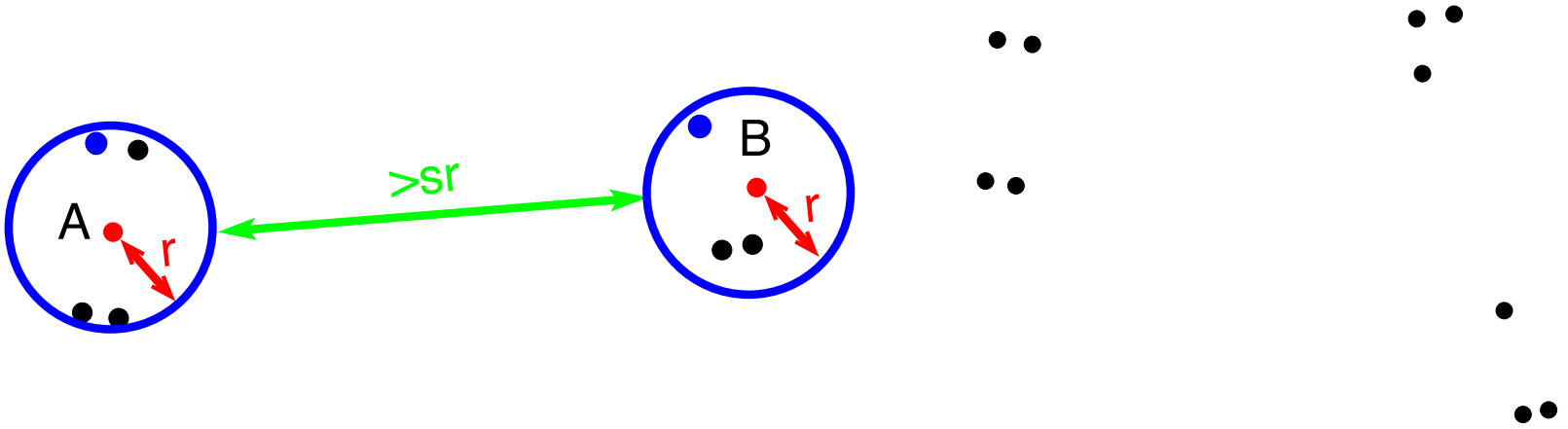
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated

Definition of Well-Separated Pairs (WSPs)



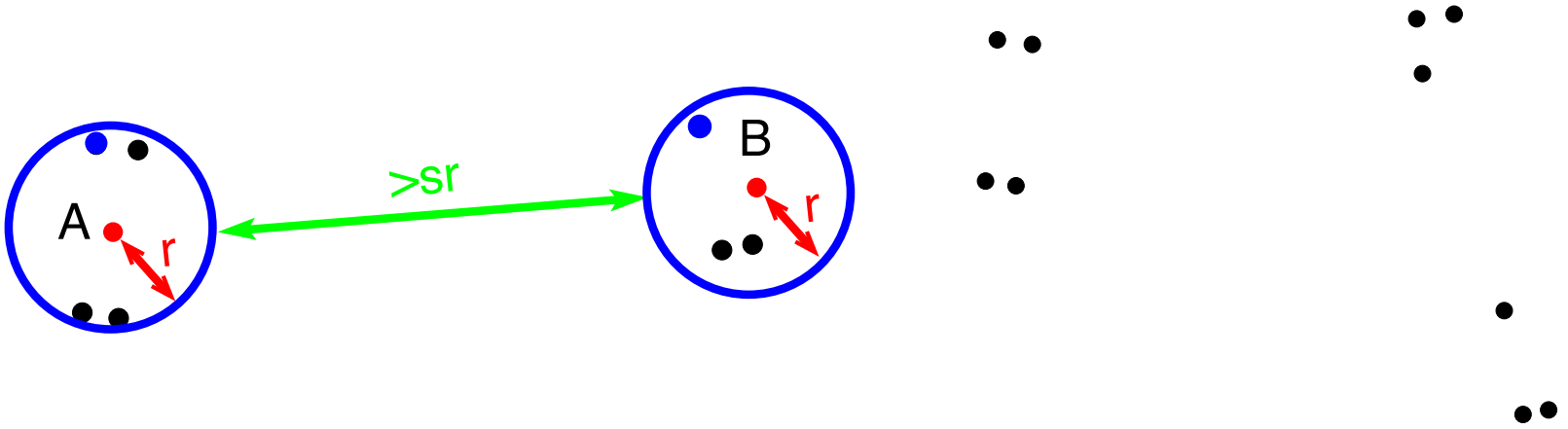
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)

Definition of Well-Separated Pairs (WSPs)



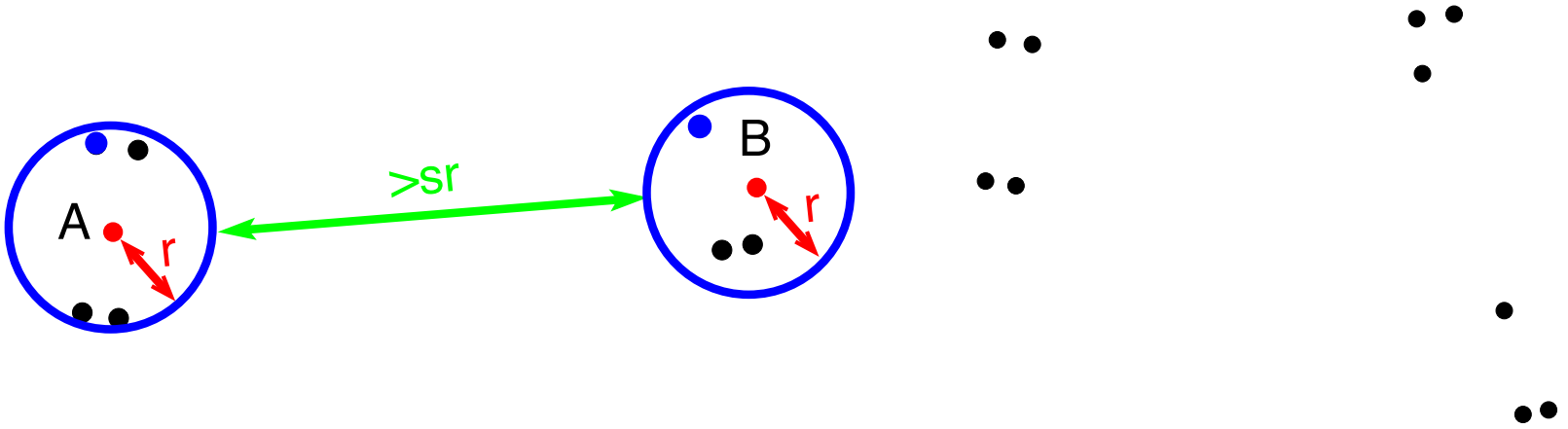
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs

Definition of Well-Separated Pairs (WSPs)



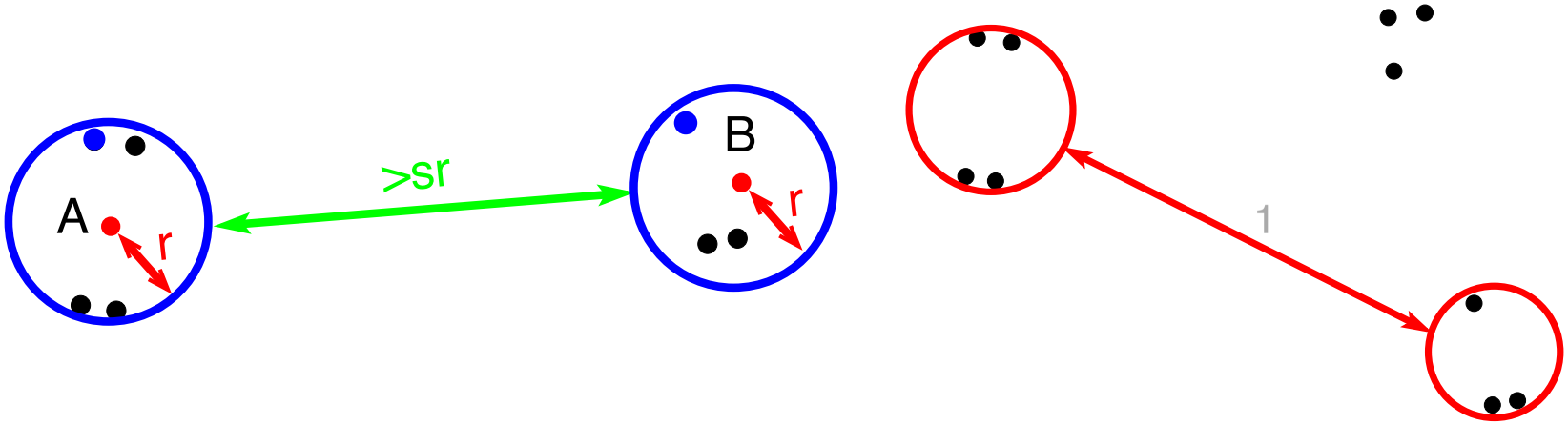
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s

Definition of Well-Separated Pairs (WSPs)



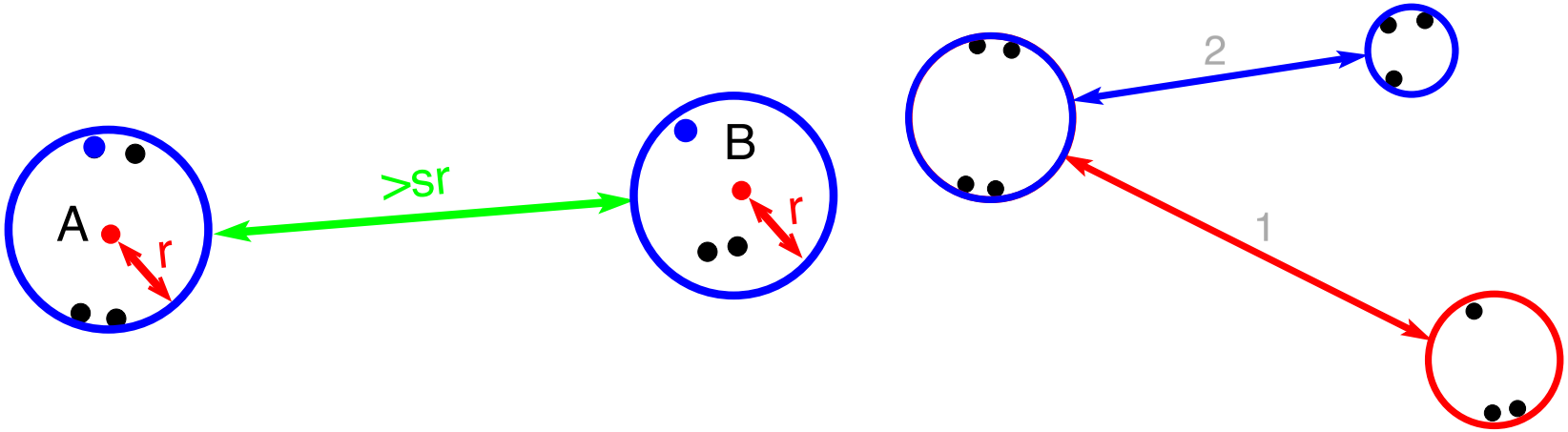
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



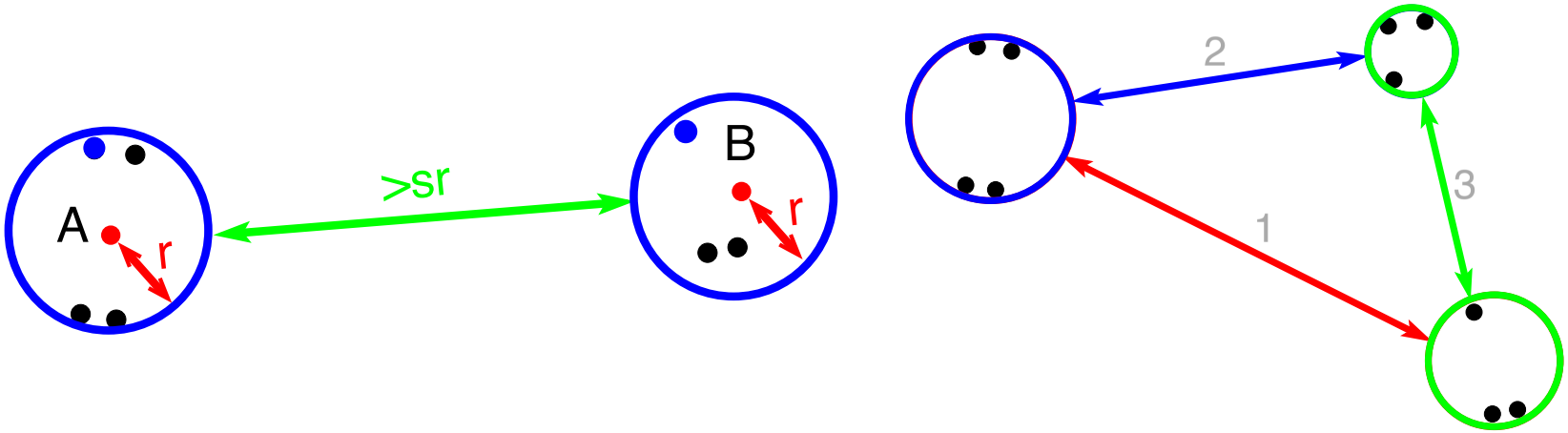
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



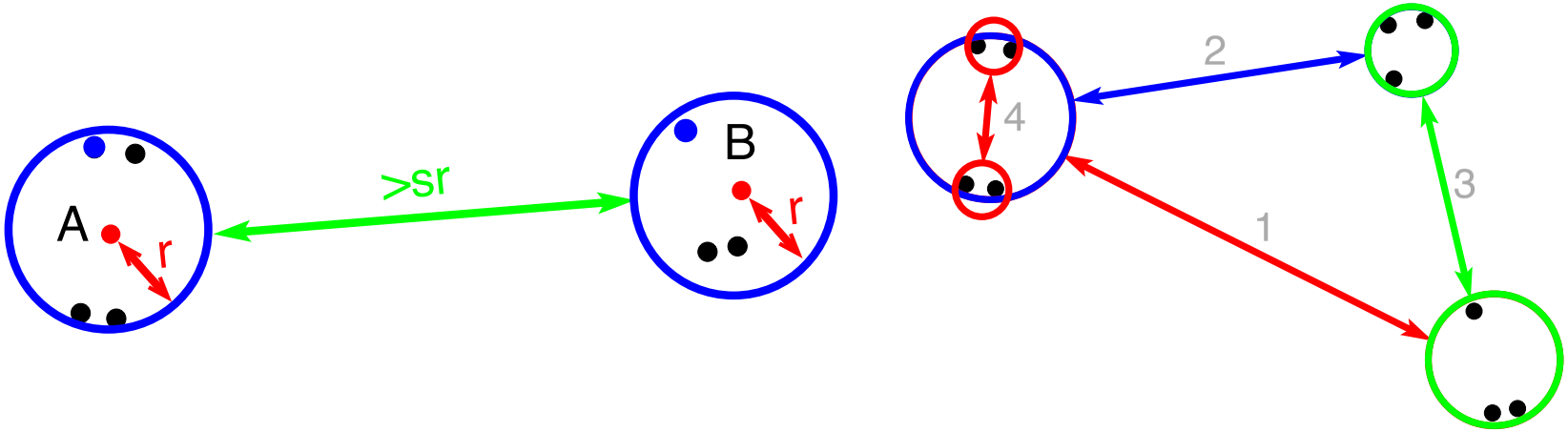
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



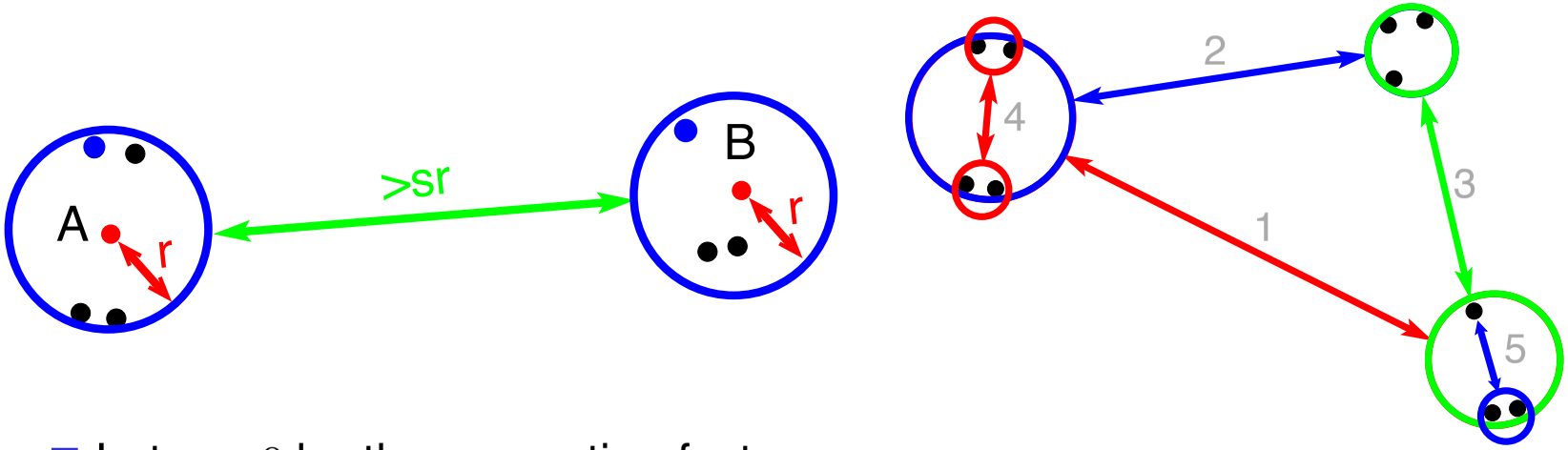
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



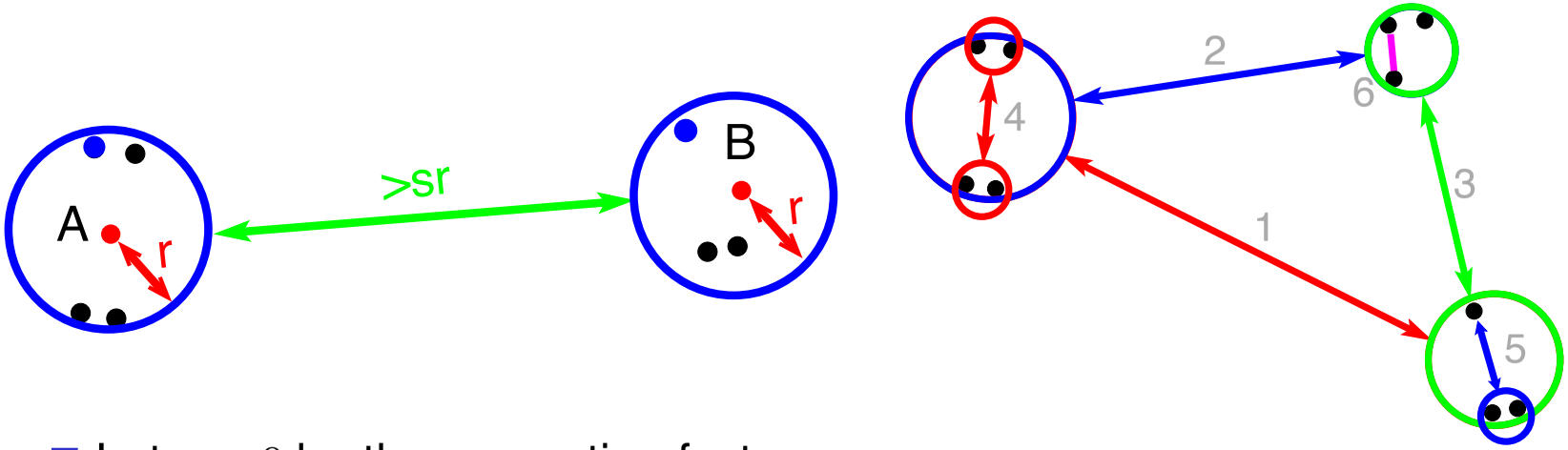
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



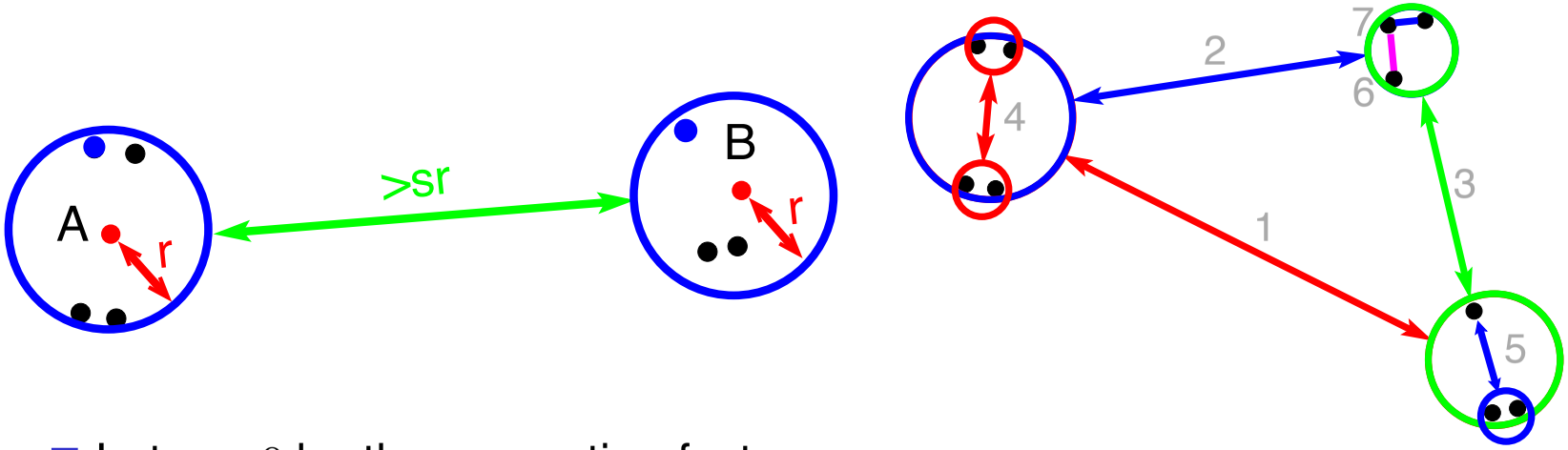
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



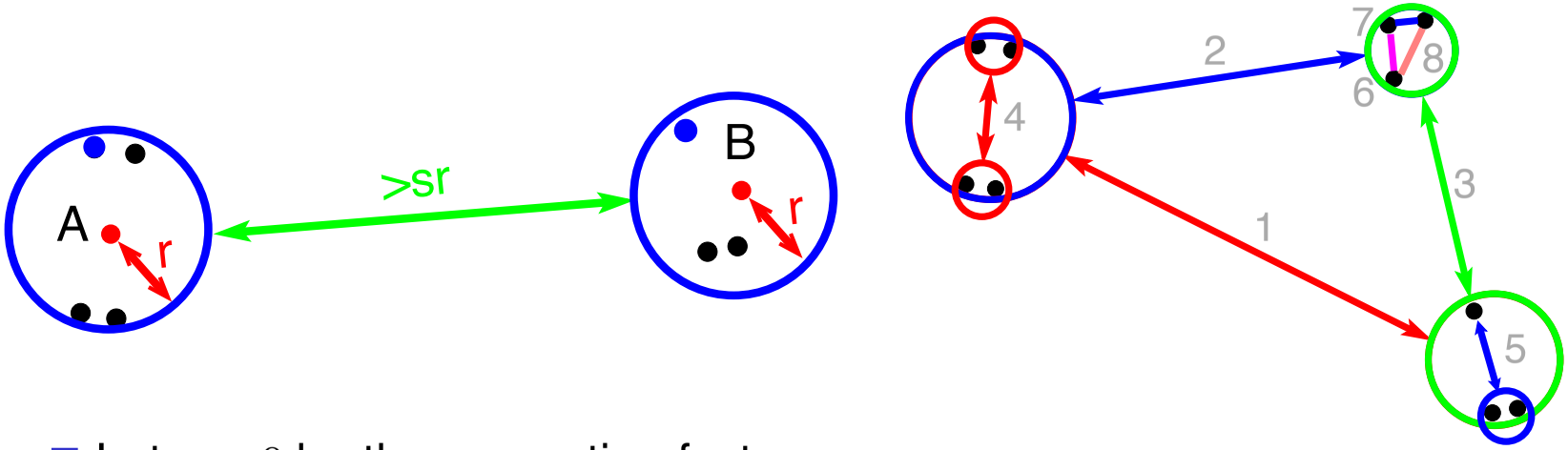
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



