Hierarchical Graph Traversal for Aggregate k Nearest Neighbors Search in Road Networks

ICAPS 2020 SEMINAR

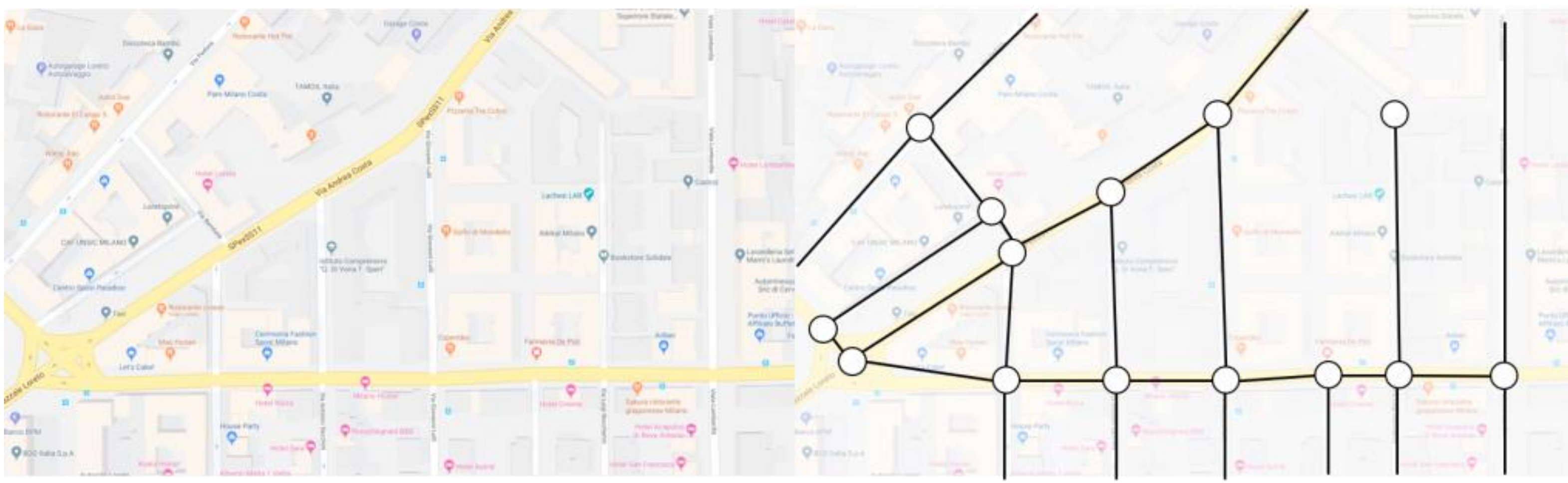
PRESENTER: TENINDRA ABEYWICKRAMA

CO-AUTHORS: MUHAMMAD AAMIR CHEEMA, SABINE STORANDT

Background: Road Network Graph

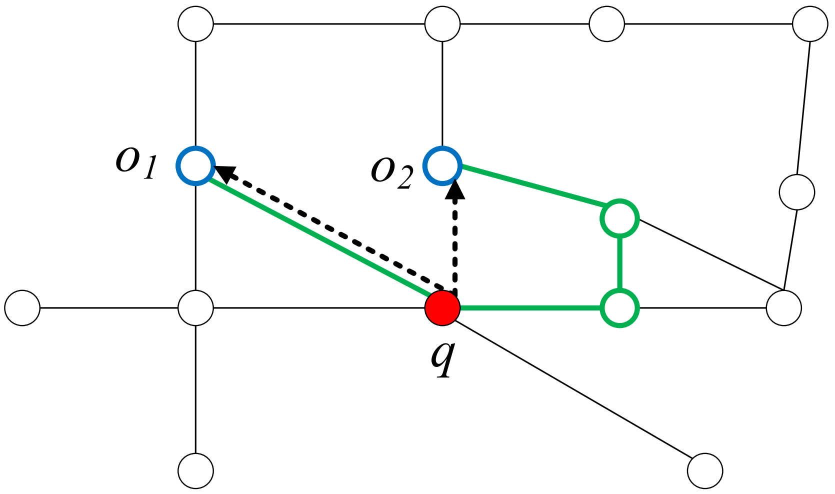
- Input: Road network graph $G = (V, E)$
- Vertex set V: Road intersections
- Edge set E: Road segments
- Each edge has weight: e.g. travel time

Source: https://magazine.impactscool.com/en/speciali/google-maps-e-la-teoria-dei-grafi/



Background: k Nearest Neighbour (kNN) Queries