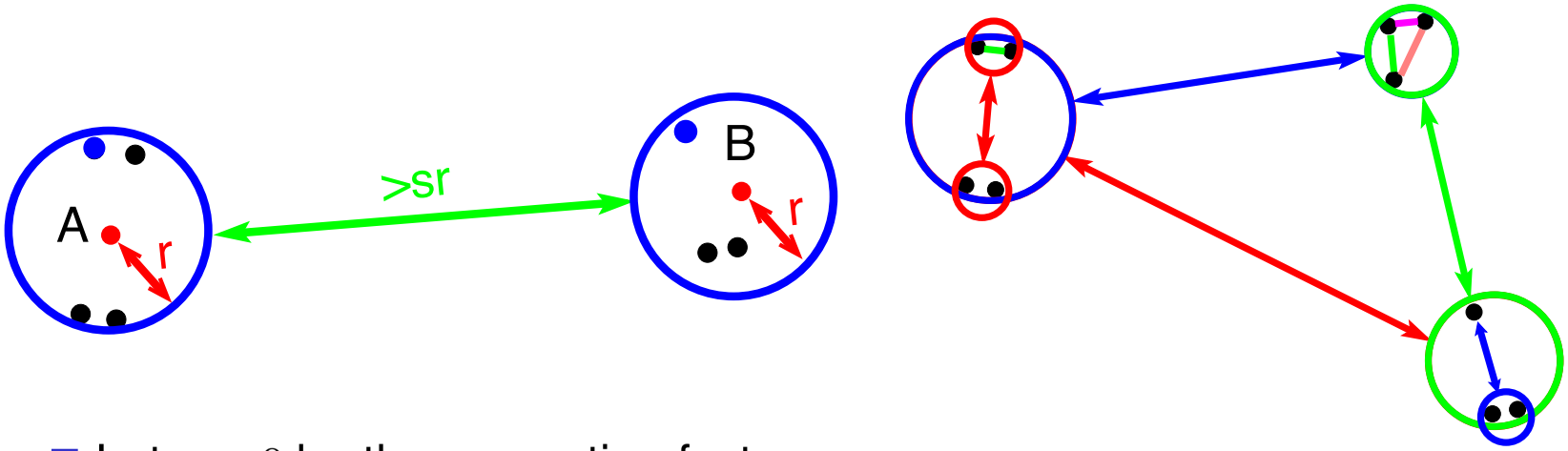
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



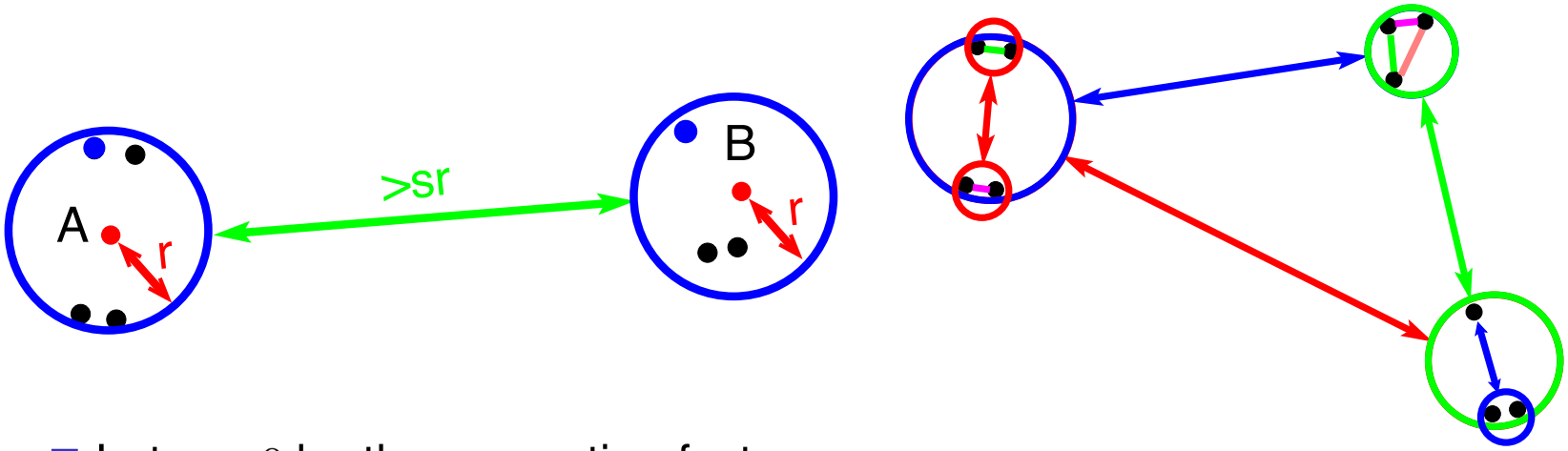
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



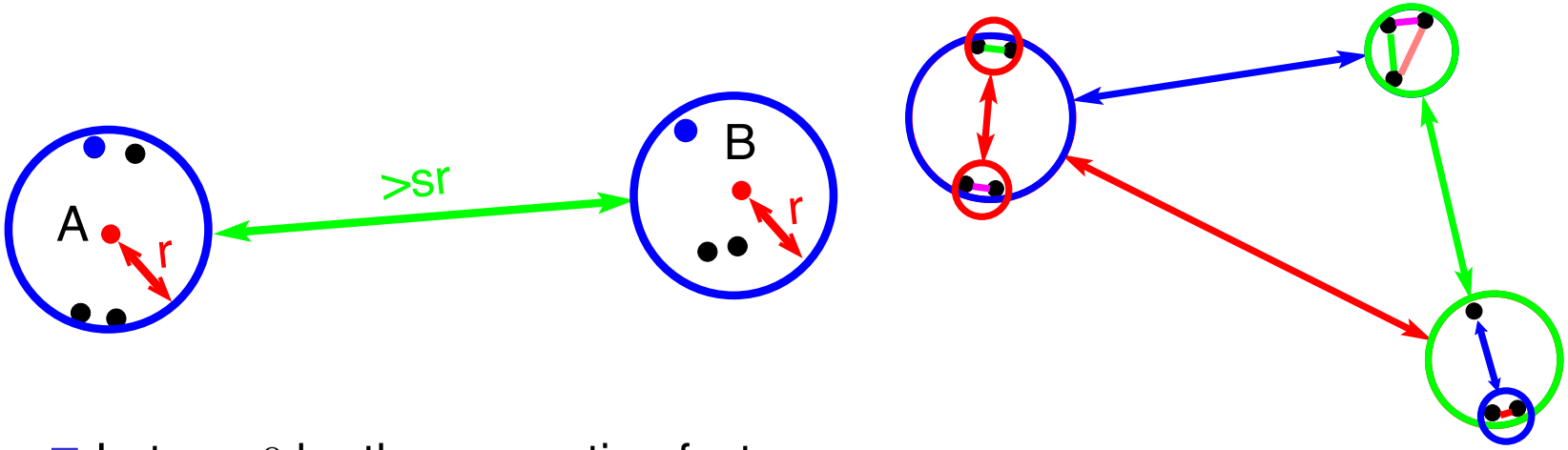
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



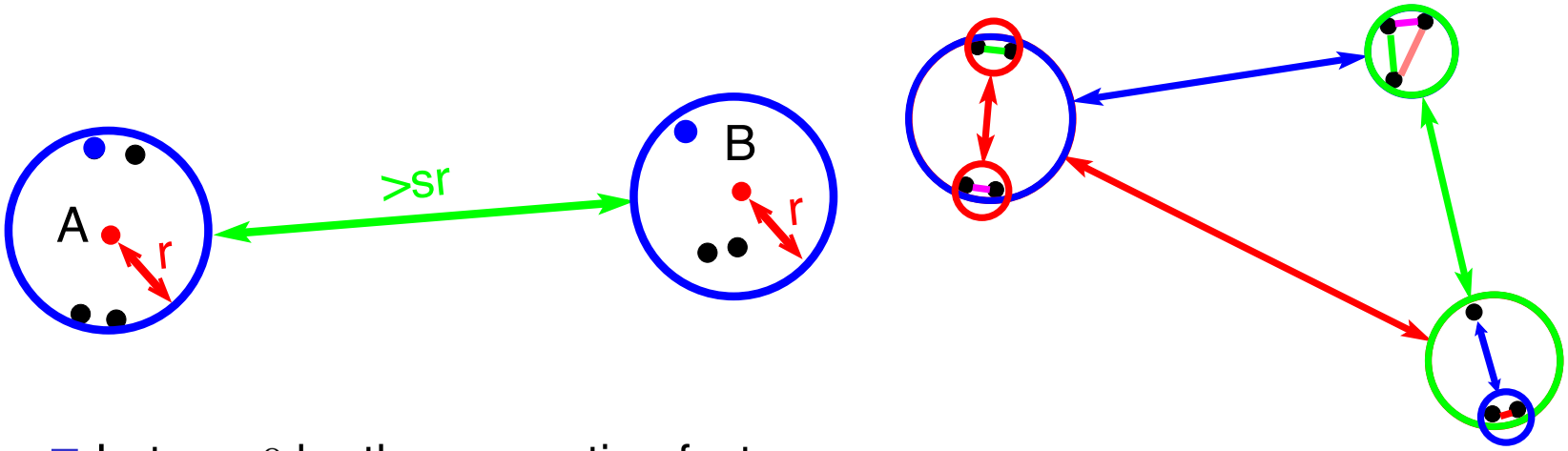
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



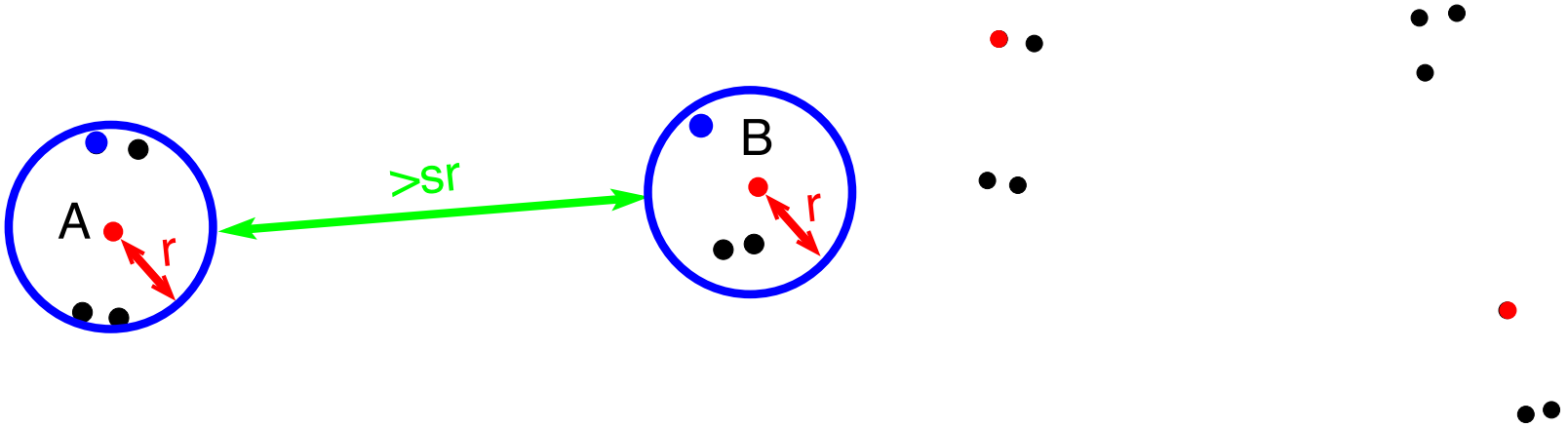
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs

Definition of Well-Separated Pairs (WSPs)



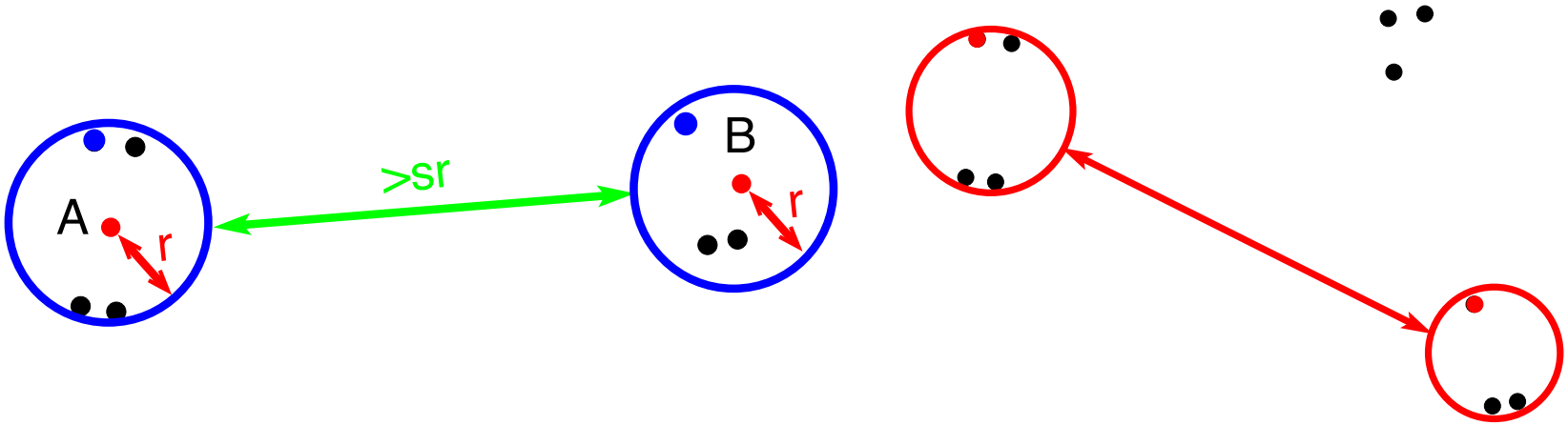
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs for a total of 11

Definition of Well-Separated Pairs (WSPs)



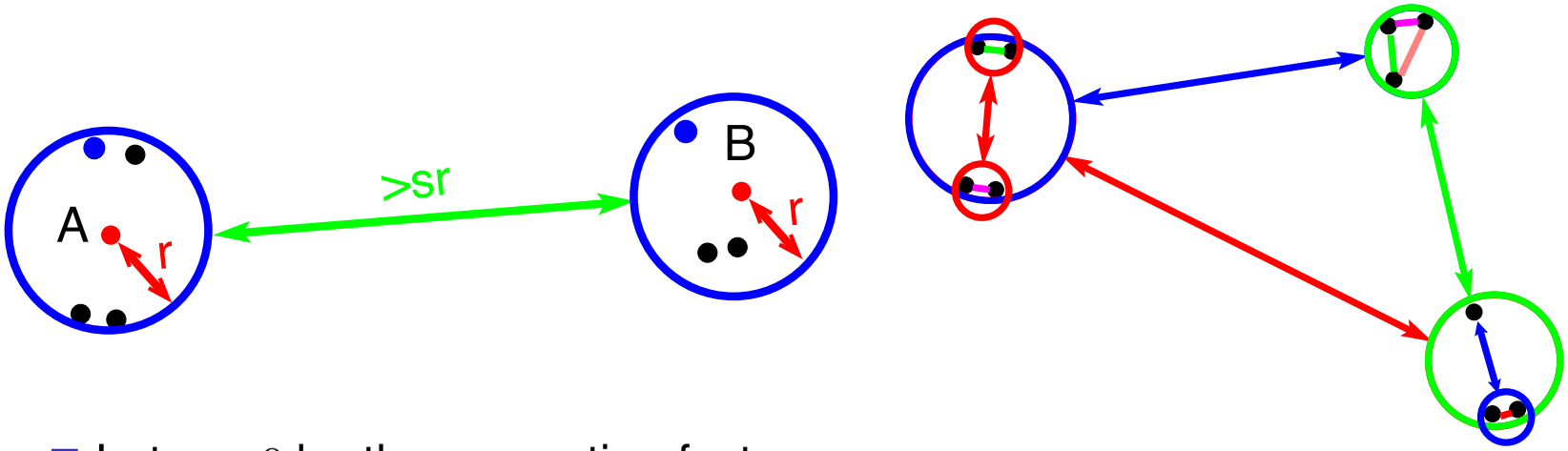
- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs for a total of 11
  - Any pair of points in S is contained in exactly one WSP

Definition of Well-Separated Pairs (WSPs)



- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs for a total of 11
  - Any pair of points in S is contained in exactly one WSP

Definition of Well-Separated Pairs (WSPs)



- Let $s > 0$ be the separation factor
- Two sets of points A and B , each contained in a hypersphere of radius r are well-separated if their closest approach is greater than sr, resulting in a Well-Separated Pair (WSP)
- Goal: Break up a set of points into WSPs
- Given point set S:R^d of size n and a given value of separation factor s
  - Decompose S with 10 points into WSPs for a total of 11
  - Any pair of points in S is contained in exactly one WSP
- WSP decomposition (WSPD) contains $O(s^d n)$ WSPs [Callahan et al., 1995]

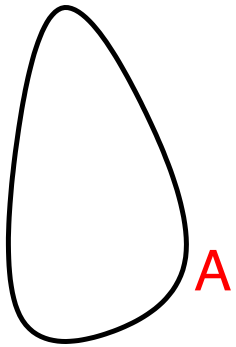
PCP Decomposition

PCP Decomposition

- Suppose that

PCP Decomposition

- Suppose that



PCP Decomposition

- Suppose that A

PCP Decomposition

- Suppose that A B

PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$

PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1

PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2,

PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1

PCP Decomposition

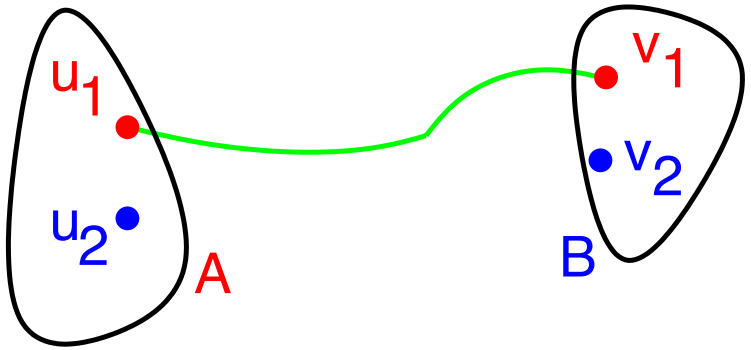
- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2,

PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t

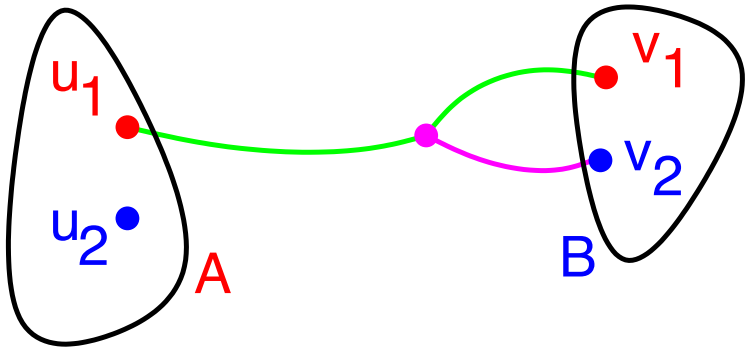
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t



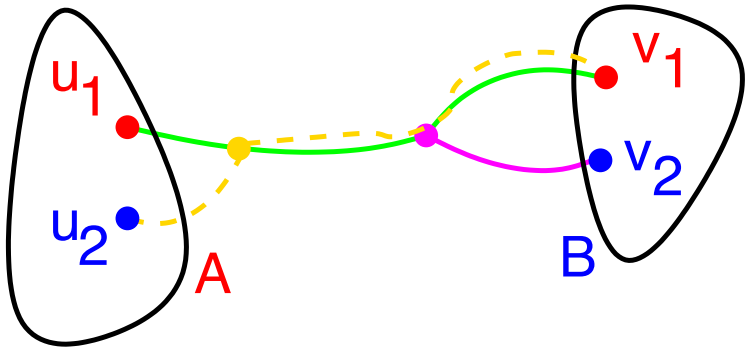
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t



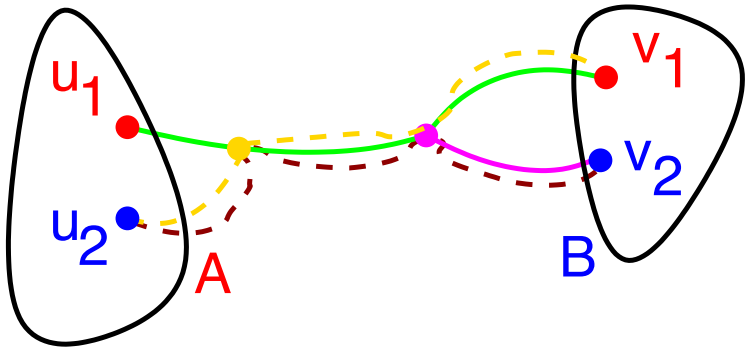
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t



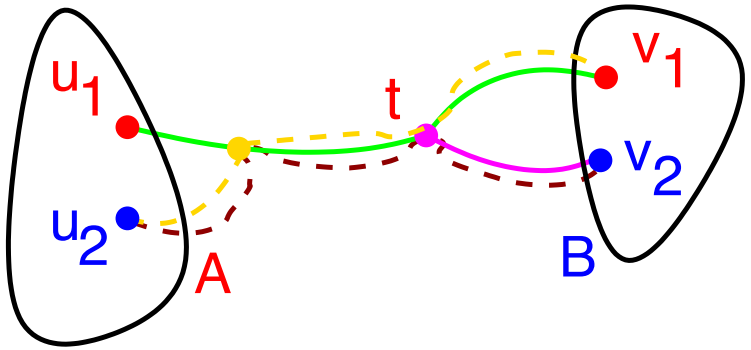
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t



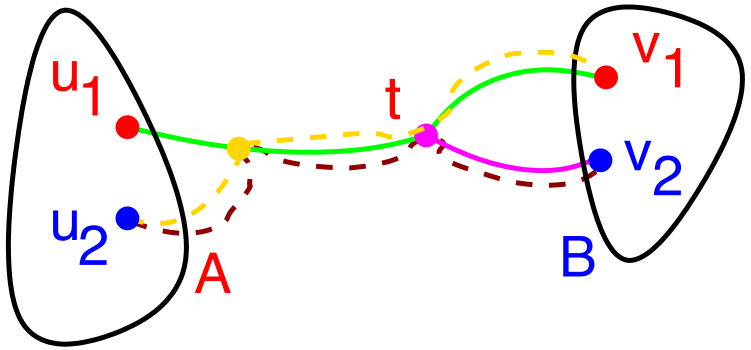
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t



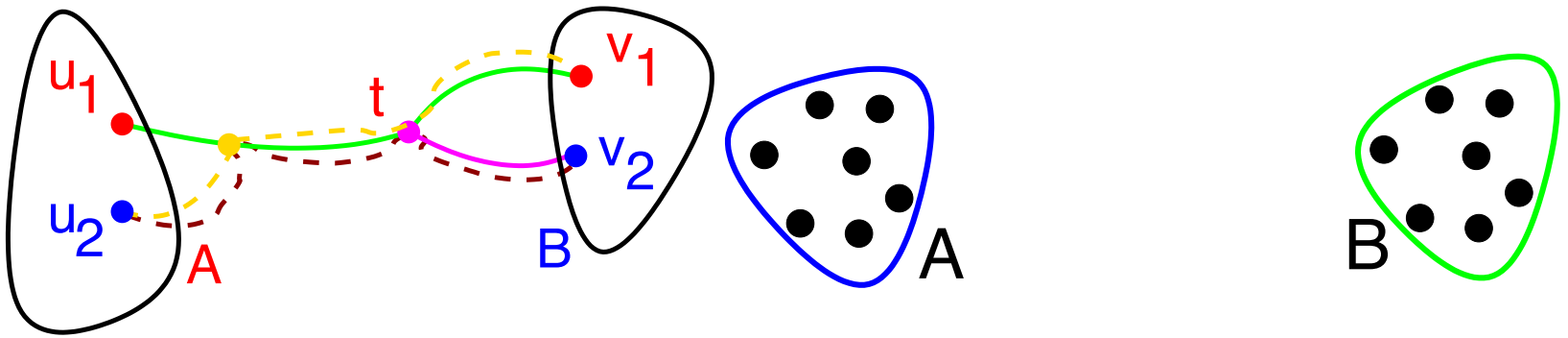
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all



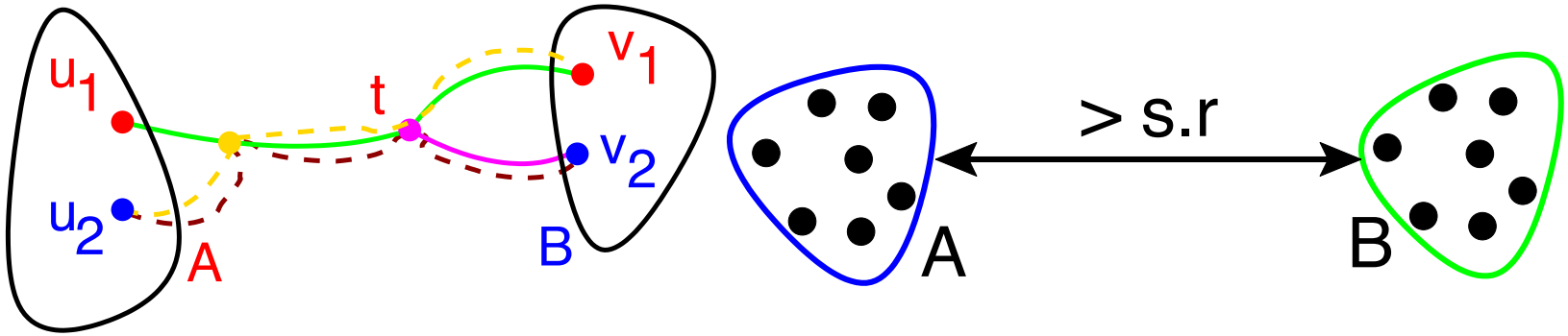
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all sources in A



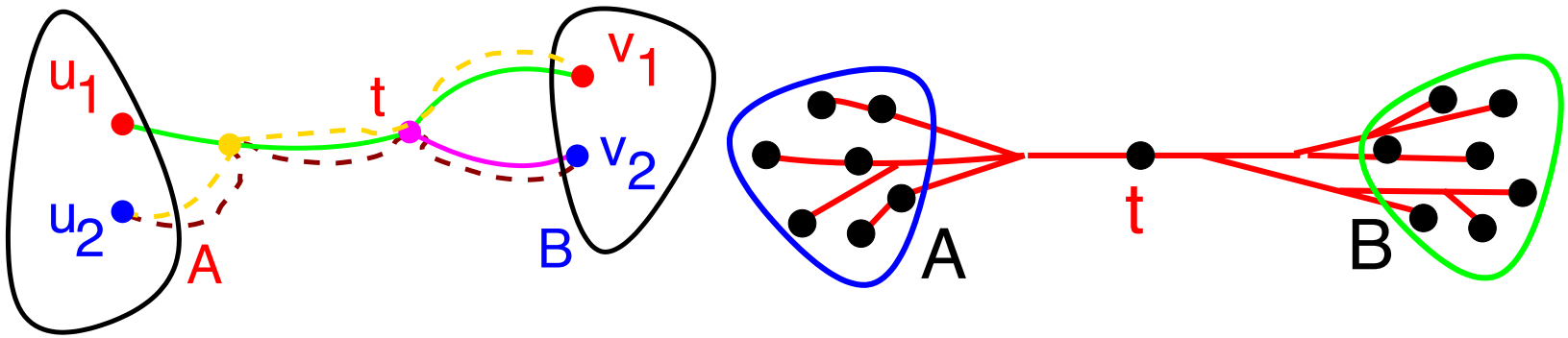
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all sources in A to destinations in B pass



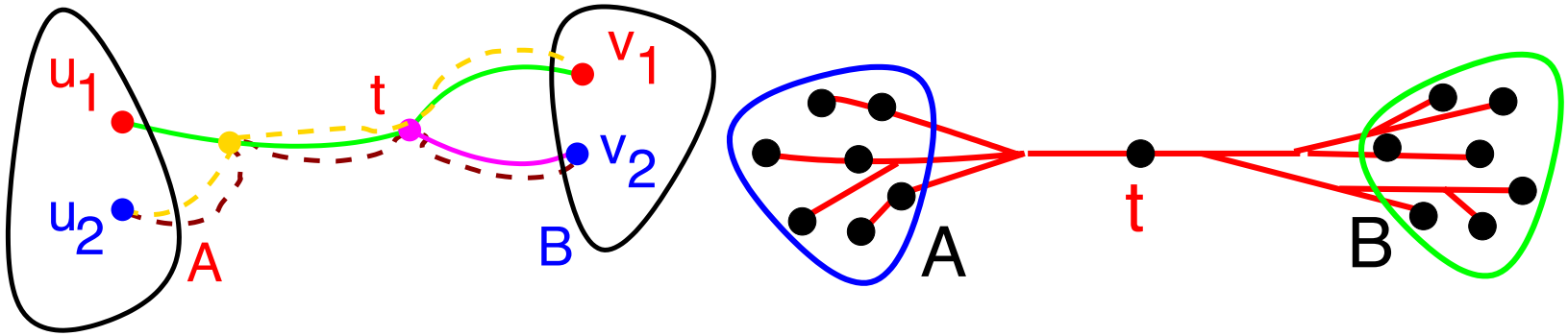
PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta - 1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all sources in A to destinations in B pass through a common vertex t

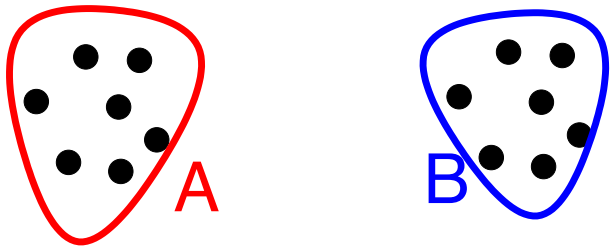


PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta-1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all sources in A to destinations in B pass through a common vertex t

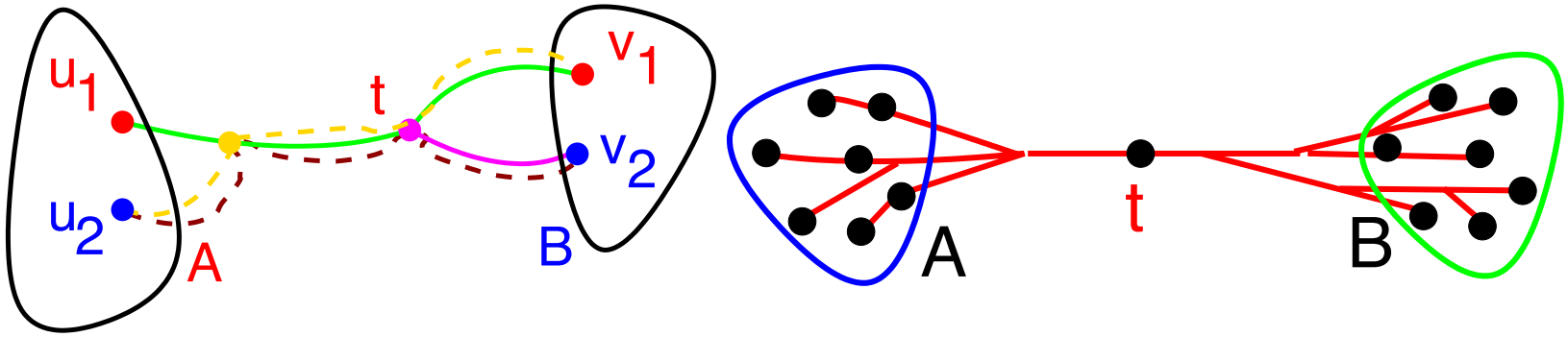


- If $s > \frac{2\gamma_H}{\epsilon\gamma_L}$, then all the network distances from A to B

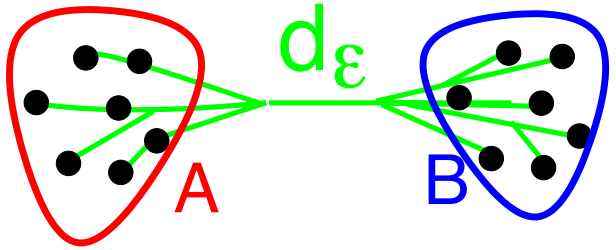


PCP Decomposition

- Suppose that A B are subset of vertices form a s-WSP
  - WSP decomposition is a PCP decomposition for $s > (2 + \frac{1}{(\delta-1)}) \cdot \frac{\gamma_H}{\gamma_L}$
    1. Given source vertices u_1 and u_2, destination vertices v_1 and v_2, four shortest paths pass through a common vertex t
    2. Extending #1, the shortest paths from all sources in A to destinations in B pass through a common vertex t



- If $s > \frac{2\gamma_H}{\epsilon\gamma_L}$, then all the network distances from A to B can be ϵ-approximated by d_ϵ

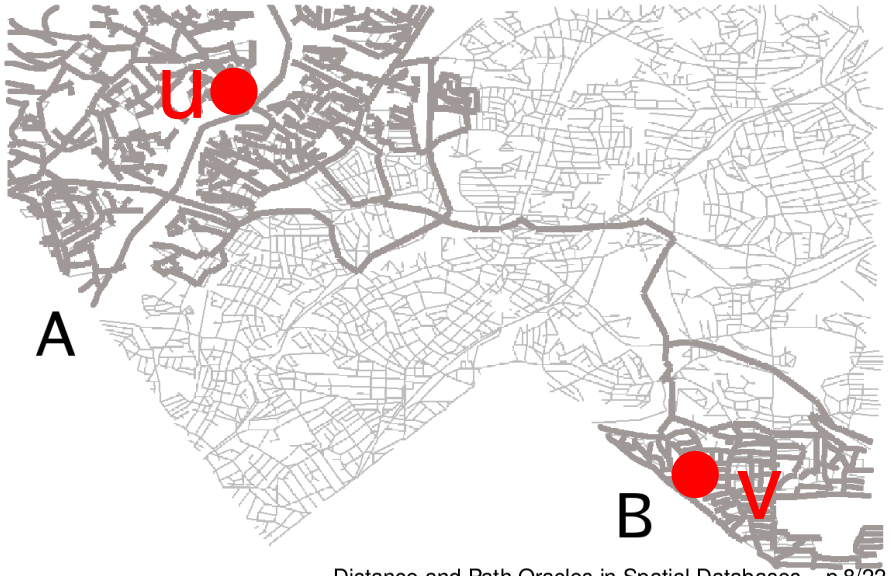


Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given

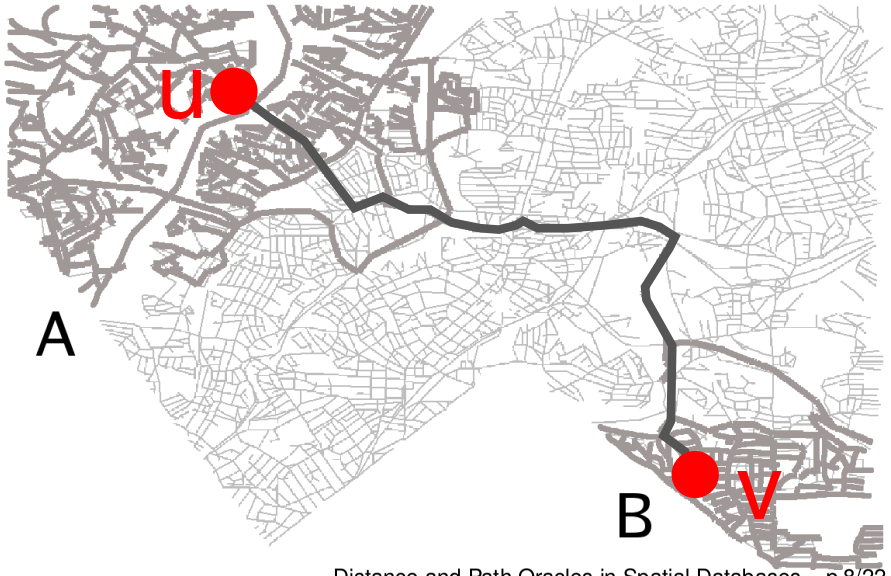
Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs



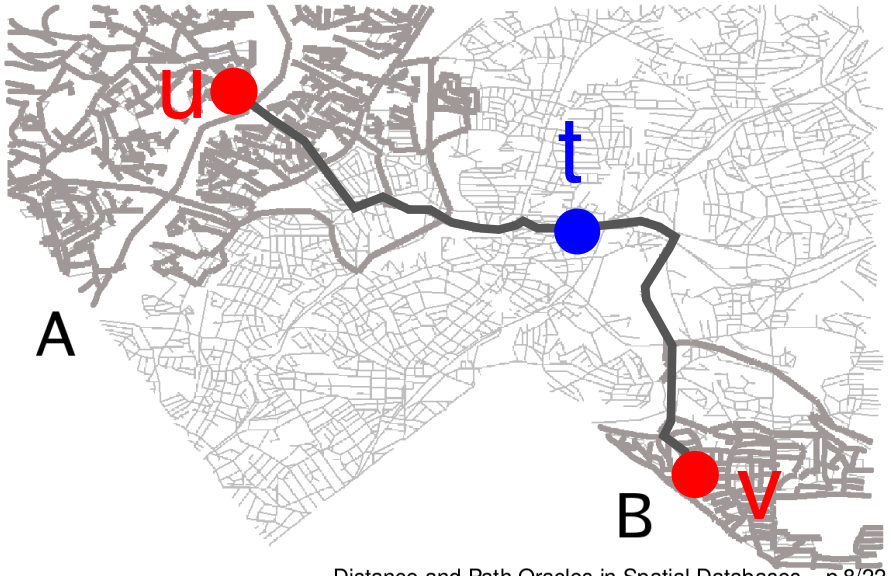
Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs



Path, Distance and Path-Distance Oracles

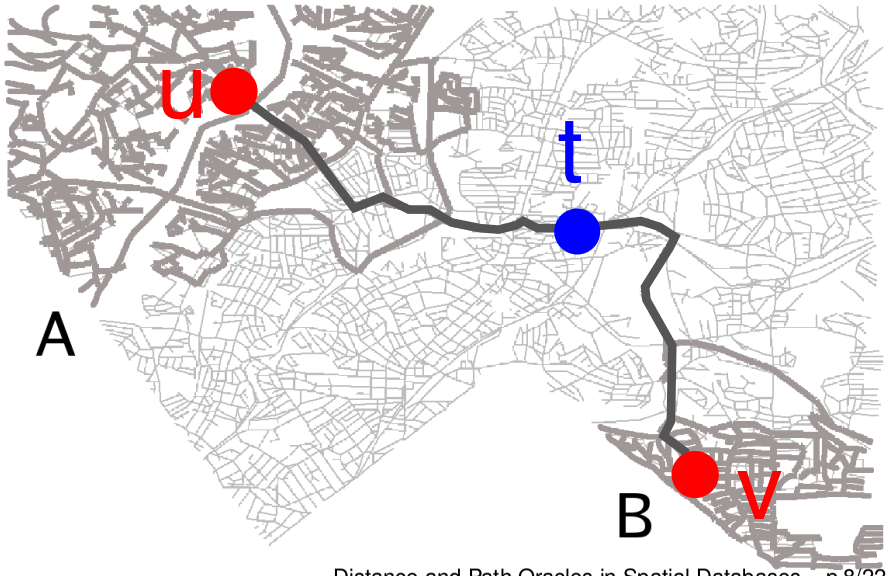
- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]



Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]

$O(s^2 n) \approx 12^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(s^2 n \log n)$	Hash	$O(1)$

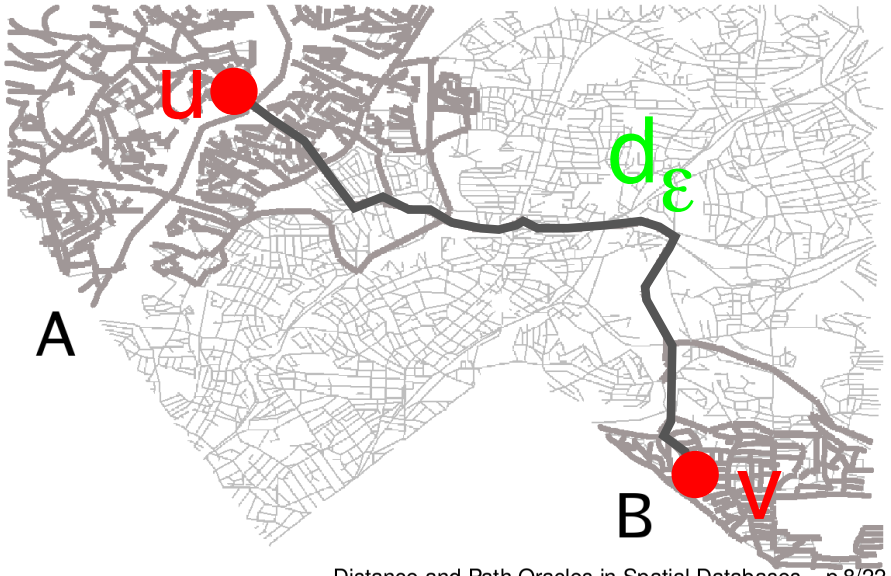


Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]

$O(s^2 n) \approx 12^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(s^2 n \log n)$	Hash	$O(1)$

  2. Distance oracle returns network distance d_ϵ allowing for a small bounded error $\epsilon \in (0, 1]$, say 5% [ICDE 2009]



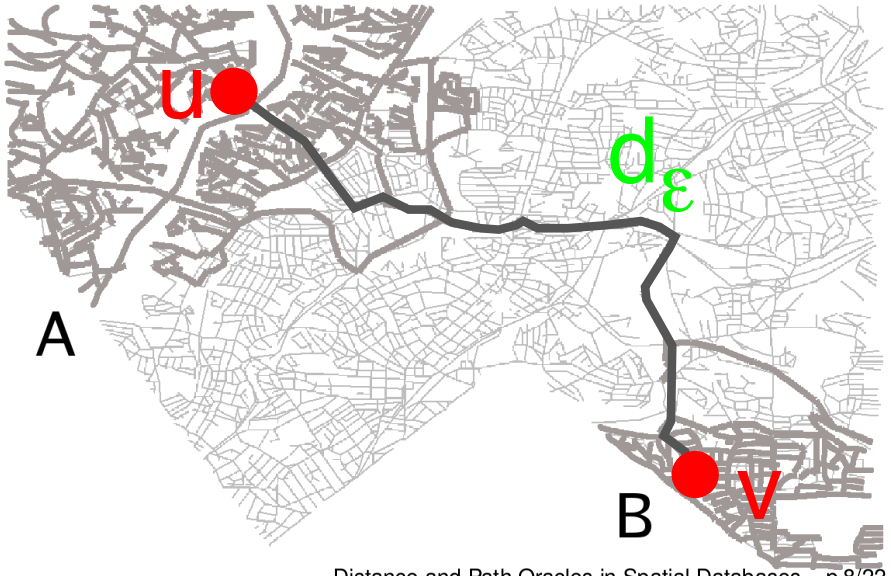
Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]

$O(s^2 n) \approx 12^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(s^2 n \log n)$	Hash	$O(1)$

  2. Distance oracle returns network distance d_ϵ allowing for a small bounded error $\epsilon \in (0, 1]$, say 5% [ICDE 2009]

$O(\frac{1}{\epsilon}^2 n) \approx 2.5\frac{1}{\epsilon}^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(\frac{1}{\epsilon}^2 n \log n)$	Hash	$O(1)$



Path, Distance and Path-Distance Oracles

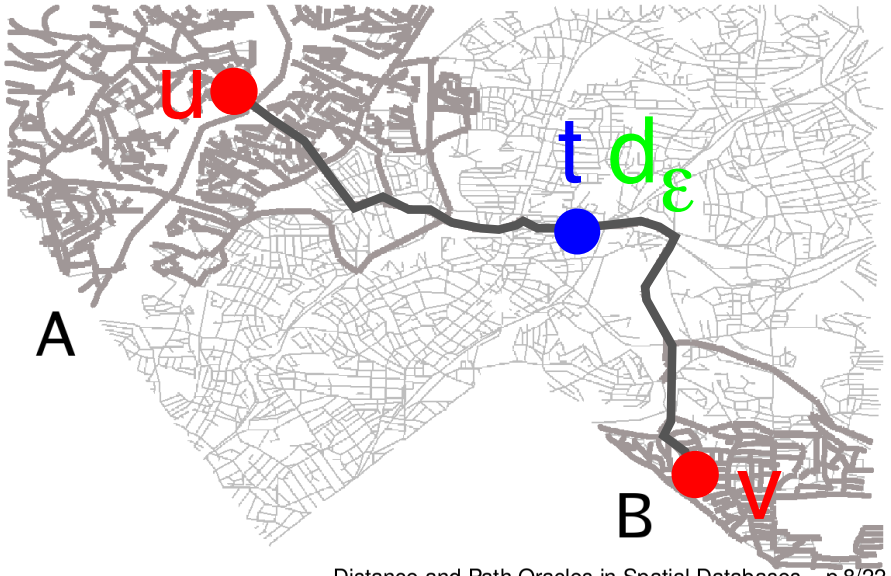
- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]

$O(s^2 n) \approx 12^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(s^2 n \log n)$	Hash	$O(1)$

  2. Distance oracle returns network distance d_ϵ allowing for a small bounded error $\epsilon \in (0, 1]$, say 5% [ICDE 2009]

$O(\frac{1}{\epsilon}^2 n) \approx 2.5\frac{1}{\epsilon}^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(\frac{1}{\epsilon}^2 n \log n)$	Hash	$O(1)$

  3. Path-Distance oracle returns both an intermediate vertex and ϵ-approximate network distance [VLDB 2009]



Path, Distance and Path-Distance Oracles

- Oracle: Database relation that given a source u and destination v as inputs
  1. Path oracle returns an intermediate vertex t [VLDB 2009]

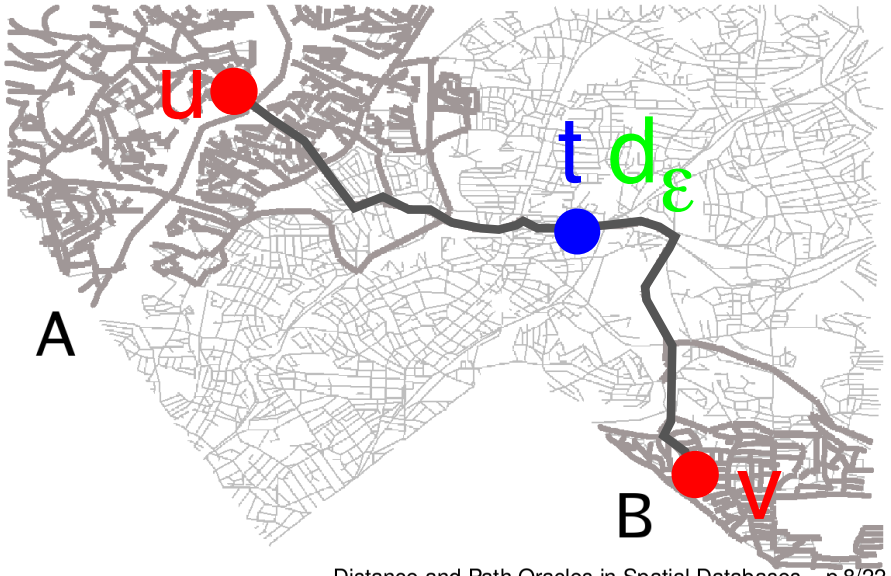
$O(s^2 n) \approx 12^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(s^2 n \log n)$	Hash	$O(1)$

  2. Distance oracle returns network distance d_ϵ allowing for a small bounded error $\epsilon \in (0, 1]$, say 5% [ICDE 2009]

$O(\frac{1}{\epsilon}^2 n) \approx 2.5\frac{1}{\epsilon}^2 n$	B-tree	$O(\log n) \approx$ 10s of μsec
$O(\frac{1}{\epsilon}^2 n \log n)$	Hash	$O(1)$

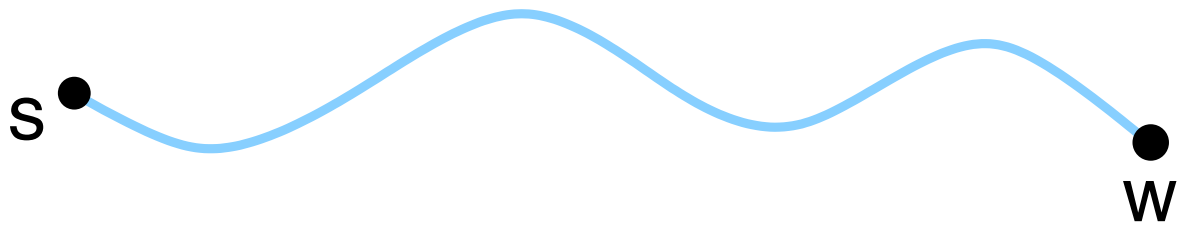
  3. Path-Distance oracle returns both an intermediate vertex and ϵ-approximate network distance [VLDB 2009]

$O(\max(s, \frac{1}{\epsilon})^2 n)$	B-tree	$O(\log n)$
$O(\max(s, \frac{1}{\epsilon})^2 n \log n)$	Hash	$O(1)$



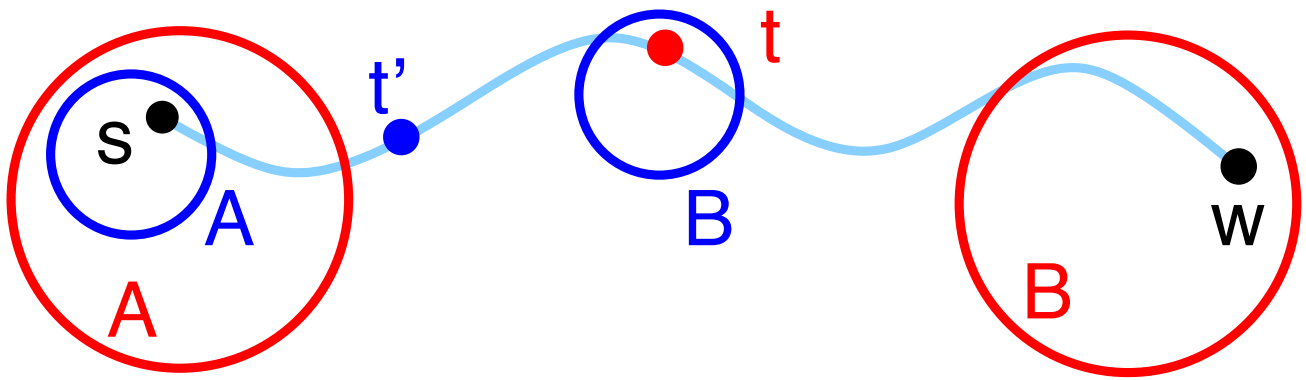
Shortest Path Retrieval

- Find the shortest path from a source p to destination q:



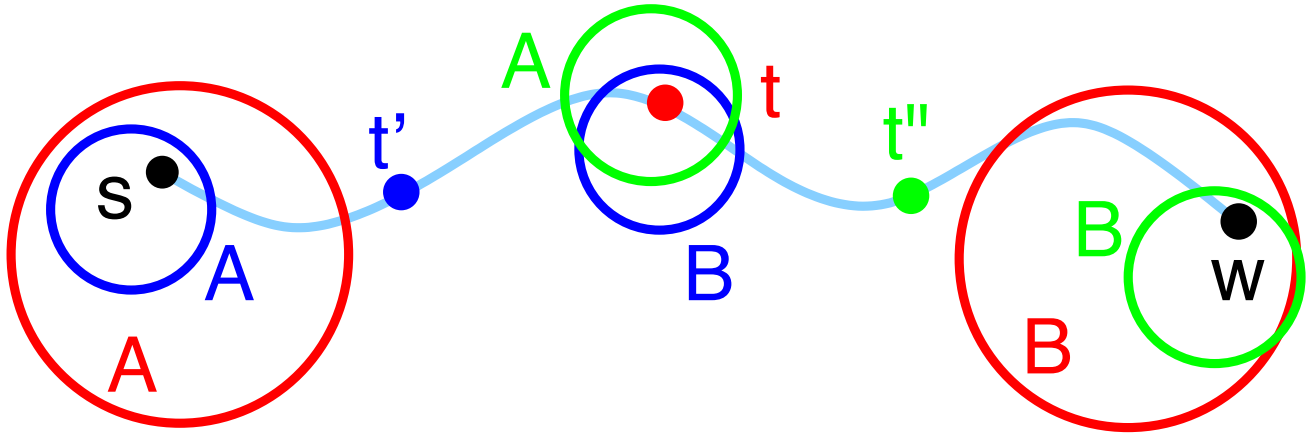
Shortest Path Retrieval

- Find the shortest path from a source p to destination q:
  1. Find the PCP (A,B,t) such that A contains p and B contains q



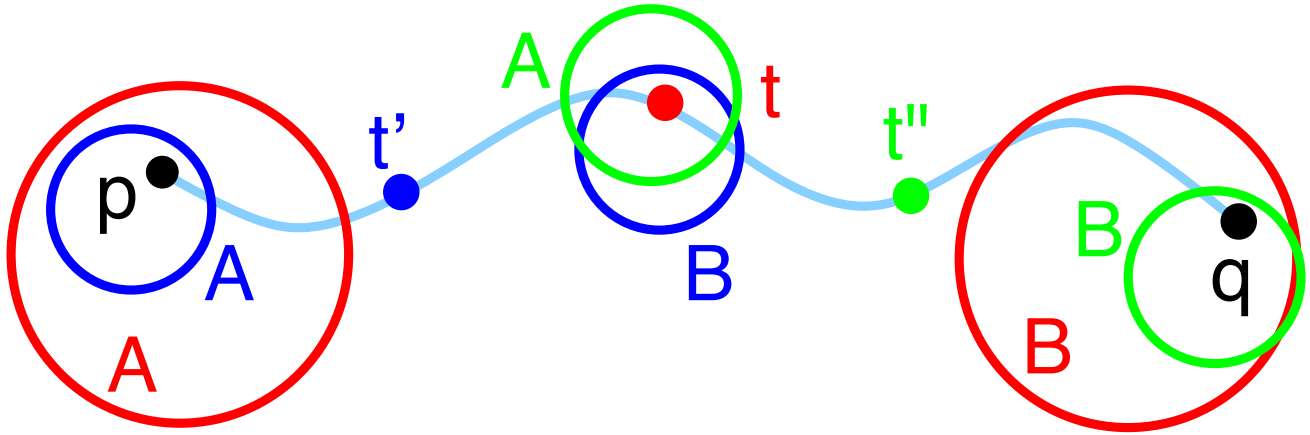
Shortest Path Retrieval

- Find the shortest path from a source p to destination q:
  1. Find the PCP (A,B,t) such that A contains p and B contains q
  2. Next, recursively, find PCP containing p and t yielding t'



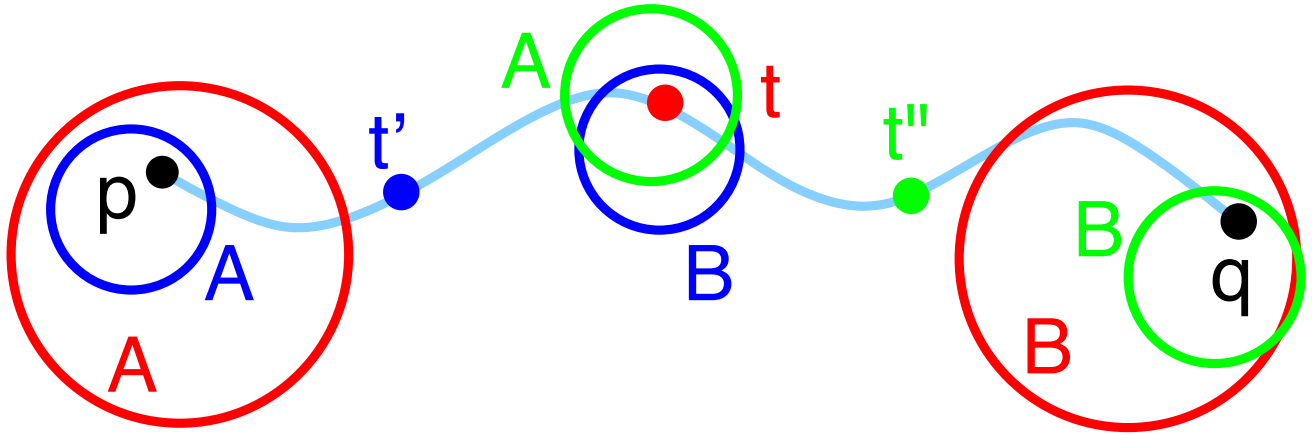
Shortest Path Retrieval

- Find the shortest path from a source p to destination q:
  1. Find the PCP (A,B,t) such that A contains p and B contains q
  2. Next, recursively, find PCP containing p and t yielding t'
     and containing t and q yielding t"



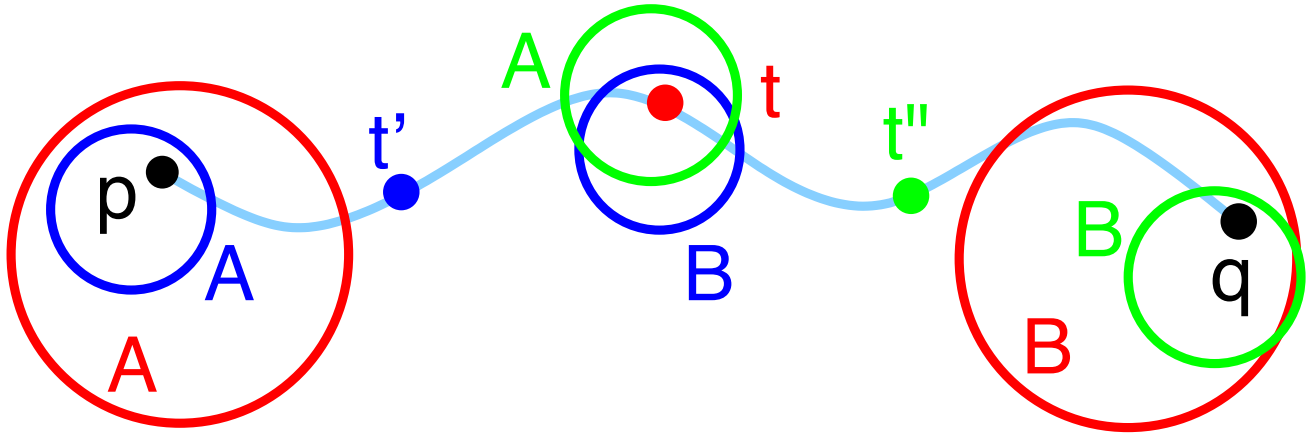
Shortest Path Retrieval

- Find the shortest path from a source p to destination q:
  1. Find the PCP (A,B,t) such that A contains p and B contains q
  2. Next, recursively, find PCP containing p and t yielding t'
     and containing t and q yielding t"
  3. . . .



Shortest Path Retrieval

- Find the shortest path from a source p to destination q:
  1. Find the PCP (A,B,t) such that A contains p and B contains q
  2. Next, recursively, find PCP containing p and t yielding t'
     and containing t and q yielding t"
  3. . . .
- Finding next vertex in shortest path takes $O(\log n)$ time

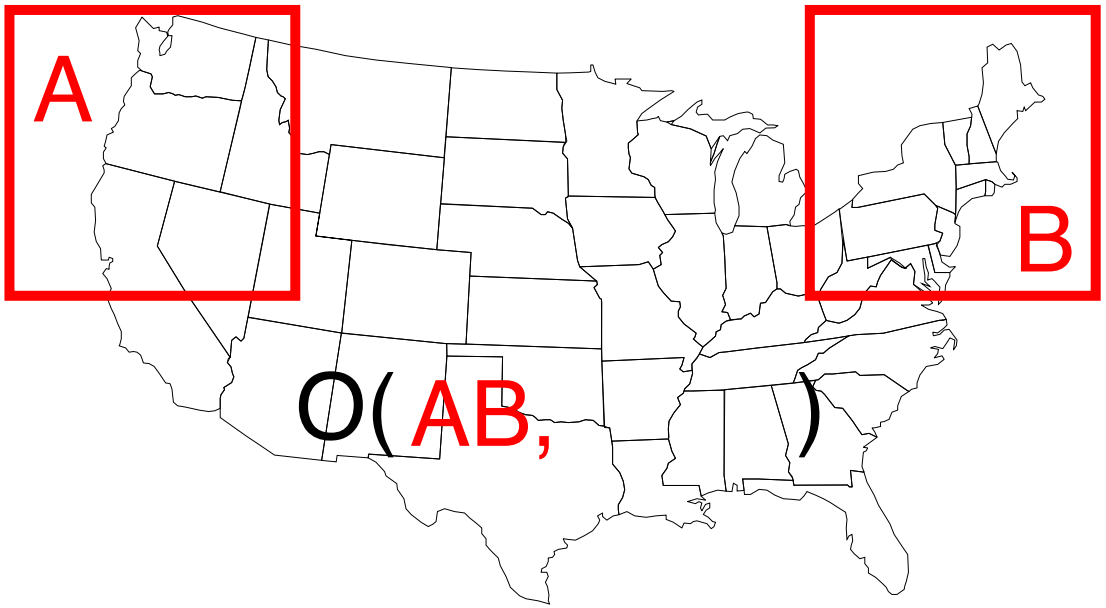


Query Processing using Oracles [TKDE 2010]

- Path-distance stored as a relation O
  - $O(n)$ tuples
  - Access time $O(\log n)$; tens of μsecond
- Schema of path-distance oracle: O(AB, Ψ, d_ϵ)

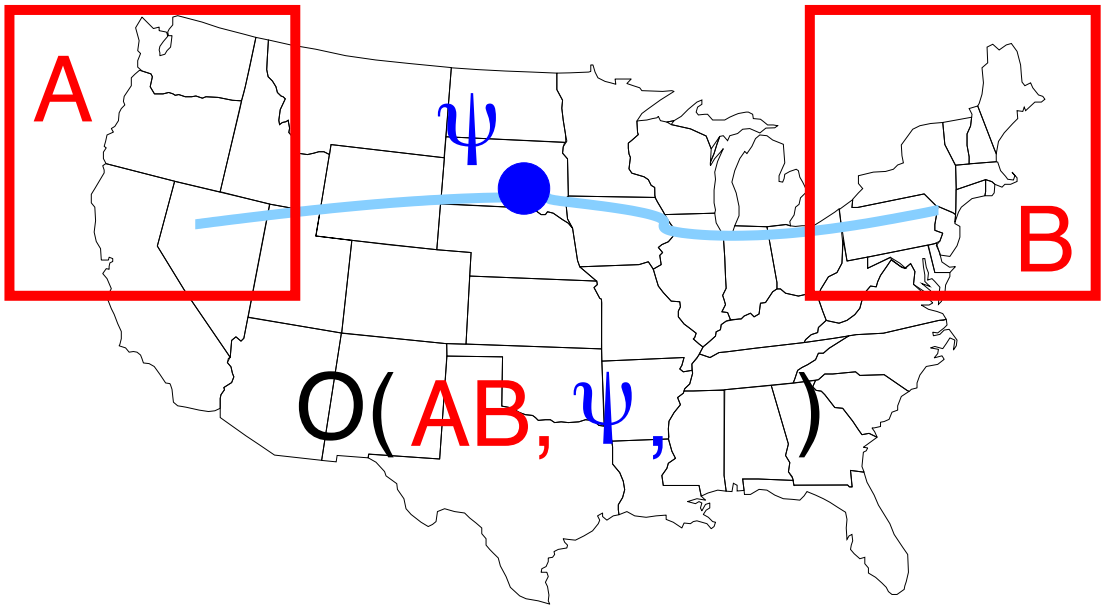
Query Processing using Oracles [TKDE 2010]

- Path-distance stored as a relation O
  - $O(n)$ tuples
  - Access time $O(\log n)$; tens of μsecond
- Schema of path-distance oracle: O(AB, Ψ, d_ϵ)
  1. AB: Four-dimensional Morton block indexed using a B-tree
     - A,B: Pair of blocks in a PR quadtree on V



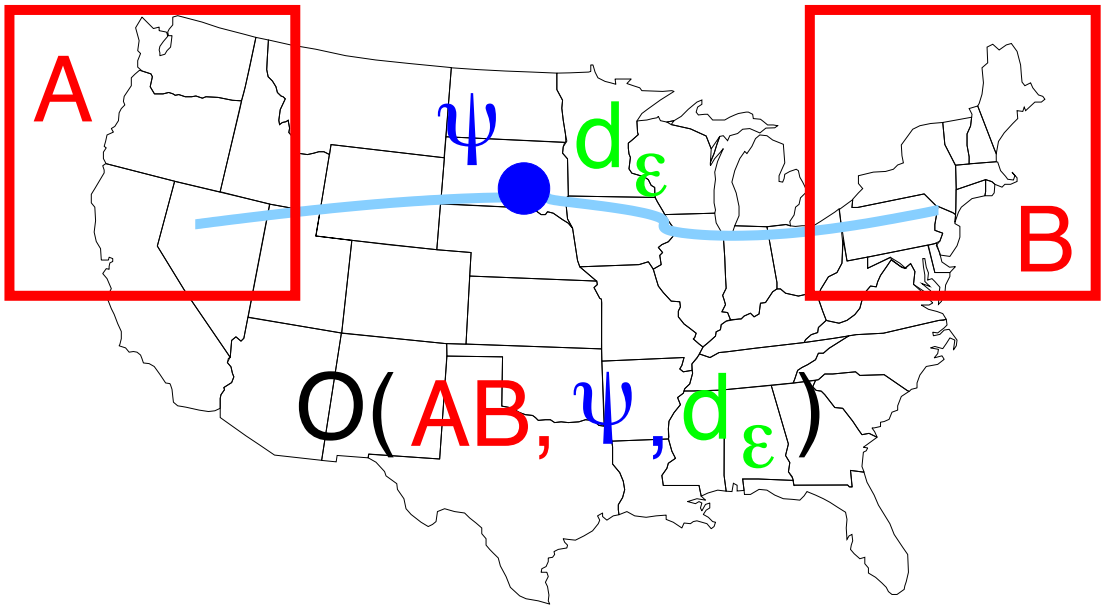
Query Processing using Oracles [TKDE 2010]

- Path-distance stored as a relation O
  - $O(n)$ tuples
  - Access time $O(\log n)$; tens of μsecond
- Schema of path-distance oracle: O(AB, Ψ, d_ϵ)
  1. AB: Four-dimensional Morton block indexed using a B-tree
     - A,B: Pair of blocks in a PR quadtree on V
  2. Ψ : intermediate vertex



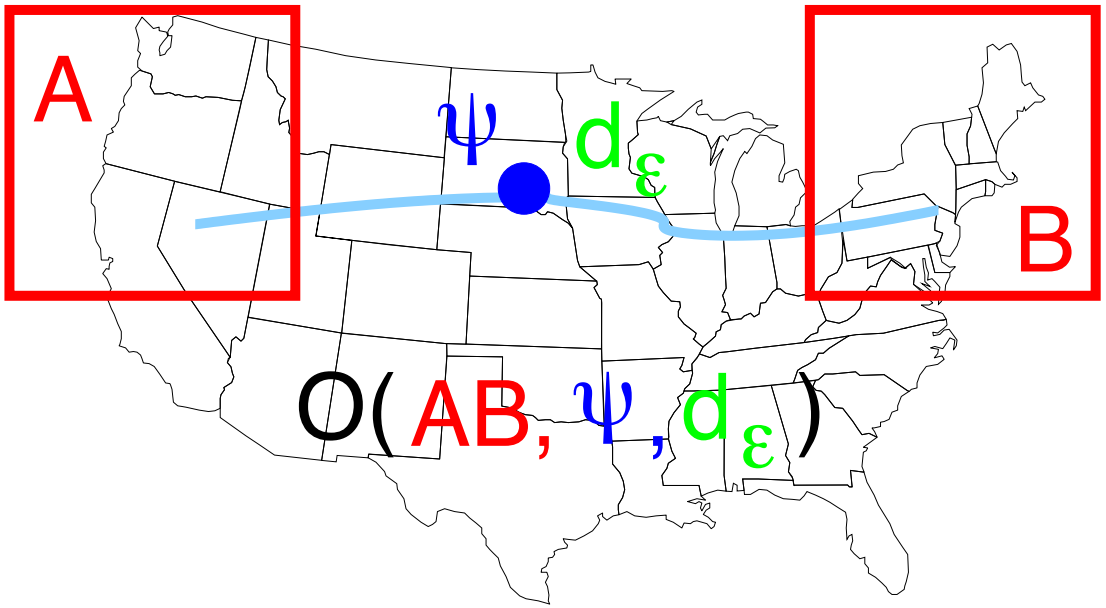
Query Processing using Oracles [TKDE 2010]

- Path-distance stored as a relation O
  - $O(n)$ tuples
  - Access time $O(\log n)$; tens of μsecond
- Schema of path-distance oracle: O(AB, Ψ, d_ϵ)
  1. AB: Four-dimensional Morton block indexed using a B-tree
     - A,B: Pair of blocks in a PR quadtree on V
  2. Ψ : intermediate vertex
  3. d_ϵ : approximate network distance

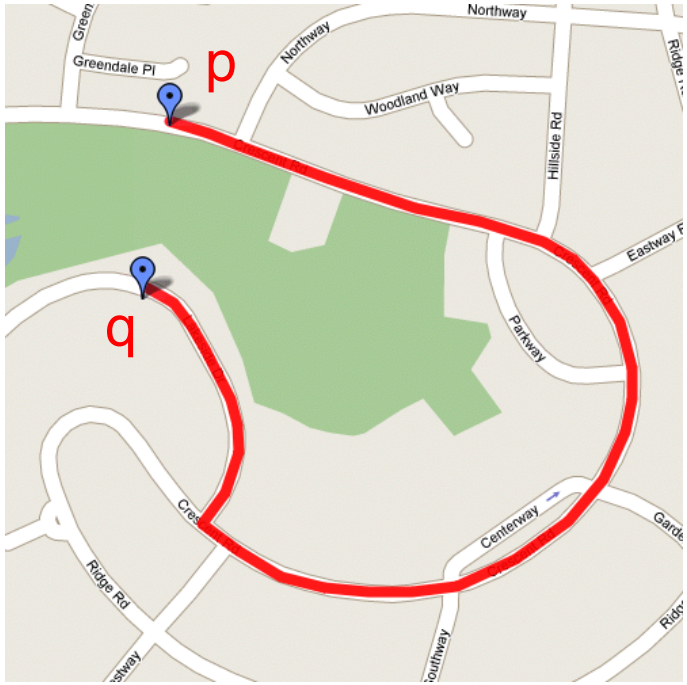


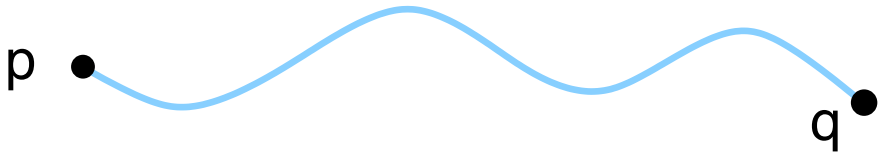
Query Processing using Oracles [TKDE 2010]

- Path-distance stored as a relation O
  - $O(n)$ tuples
  - Access time $O(\log n)$; tens of μsecond
- Schema of path-distance oracle: O(AB, Ψ, d_ϵ)
  1. AB: Four-dimensional Morton block indexed using a B-tree
     - A,B: Pair of blocks in a PR quadtree on V
  2. Ψ : intermediate vertex
  3. d_ϵ : approximate network distance
- Z_4: $(\mathbb{R}^2, \mathbb{R}^2) \rightarrow \mathbb{N}$: Function to convert points to Morton code



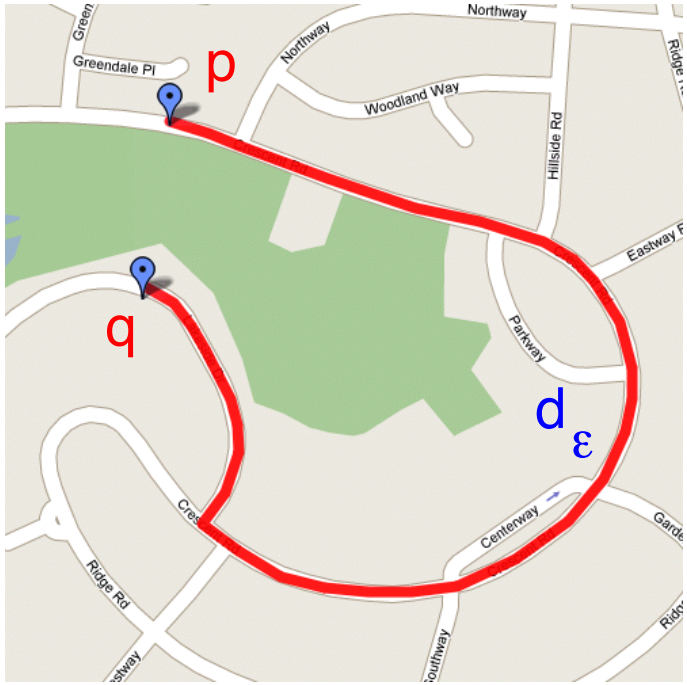
Path and Network Distance Query

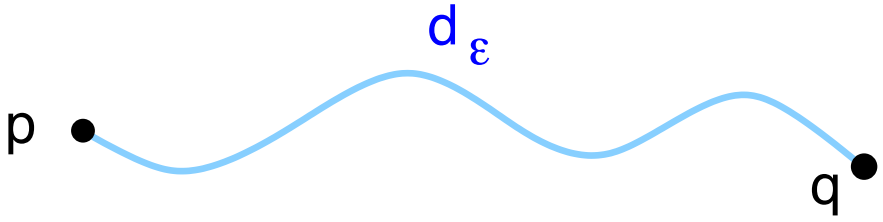




- Find network distance d_ϵ from p to q of approximation ϵ

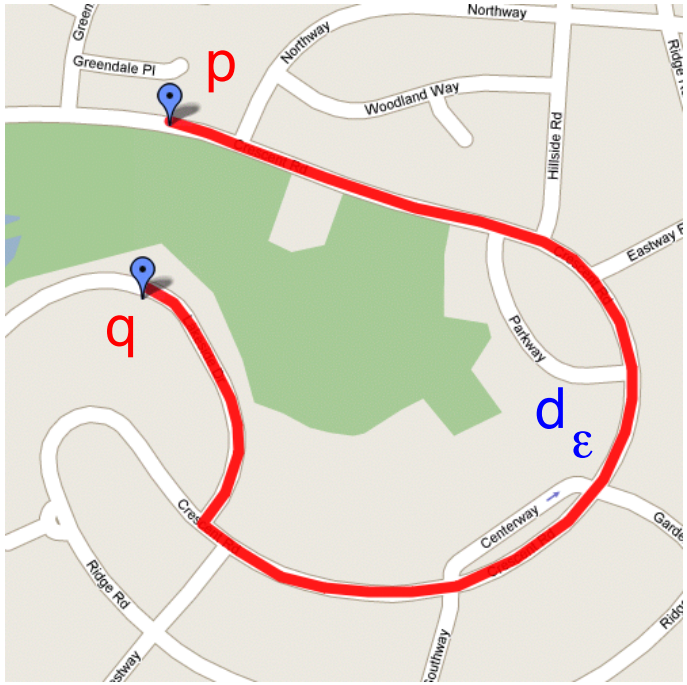
Path and Network Distance Query

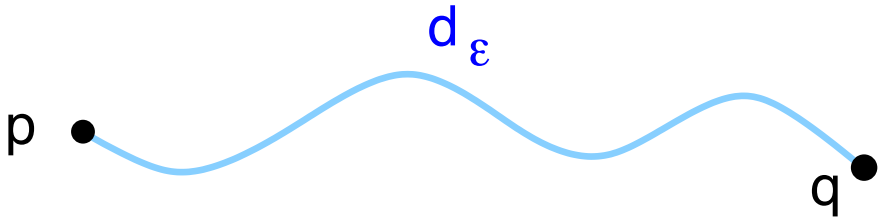




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$

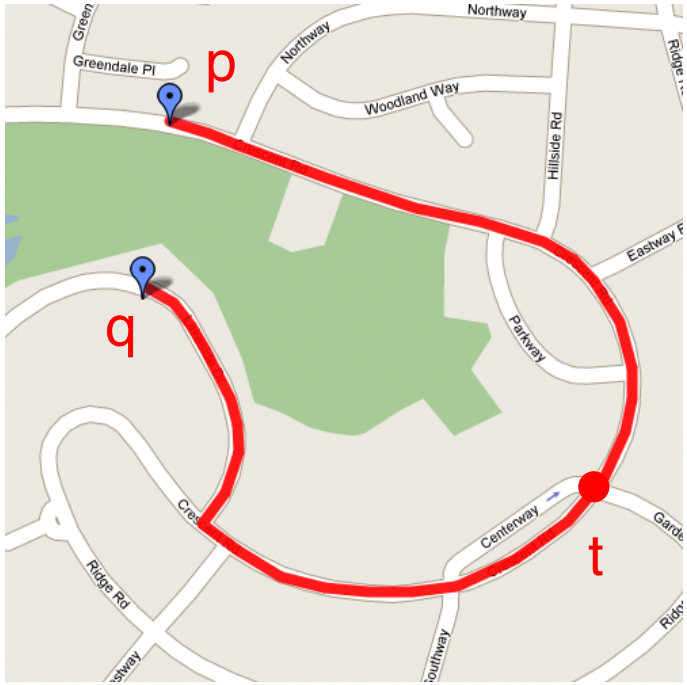
Path and Network Distance Query

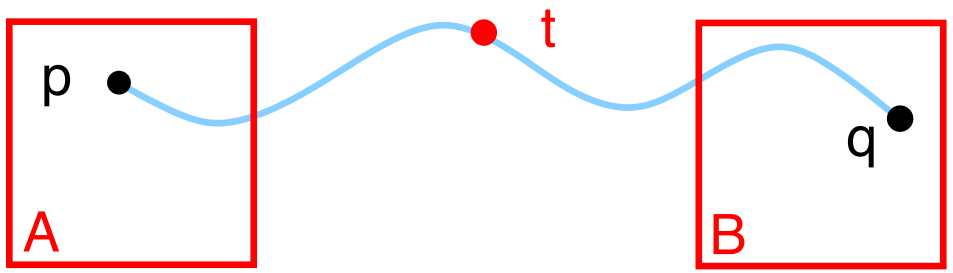




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$
- Find shortest path from p to q

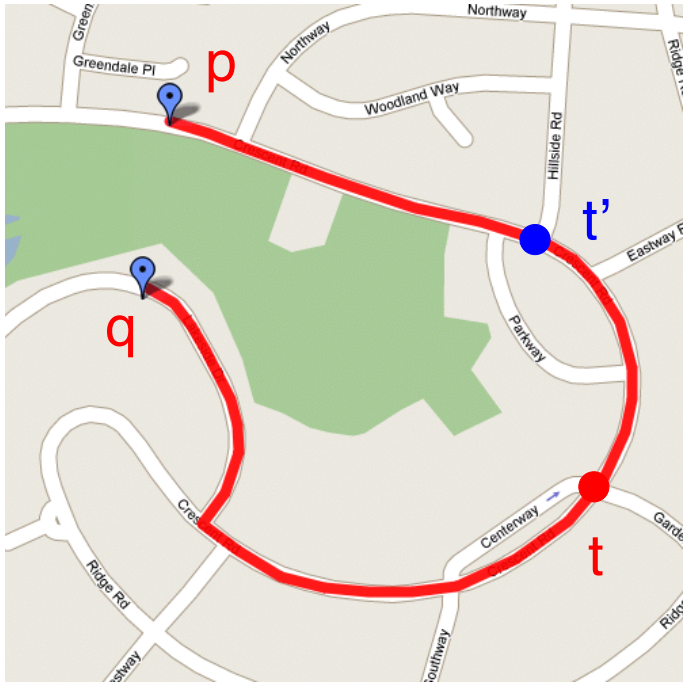
Path and Network Distance Query

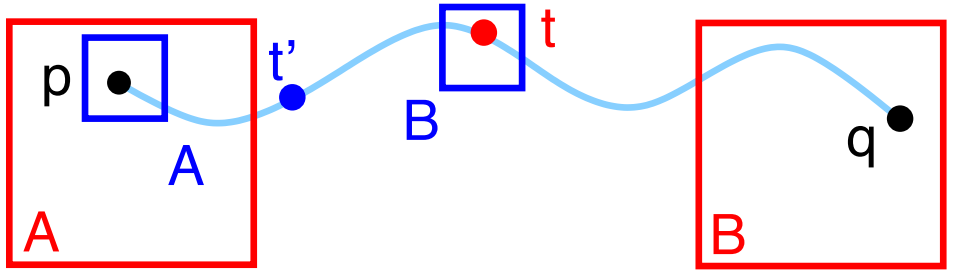




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$
- Find shortest path from p to q
  1. SELECT $O.\Psi$ AS t FROM O WHERE O.AB $= Z_4(p, q)$

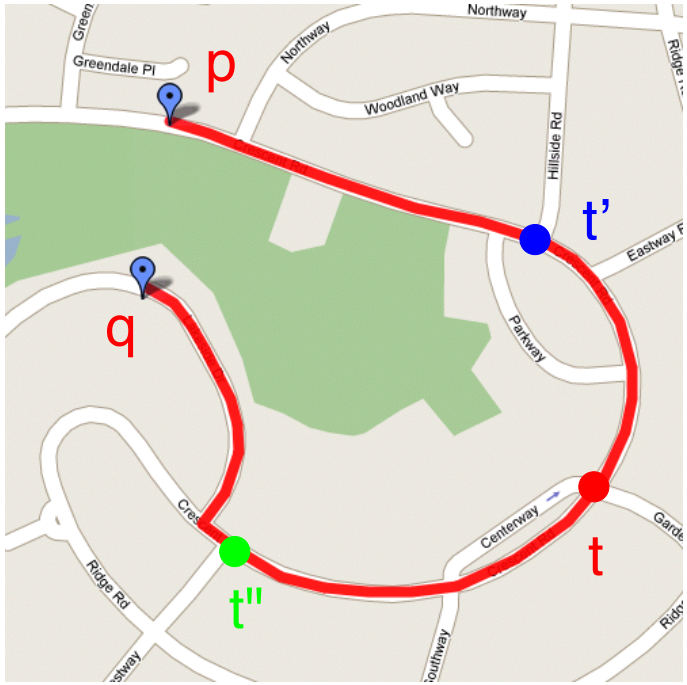
Path and Network Distance Query

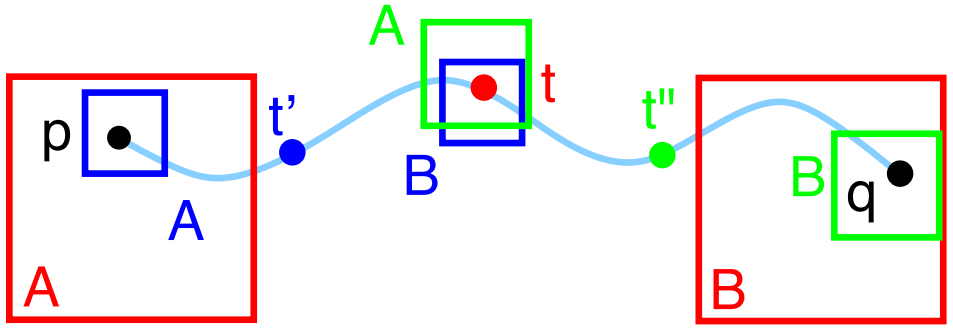




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$
- Find shortest path from p to q
  1. SELECT $O.\Psi$ AS t FROM O WHERE O.AB $= Z_4(p, q)$
  2. SELECT $O.\Psi$ AS t' FROM O WHERE O.AB $= Z_4(p, t)$

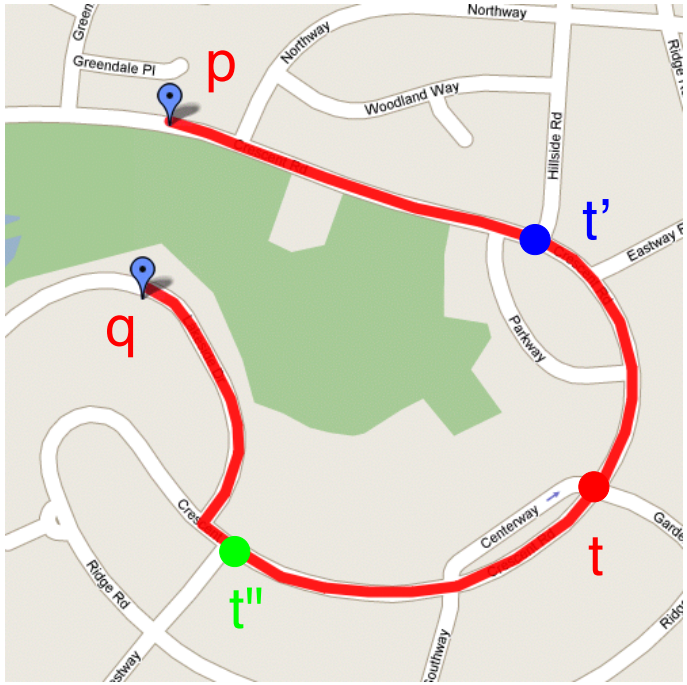
Path and Network Distance Query

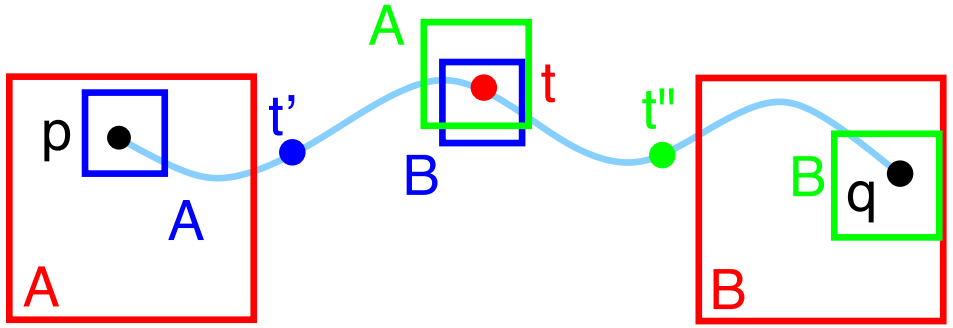




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$
- Find shortest path from p to q
  1. SELECT $O.\Psi$ AS t FROM O WHERE O.AB $= Z_4(p, q)$
  2. SELECT $O.\Psi$ AS t' FROM O WHERE O.AB $= Z_4(p, t)$
  3. SELECT $O.\Psi$ AS t" FROM O WHERE O.AB $= Z_4(t, q)$

Path and Network Distance Query

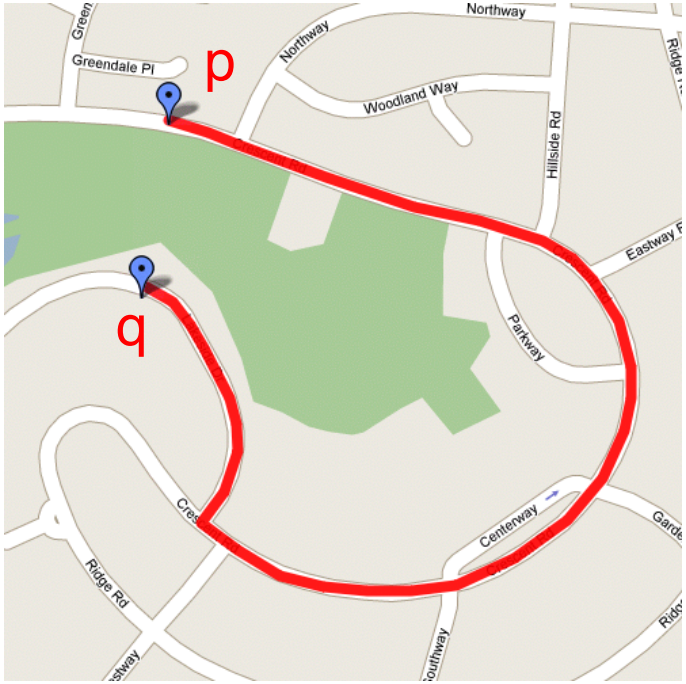




- Find network distance d_ϵ from p to q of approximation ϵ
  - SELECT $O.d_\epsilon$ FROM O WHERE O.AB $= Z_4(p, q)$
- Find shortest path from p to q
  1. SELECT $O.\Psi$ AS t FROM O WHERE O.AB $= Z_4(p, q)$
  2. SELECT $O.\Psi$ AS t' FROM O WHERE O.AB $= Z_4(p, t)$
  3. SELECT $O.\Psi$ AS t" FROM O WHERE O.AB $= Z_4(t, q)$
  4. ...

Network Distance Query

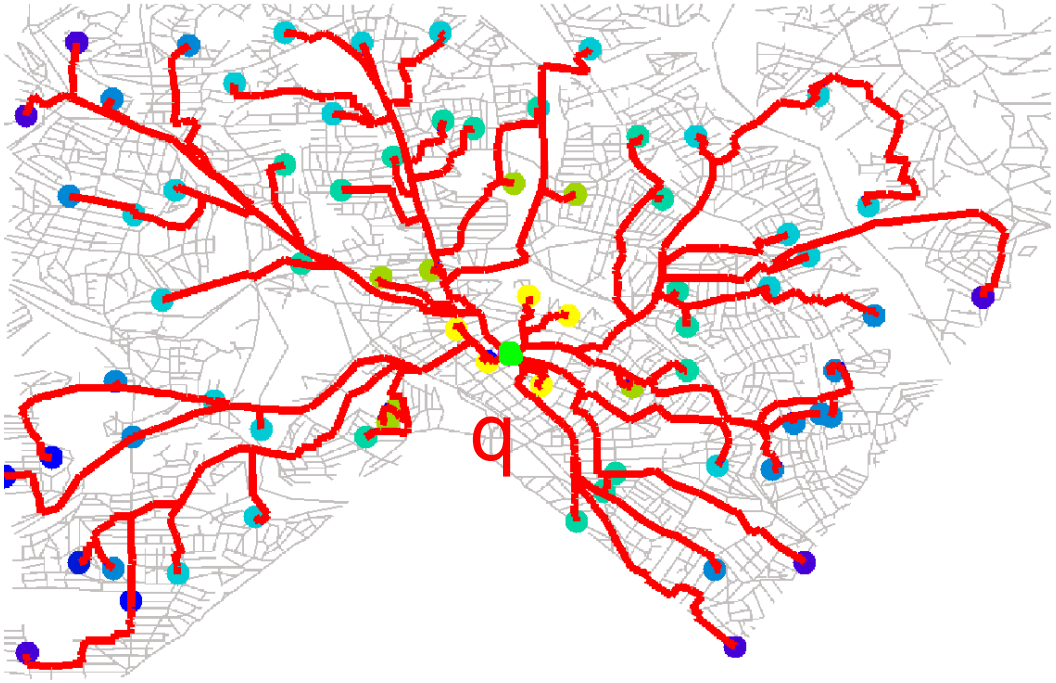
- Find network distance from p to q of approximation ϵ



SELECT $O.d_\epsilon$, $O.\Psi$, $O.\epsilon_L$, $O.\epsilon_H$ FROM O WHERE O.AB $= Z_4(p, q)$

Range Query

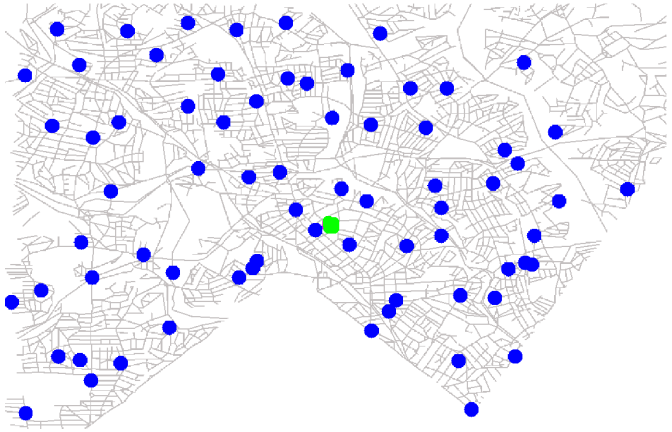
- Find all restaurants in R(pos, type) within 10 miles of q of approximation ϵ



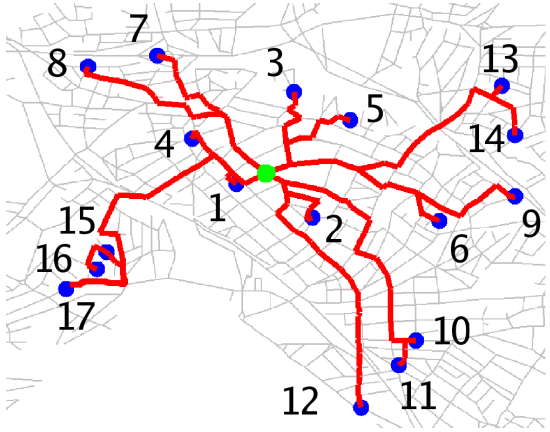
SELECT R.pos, $O.d_\epsilon$ FROM R, O WHERE
O.AB $= Z_4(q, R.\text{pos})$ and $d_\epsilon \leq 10$ MILES

Nearest Neighbor Query

- Find k nearest Italian restaurants in R(pos, type) to q of error ϵ



Query object q and a set of restaurants R



Ordering based on network distance from q

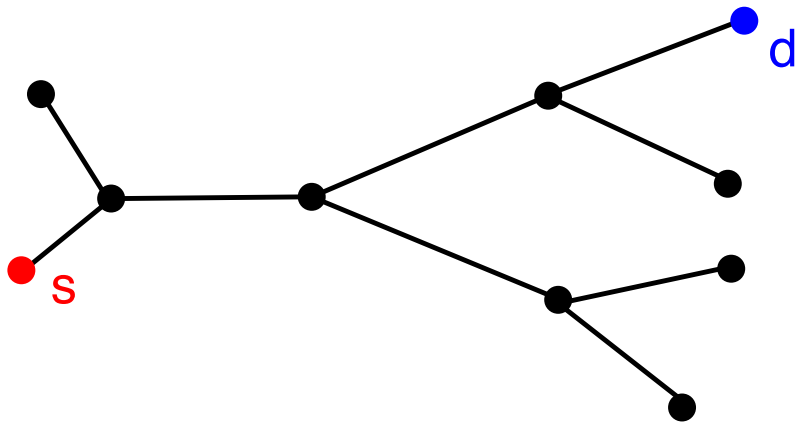
SELECT R.pos, $O.d_\epsilon$ FROM R, O WHERE
O.AB $= Z_4(q, R.\text{pos})$ and R.type = "Italian"
ORDER BY $O.d_\epsilon$ LIMIT k

Foundations of k Nearest Neighbor Finding (kNN)

- A non-incremental best-first algorithm
  - Set of objects (with spatial information)
  - A spatial data structure (e.g., a quadtree or R-tree) on objects
  - Shortest-path quadtrees for the spatial network
  - Note decoupling of data (objects) from domain (spatial network)
  - Cost Justification for Precomputing: Provision to reuse computations across queries and across datasets
- Primitive operations using Progressive Refinement

Foundations of k Nearest Neighbor Finding (kNN)

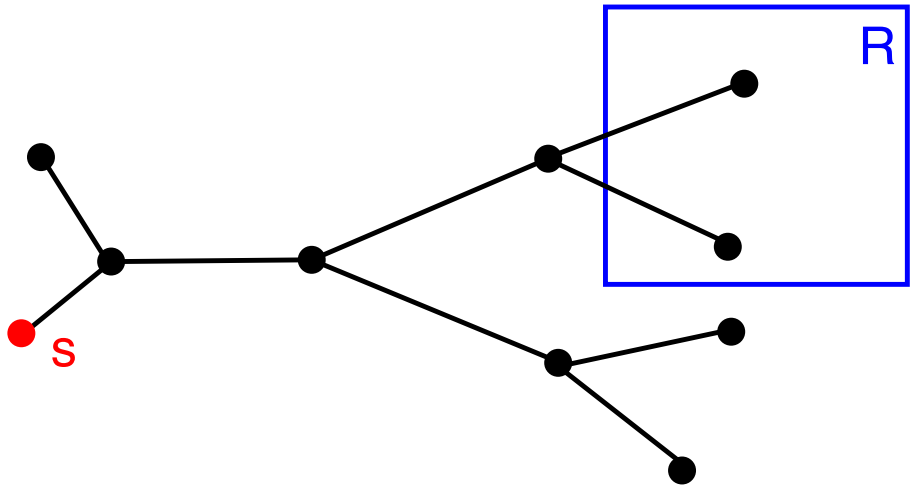
- A non-incremental best-first algorithm
  - Set of objects (with spatial information)
  - A spatial data structure (e.g., a quadtree or R-tree) on objects
  - Shortest-path quadtrees for the spatial network
  - Note decoupling of data (objects) from domain (spatial network)
  - Cost Justification for Precomputing: Provision to reuse computations across queries and across datasets
- Primitive operations using Progressive Refinement



  - DISTANCE_INTERVAL(object,object)

Foundations of k Nearest Neighbor Finding (kNN)

- A non-incremental best-first algorithm
  - Set of objects (with spatial information)
  - A spatial data structure (e.g., a quadtree or R-tree) on objects
  - Shortest-path quadtrees for the spatial network
  - Note decoupling of data (objects) from domain (spatial network)
  - Cost Justification for Precomputing: Provision to reuse computations across queries and across datasets
- Primitive operations using Progressive Refinement



  - DISTANCE_INTERVAL(object,object)
  - DISTANCE_INTERVAL(object,Region)

Properties of kNN Algorithm

- Neighbors produced in increasing order of distance from q
- Use a priority queue Q of objects and blocks
- Q contains network distance interval $[\delta^-, \delta^+]$ of objects from q
- Additional information stored with each object o in Q
  1. An intermediate vertex u in shortest path from q to u
  2. network distance d from q to u
- Uses another priority queue L in addition to Q
  - Stores k objects found so far in increasing order of δ^+
  - D_k is the maximum of the distance interval of the $k\text{th}$ element in L
  - Idea: Prune elements e from Q such that $\delta_e^- \geq D_k$
- Elements are removed from Q in increasing order of the minimum of their distance interval δ^- from q
  - Objects may be reinserted in Q if $\delta^- < D_k$
  - Terminate when $\delta^- \geq D_k$
- Advantages over Incremental best-first kNN (INN)
  - Smaller size of Q
  - Faster than INN

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q
   - A collision occurs if the distance interval of p intersects the distance interval of the current top element in Q

kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q
   - A collision occurs if the distance interval of p intersects the distance interval of the current top element in Q
   - Collision:

kNN Algorithm

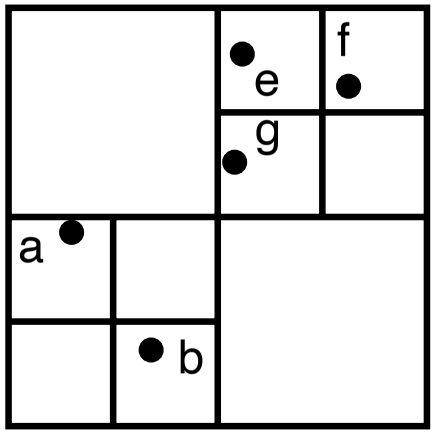
1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q
   - A collision occurs if the distance interval of p intersects the distance interval of the current top element in Q
   - Collision:
     - Remove p from L if $\delta^+ \leq D_k$
     - Apply refinement to improve distance interval of p and reinsert p in L if $\delta^+ \leq D_k$ and in Q if $\delta^- < D_k$ and go to Step 2
   - No collision:

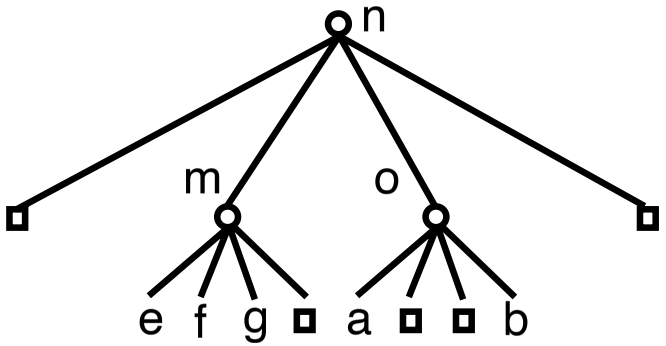
kNN Algorithm

1. Initialize priority queue Q by inserting the quadtree root T of restaurants
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
   - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q
   - A collision occurs if the distance interval of p intersects the distance interval of the current top element in Q
   - Collision:
     - Remove p from L if $\delta^+ \leq D_k$
     - Apply refinement to improve distance interval of p and reinsert p in L if $\delta^+ \leq D_k$ and in Q if $\delta^- < D_k$ and go to Step 2
   - No collision: p is already one of k nearest neighbors in L (Theorem 1) and go to Step 2

Example of an Non-incremental k Neighbor Search

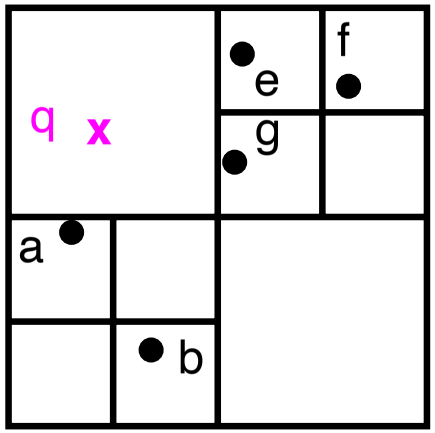
k = 2

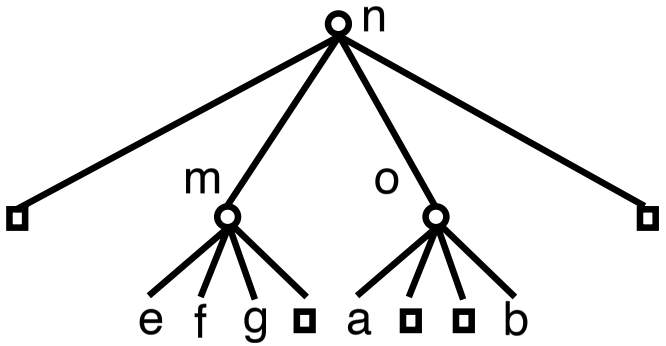




Example of an Non-incremental k Neighbor Search

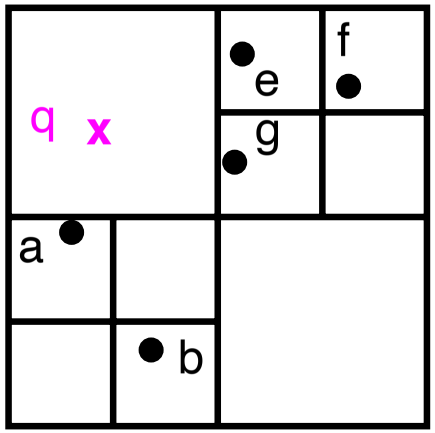
k = 2

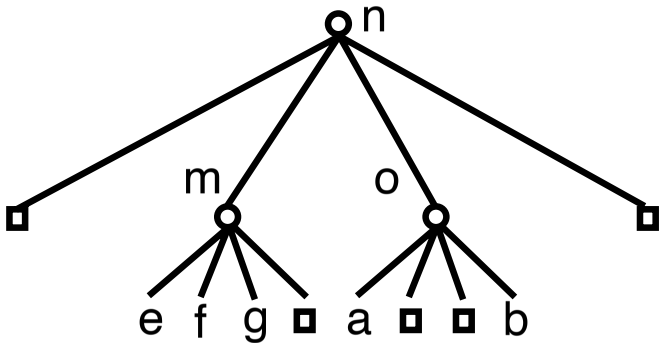




Example of an Non-incremental k Neighbor Search

k = 2



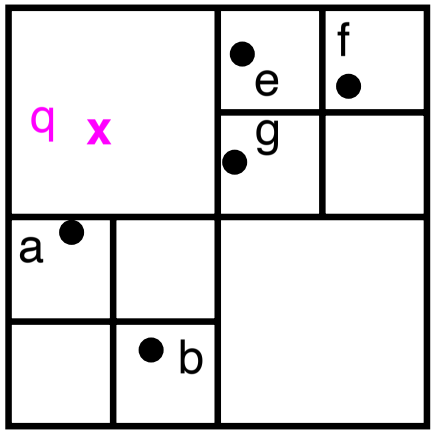


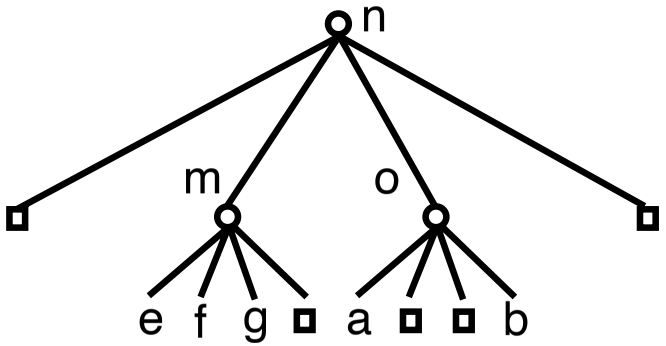
front

L Queue

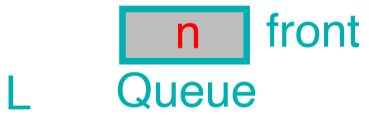
Example of an Non-incremental k Neighbor Search

k = 2



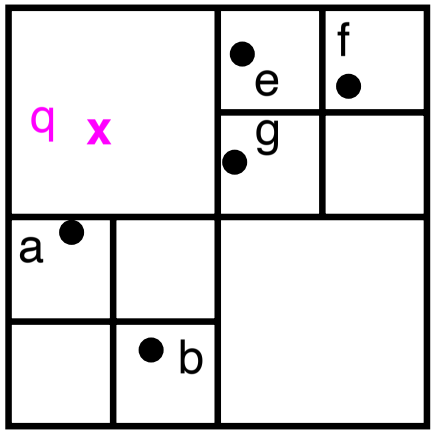


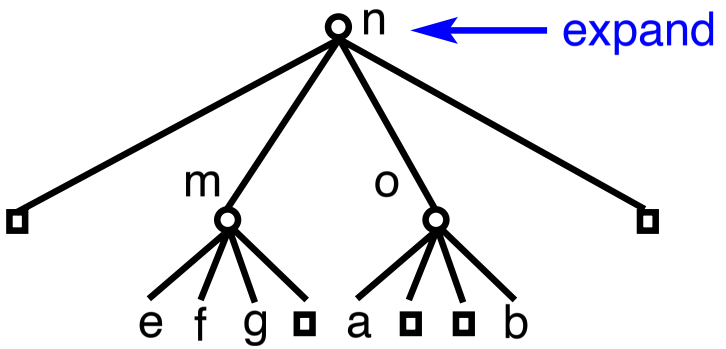
1. Insert n into Queue.



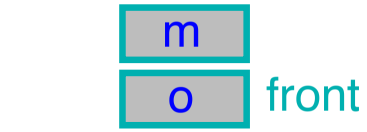
Example of an Non-incremental k Neighbor Search

k = 2





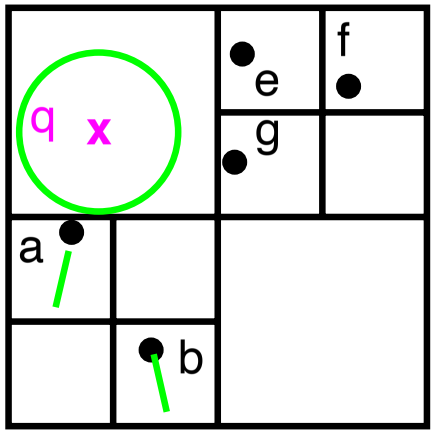
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.

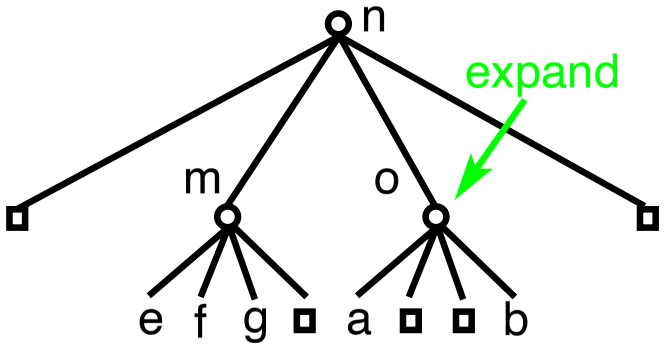


L Queue

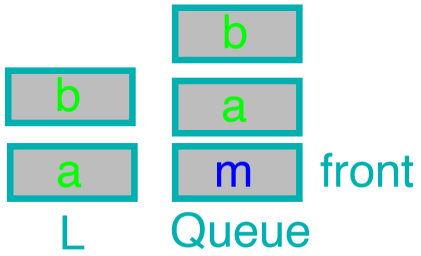
Example of an Non-incremental k Neighbor Search

k = 2

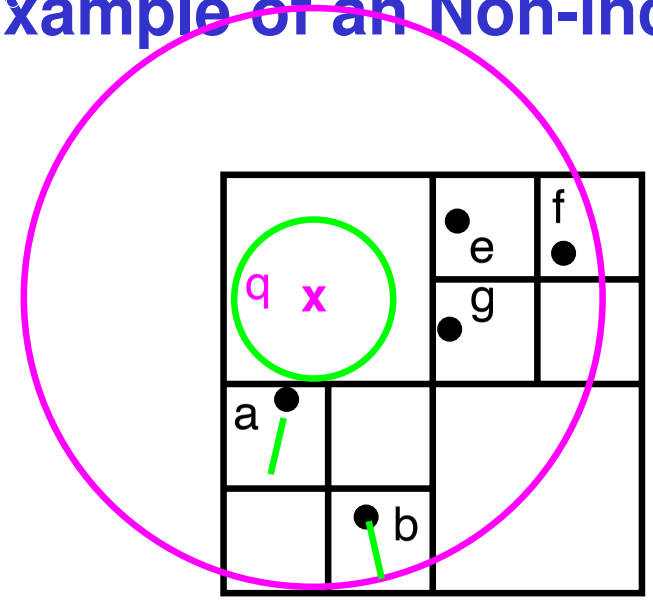




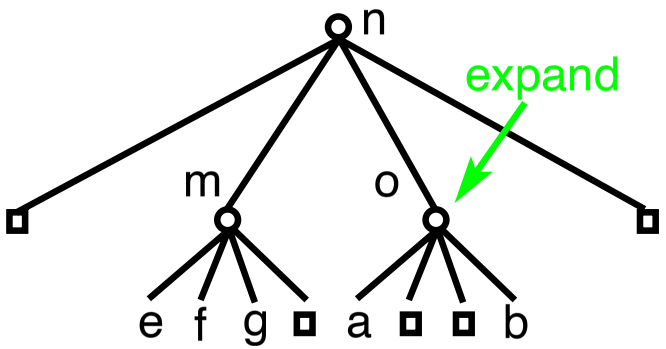
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L.



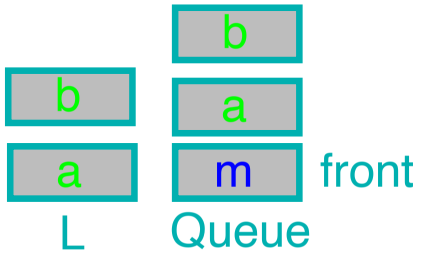
Example of an Non-incremental k Neighbor Search



k = 2

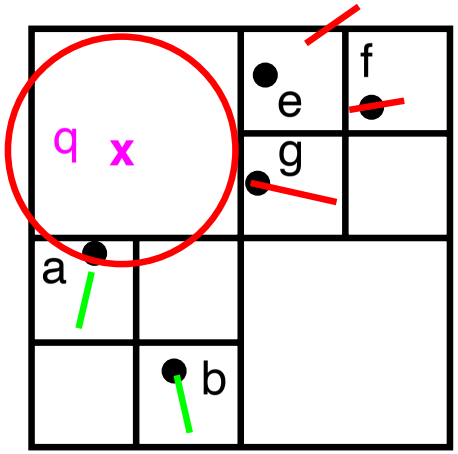


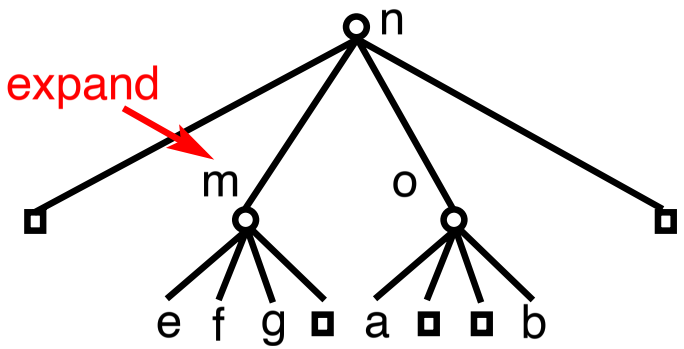
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.



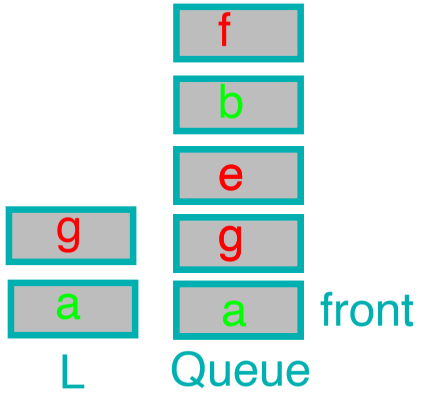
Example of an Non-incremental k Neighbor Search

k = 2

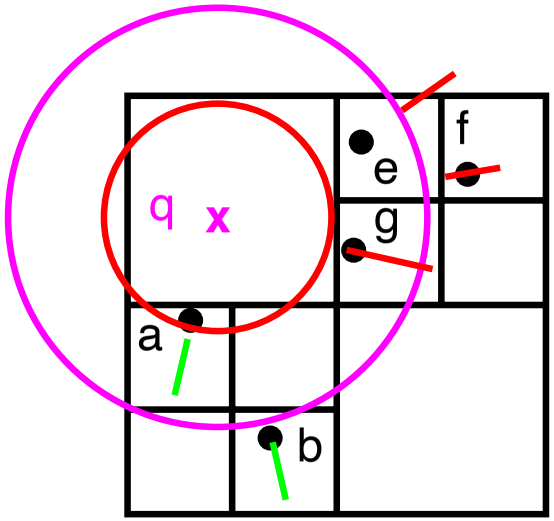


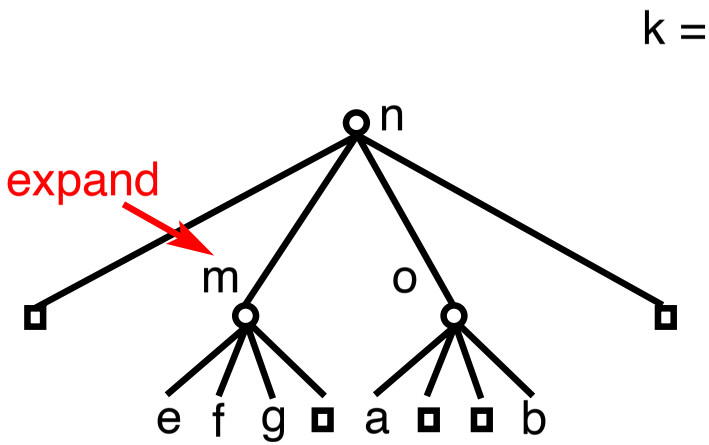


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.

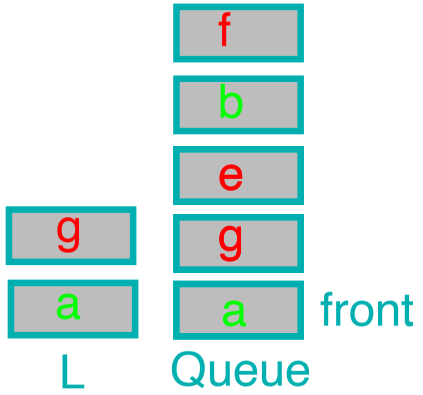


Example of an Non-incremental k Neighbor Search

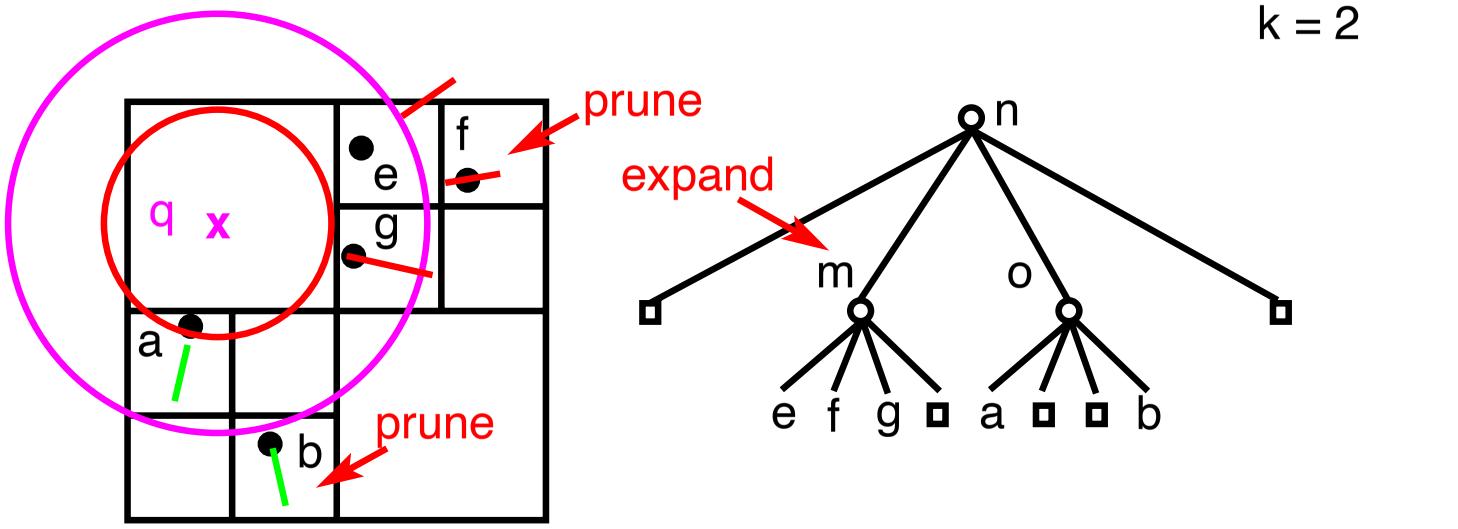




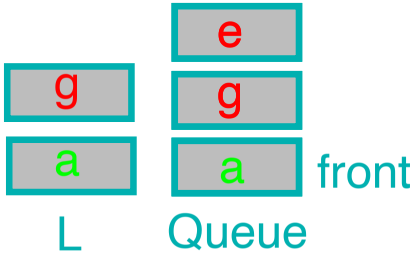
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k.



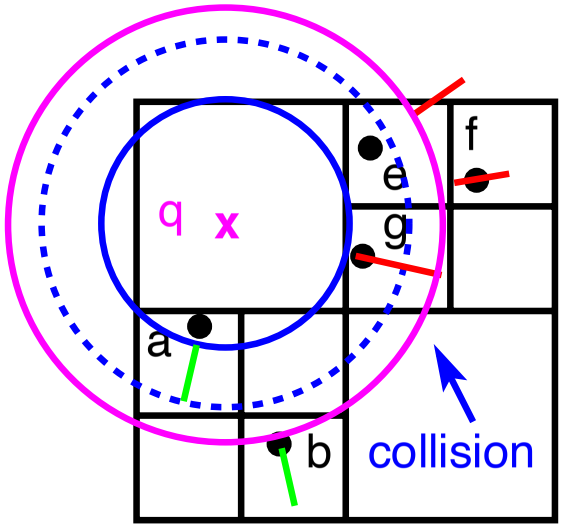
Example of an Non-incremental k Neighbor Search

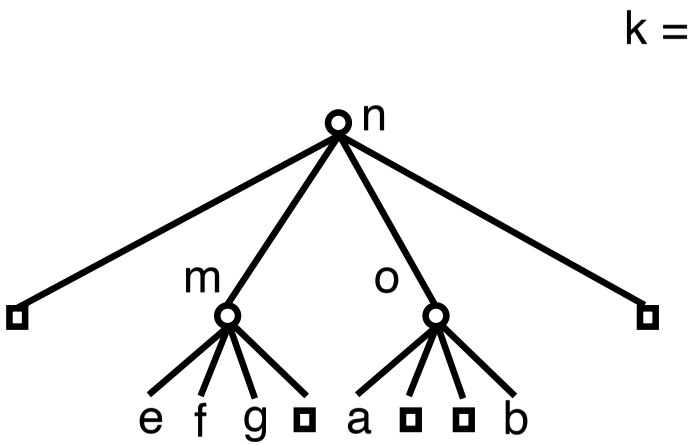


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.

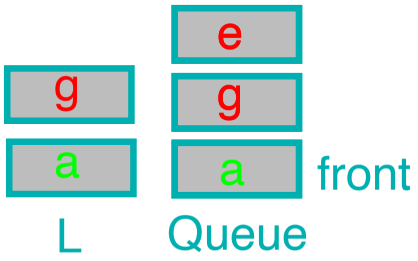


Example of an Non-incremental k Neighbor Search

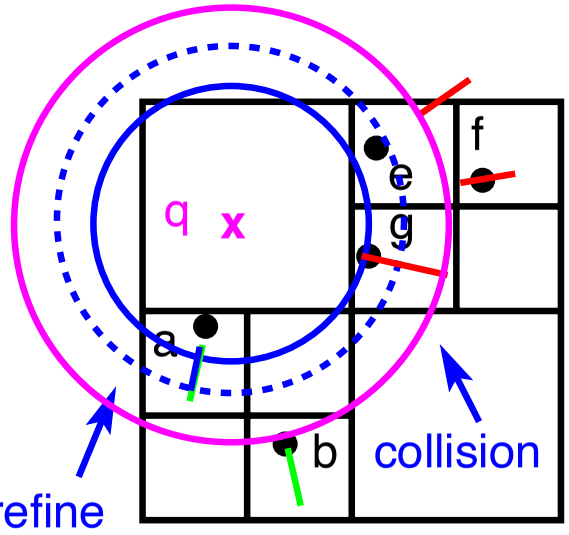


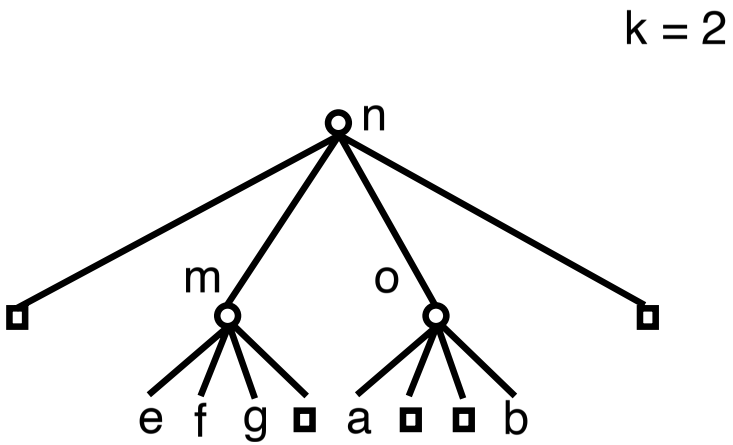


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.

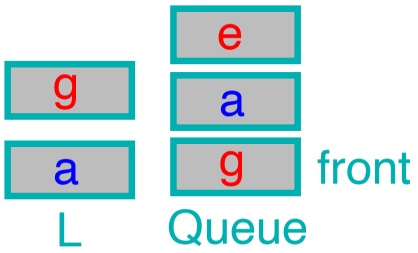


Example of an Non-incremental k Neighbor Search

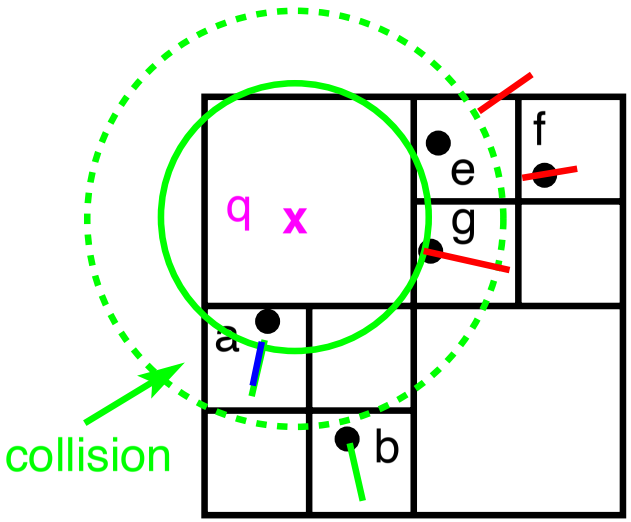


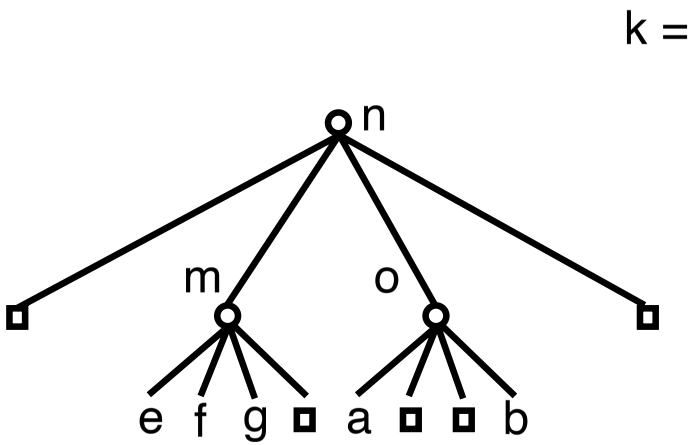


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.

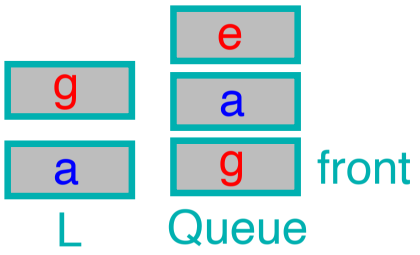


Example of an Non-incremental k Neighbor Search

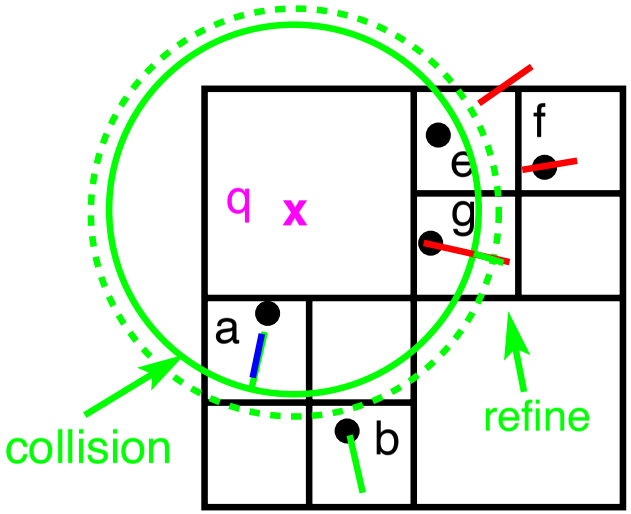


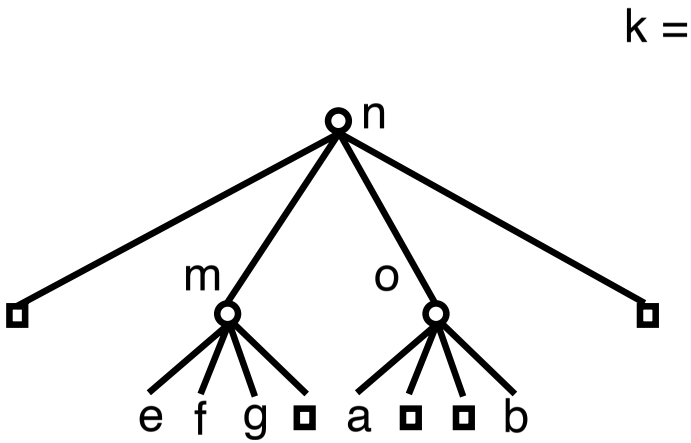


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.

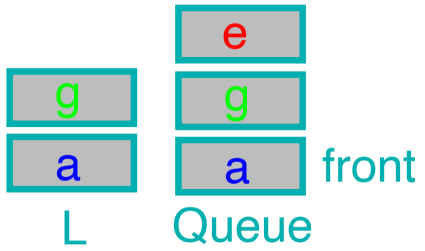


Example of an Non-incremental k Neighbor Search

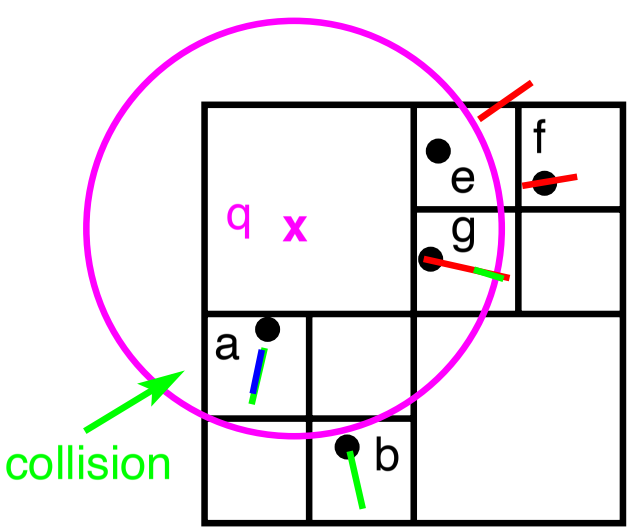


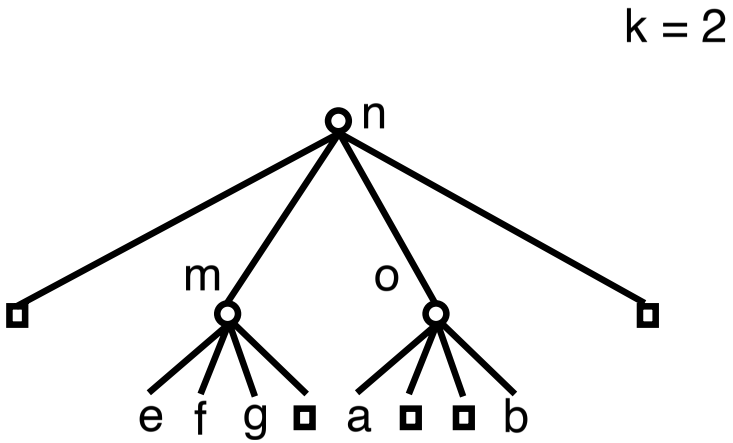


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L.

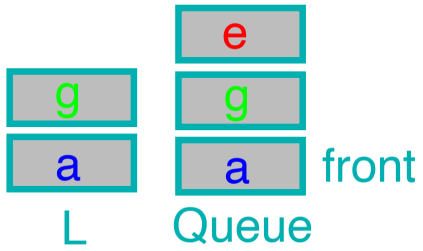


Example of an Non-incremental k Neighbor Search

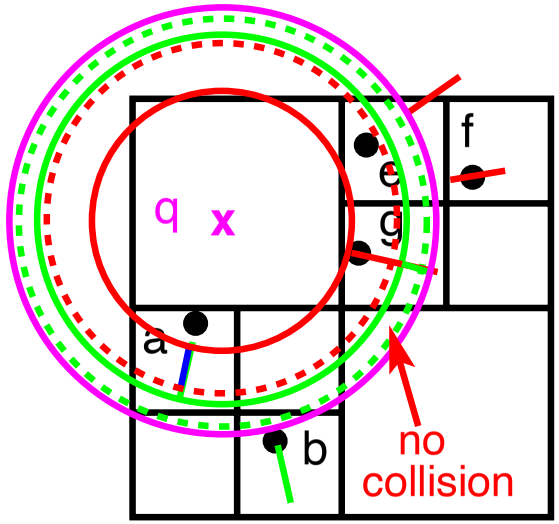


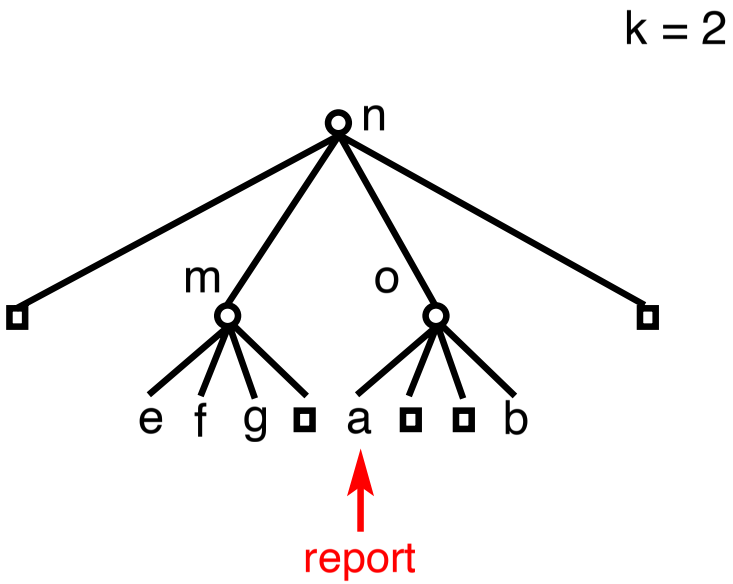


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.

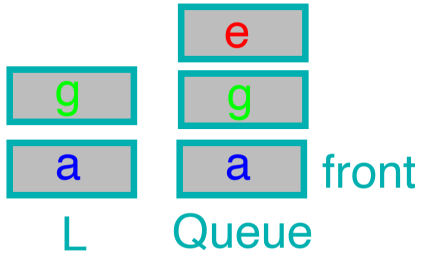


Example of an Non-incremental k Neighbor Search

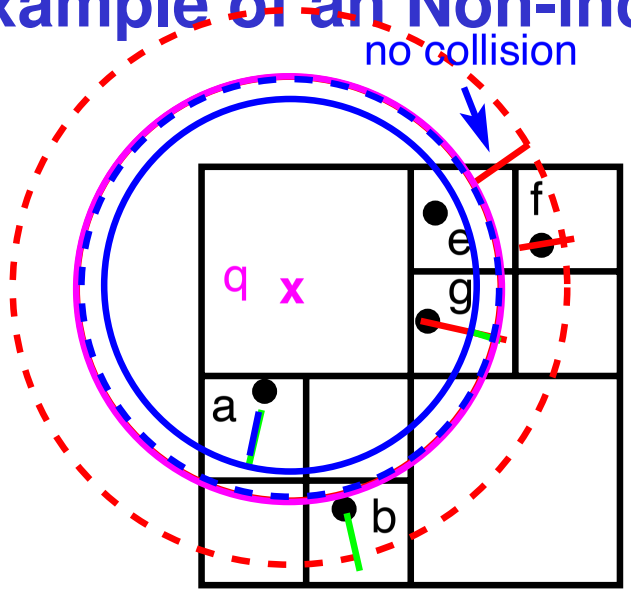


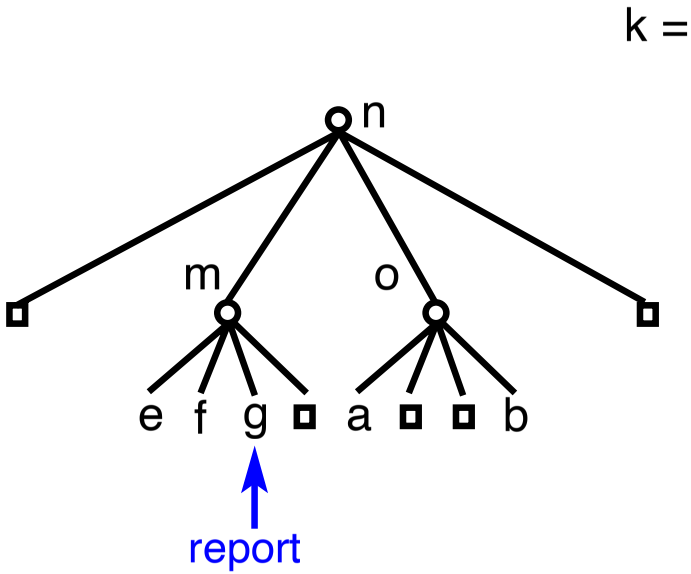


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.
7. Process a. No collision of a with g. No need to refine a further.

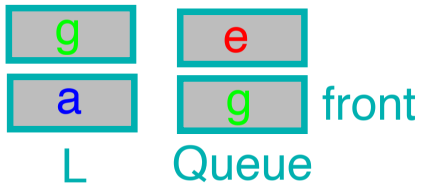


Example of an Non-incremental k Neighbor Search

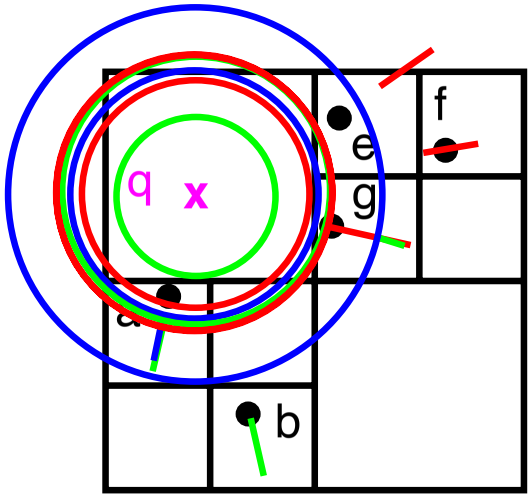


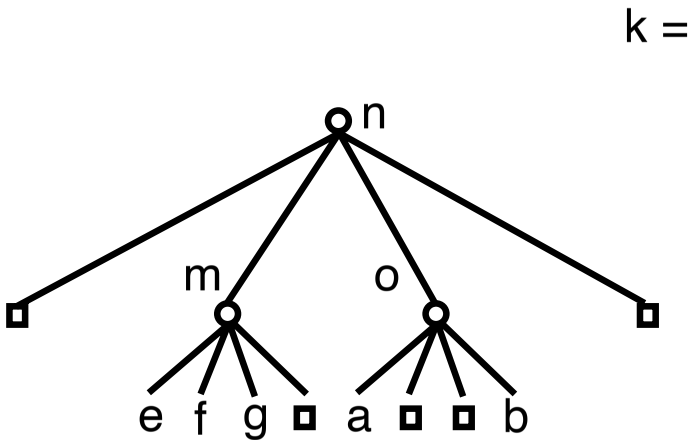


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.
7. Process a. No collision of a with g. No need to refine a further.
8. Process g. No collision of g with e.
   No need to refine g further. Report L.



Example of an Non-incremental k Neighbor Search





1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.
7. Process a. No collision of a with g. No need to refine a further.
8. Process g. No collision of g with e.
   No need to refine g further. Report L.

front

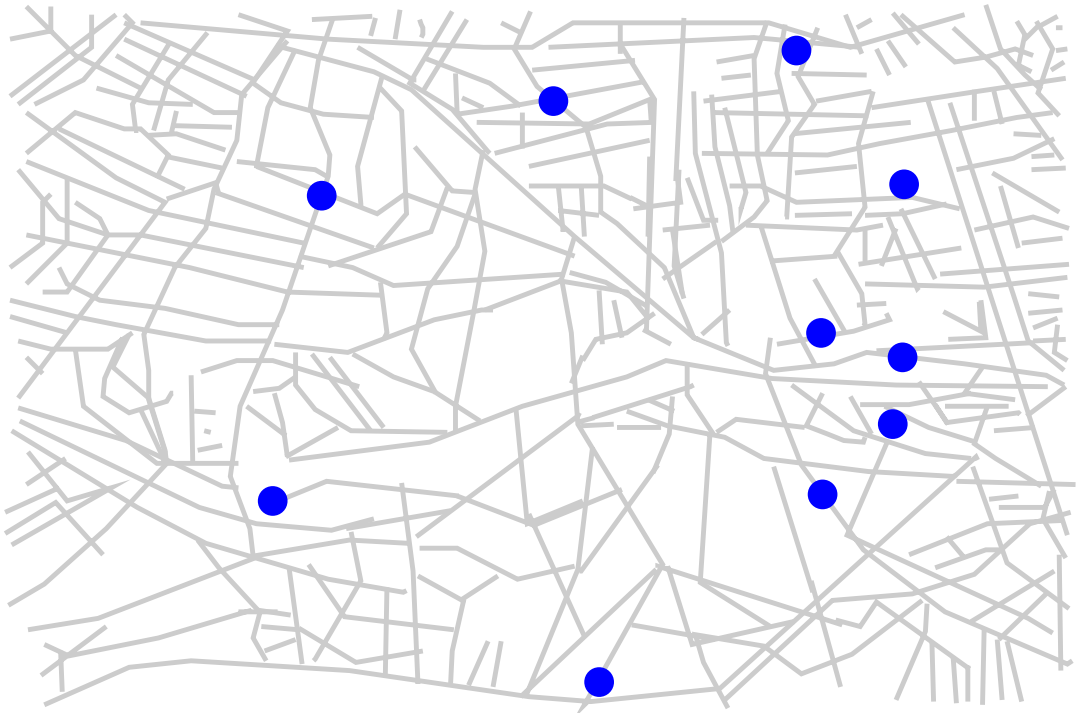
L Queue

Example of a best–first nearest neighbor algorithm.
(Search radius to first element in Queue)

Application in Two Minutes: Dinner and a Movie

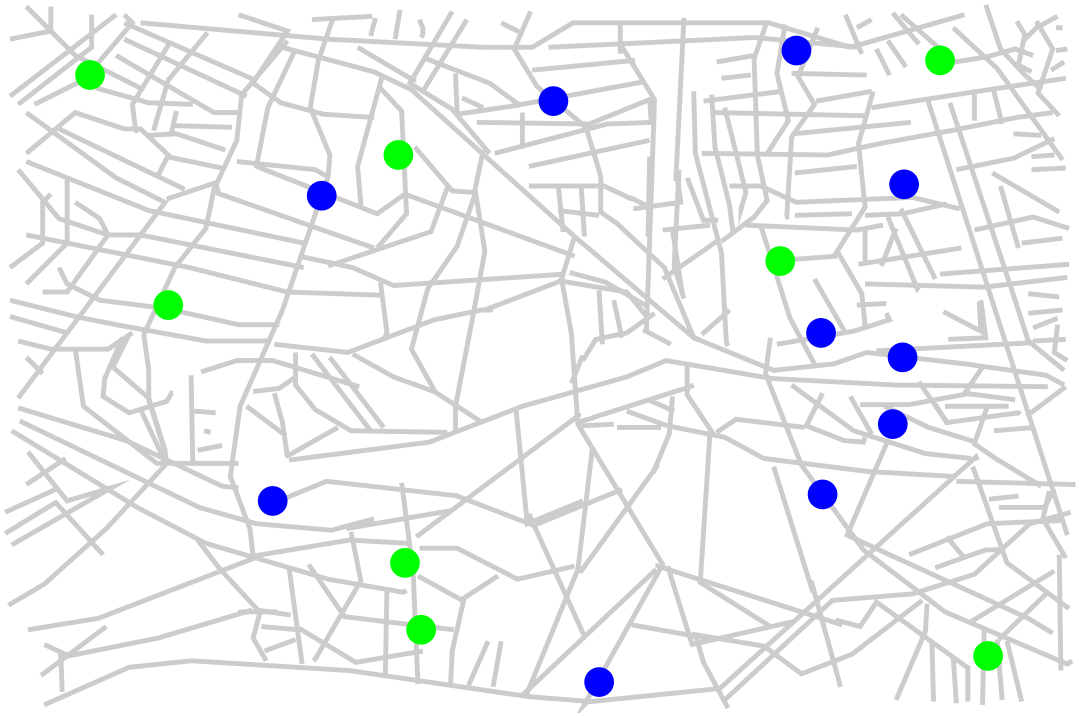
Application in Two Minutes: Dinner and a Movie

Application in Two Minutes: Dinner and a Movie



Restaurants R

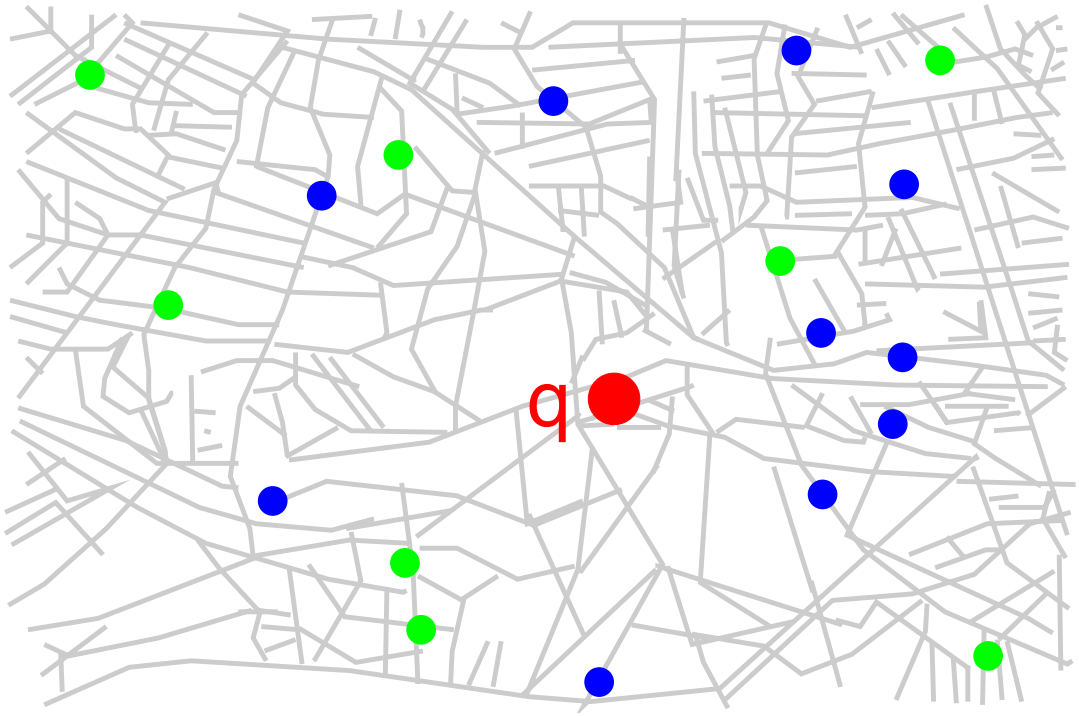
Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Application in Two Minutes: Dinner and a Movie

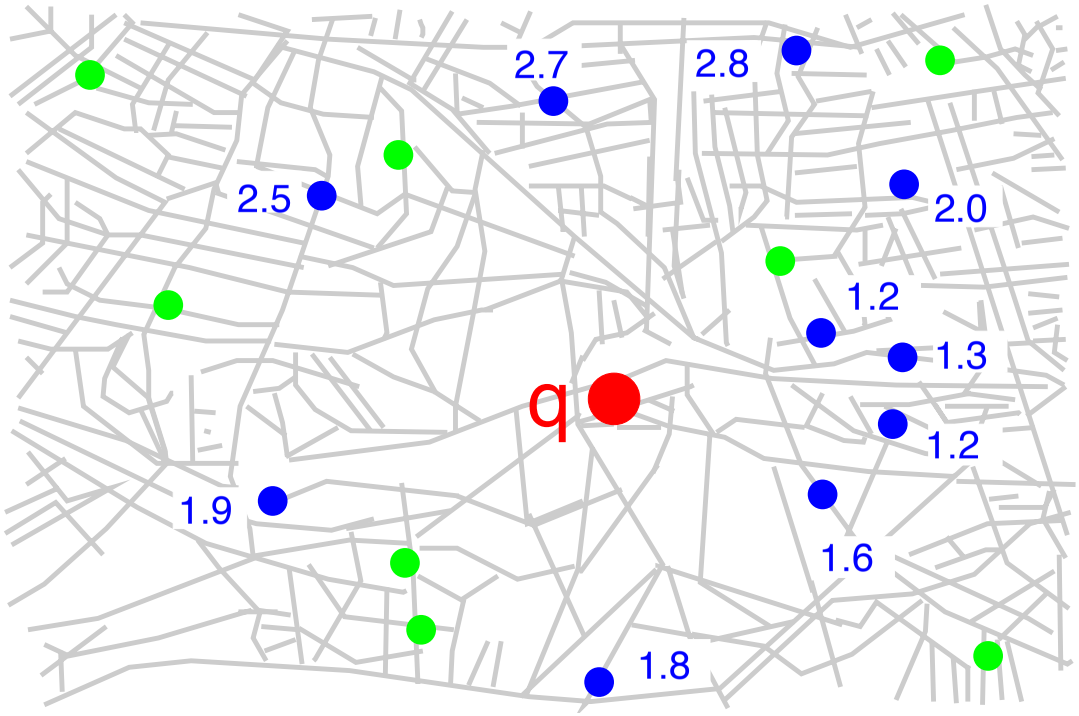


Restaurants R

Theaters T

Query point q

Application in Two Minutes: Dinner and a Movie



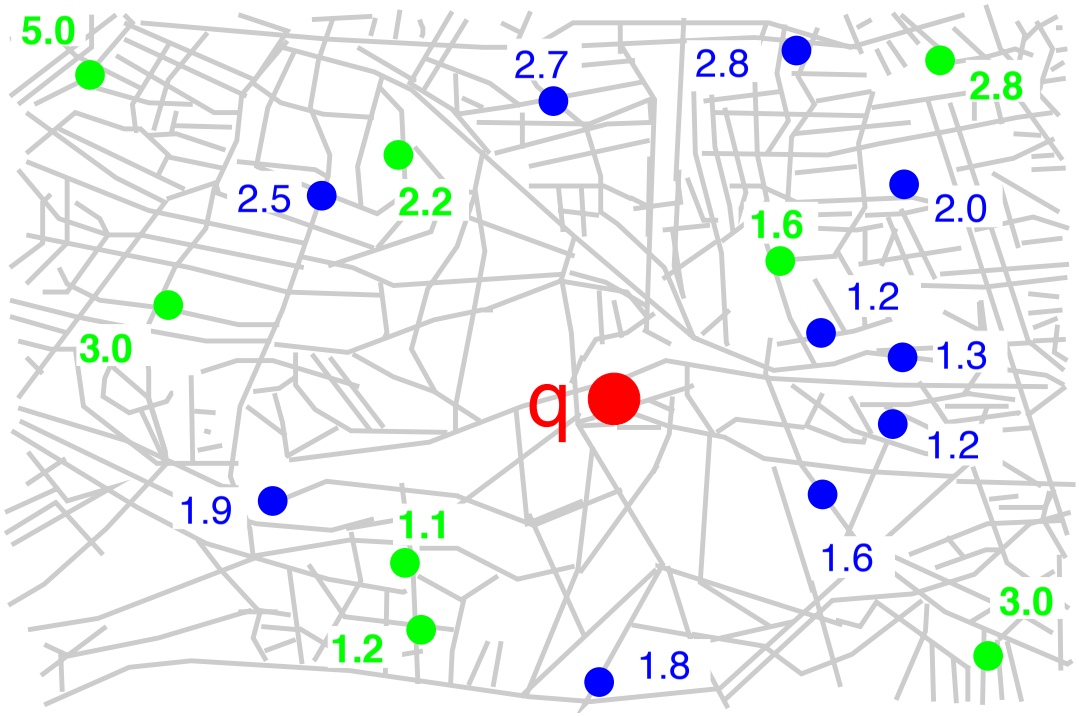
Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

Application in Two Minutes: Dinner and a Movie



Restaurants R

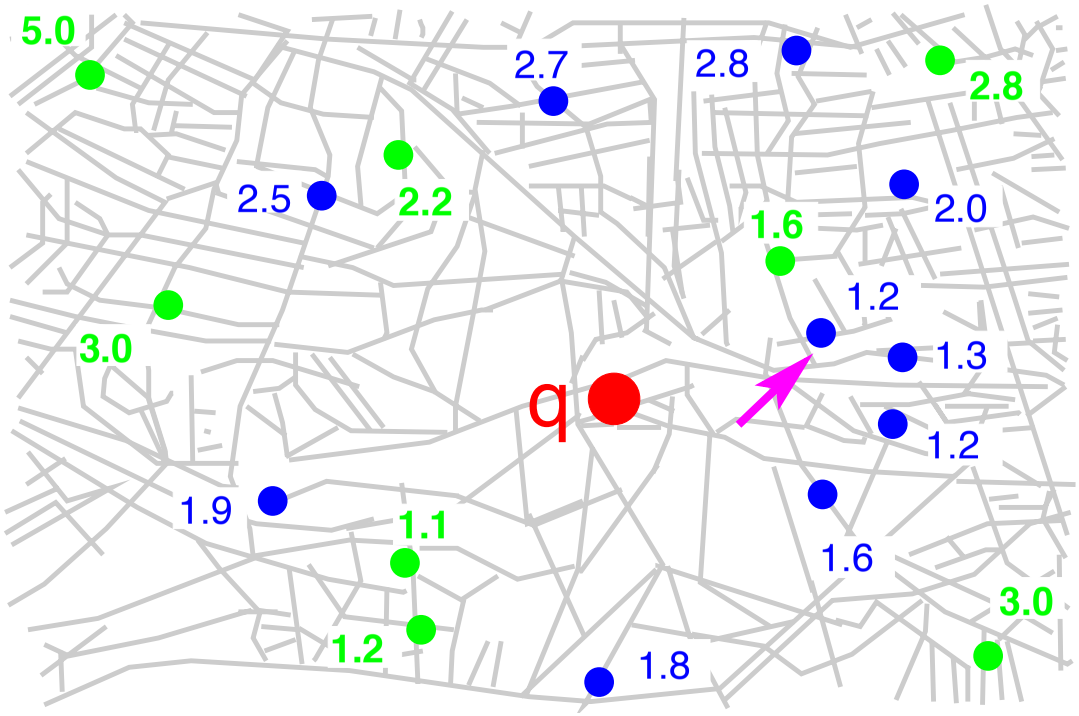
Theaters T

Query point q

Dijkstra's algorithm from q to R

Dijkstra's algorithm from q to T

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

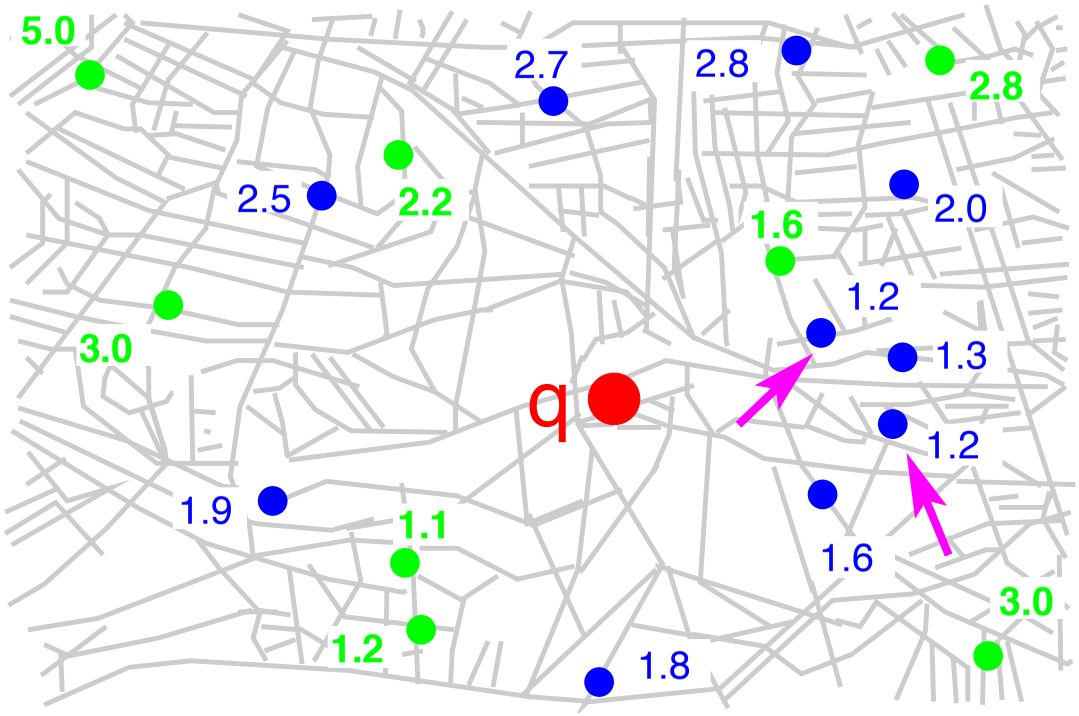
Dijkstra's algorithm from q to T

For each restaurant r in R

  Dijkstra's algorithm from r to T

  Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

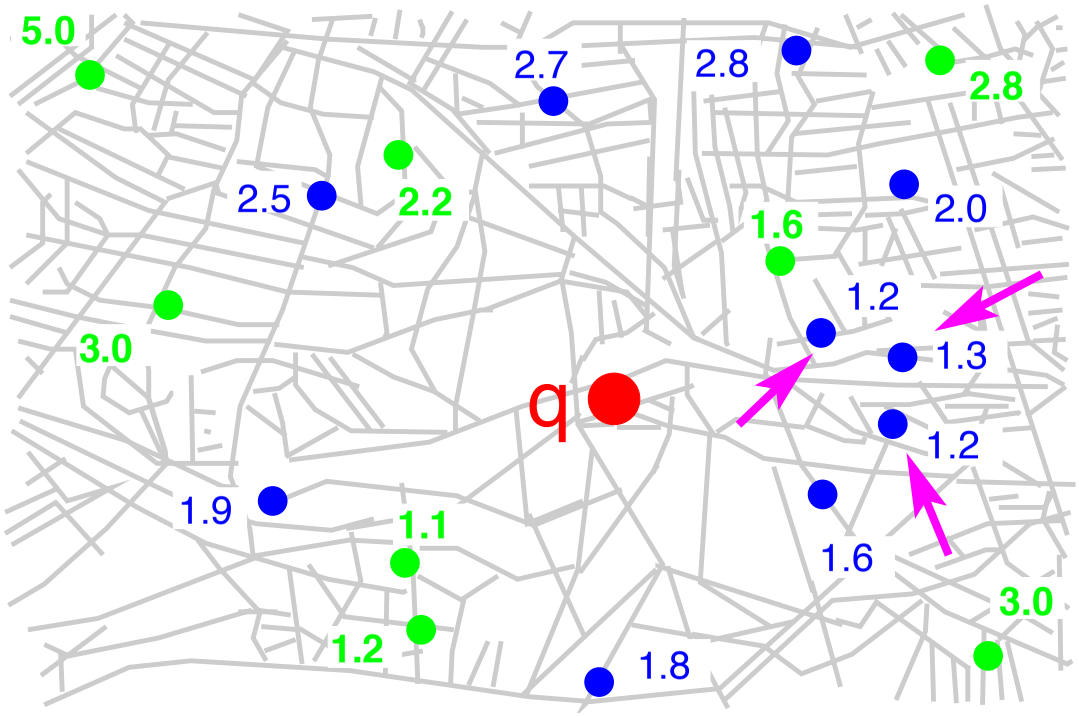
Dijkstra's algorithm from q to T

For each restaurant r in R

  Dijkstra's algorithm from r to T

  Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

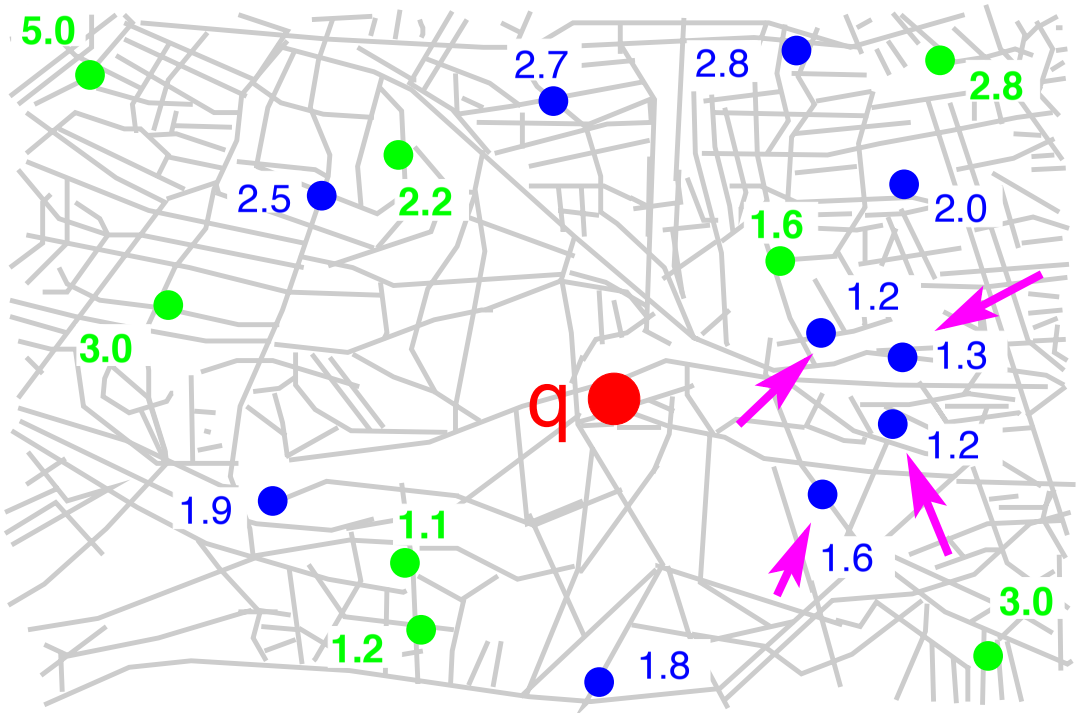
Dijkstra's algorithm from q to T

For each restaurant r in R

  Dijkstra's algorithm from r to T

  Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

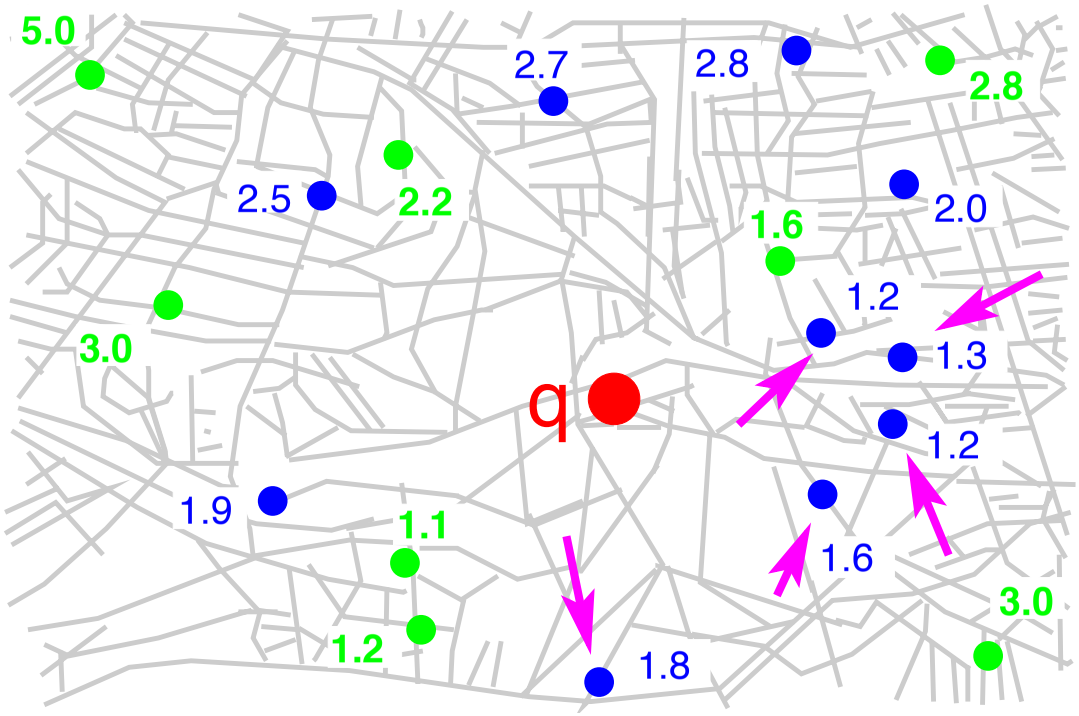
Dijkstra's algorithm from q to R

Dijkstra's algorithm from q to T

For each restaurant r in R

- Dijkstra's algorithm from r to T
- Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

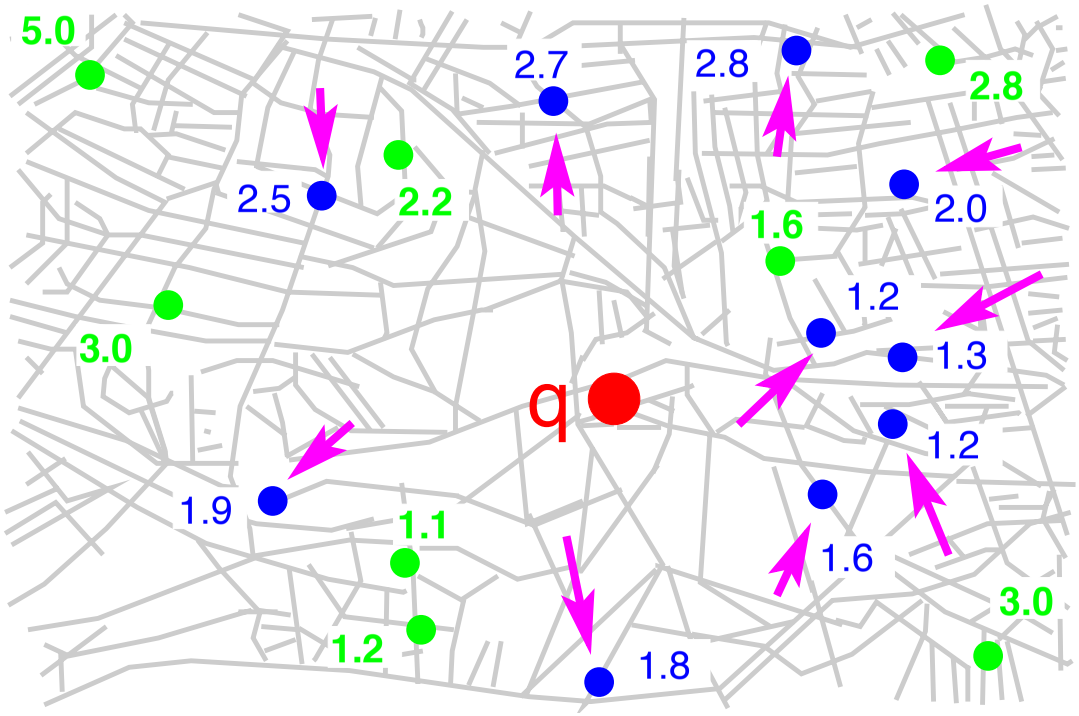
Dijkstra's algorithm from q to R

Dijkstra's algorithm from q to T

For each restaurant r in R

- Dijkstra's algorithm from r to T
- Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

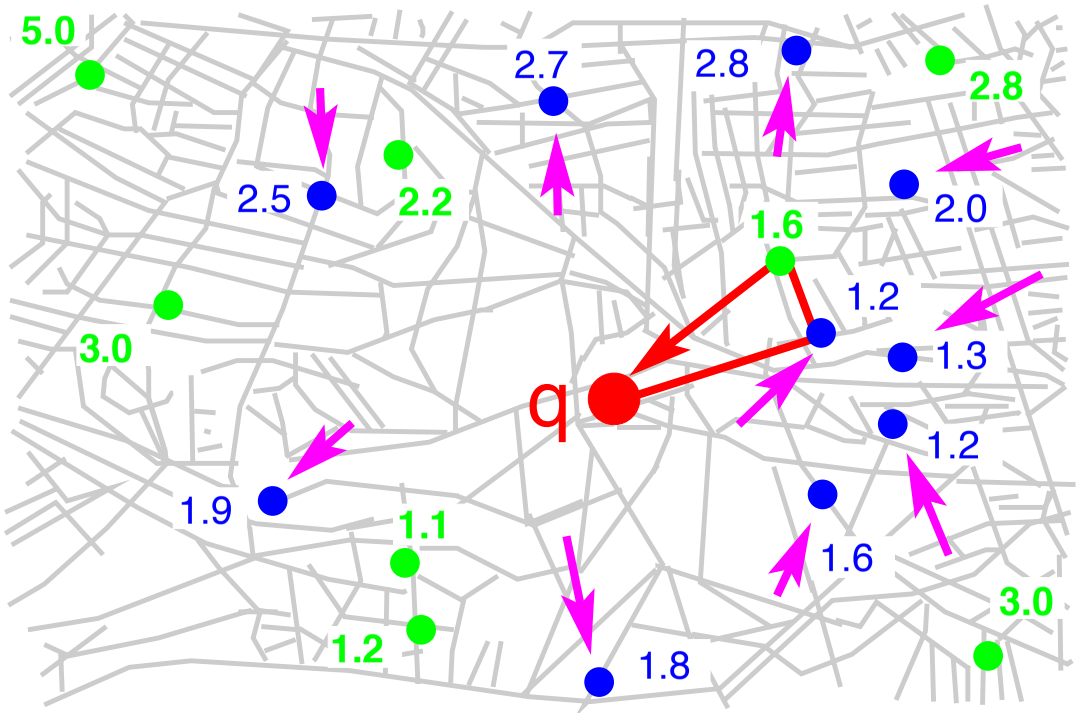
Dijkstra's algorithm from q to T

For each restaurant r in R

  Dijkstra's algorithm from r to T

  Update k best answer buffer

Application in Two Minutes: Dinner and a Movie



Restaurants R

Theaters T

Query point q

Dijkstra's algorithm from q to R

Dijkstra's algorithm from q to T

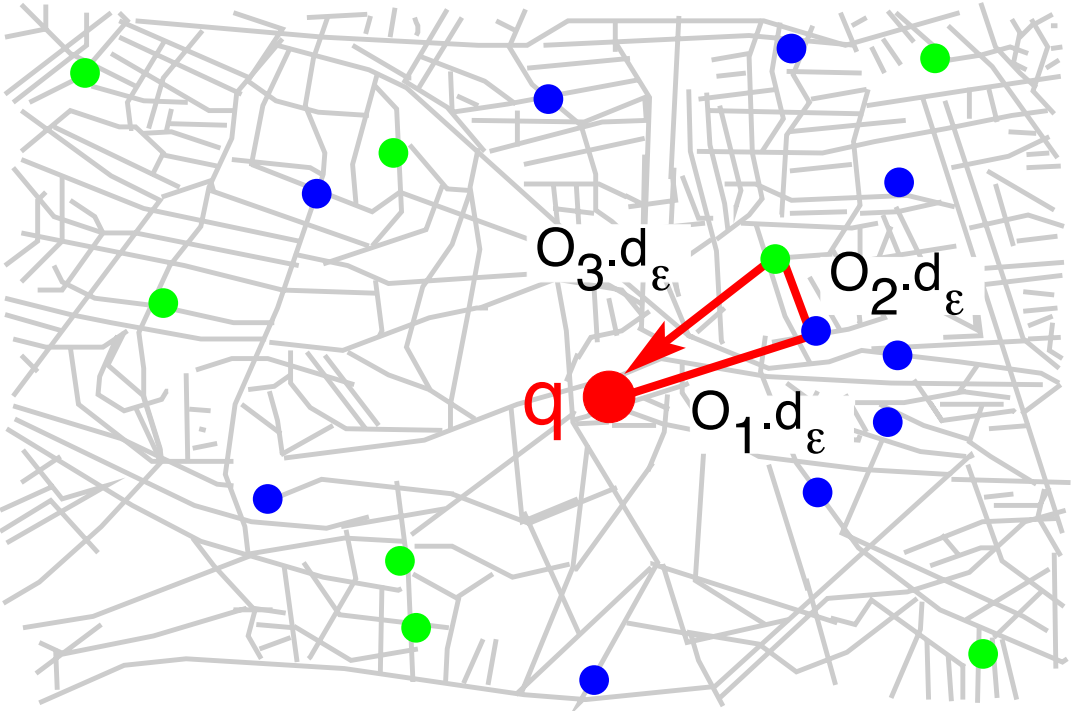
For each restaurant r in R

  Dijkstra's algorithm from r to T

  Update k best answer buffer

Report top k answers

Application in Two Minutes: Dinner and a Movie



Restaurants R
Theaters T
Query point q
Dijkstra's algorithm from q to R
Dijkstra's algorithm from q to T
For each restaurant r in R
  Dijkstra's algorithm from r to T
  Update k best answer buffer
Report top k answers

- In contrast, using an oracle:

SELECT R.pos, T.pos, ($O_1.d_\epsilon$ + $O_2.d_\epsilon$ + $O_3.d_\epsilon$) AS TOTAL
FROM R, T, O AS O_1, O AS O_2, O AS O_3
WHERE O_1.AB = Z_4(q, R.pos) AND
  O_2.AB = Z_4(R.pos, T.pos) AND
  O_3.AB = Z_4(T.pos, q)
ORDER BY TOTAL LIMIT k

Summary

1. Geometric view of spatial networks
2. Observation of path coherence
   - Interplay between spatial and connectivity
   - Transformation of graph-based in to geometric operations
3. Scalable framework for query processing
   - Precomputation shown to be a feasible strategy
   - Oracles are linear in the size of the spatial network
4. Operations expressed using relational operators
   - Efficient query processing in the context of a database

References

1. [Call95] P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 263–272, San Francisco, 1995.
2. [Filh02] G. G. Filho and H. Samet. A linear iterative approach for hierarchical shortest path finding. Computer Science Department CS-TR-4417, University of Maryland, College Park, MD, Nov. 2002.
3. [Gold05a] A. V. Goldberg and C. Harrelson. Computing the shortest path: A^* search meets graph theory. In *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, Vancouver, BC, Canada, Jan. 2005.
4. [Hjal95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD '95: Proceedings of the 4th International Symposium on Large Spatial Databases*, pages 83–95, Portland, ME, Aug. 1995.
5. [Hjal98] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD '98: Proceedings of the International Conference on Management of Data*, pages 237–248, Seattle, WA, June 1998.

References

6. [Jing98] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation. ons of Multidimensional and Metric Data Structures. Morgan-Kaufmann, San Francisco, CA, 2005.
7. [Papa03] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB'03: Proceedings of the 29th International Conference on Very Large Databases*, pages 802–813, Berlin, Germany, Sept. 2003.
8. [Same05] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, CA, 2005.
9. [Shah03] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica*, 7(3):255–273, Sept. 2003.
10. [Shin00] H. Shin, B. Moon, and S. Lee. Adaptive multi-stage distance join processing. In *SIGMOD '00: Proceedings of the International Conference on Management of Data*, pages 343–354, Dallas, TX, May 2000.
11. [Wagn03] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *ESA '03: Proceedings of the 11th Annual European Symposium on Algorithms*, pages 776–787, Budapest, Hungary, Sept. 2003.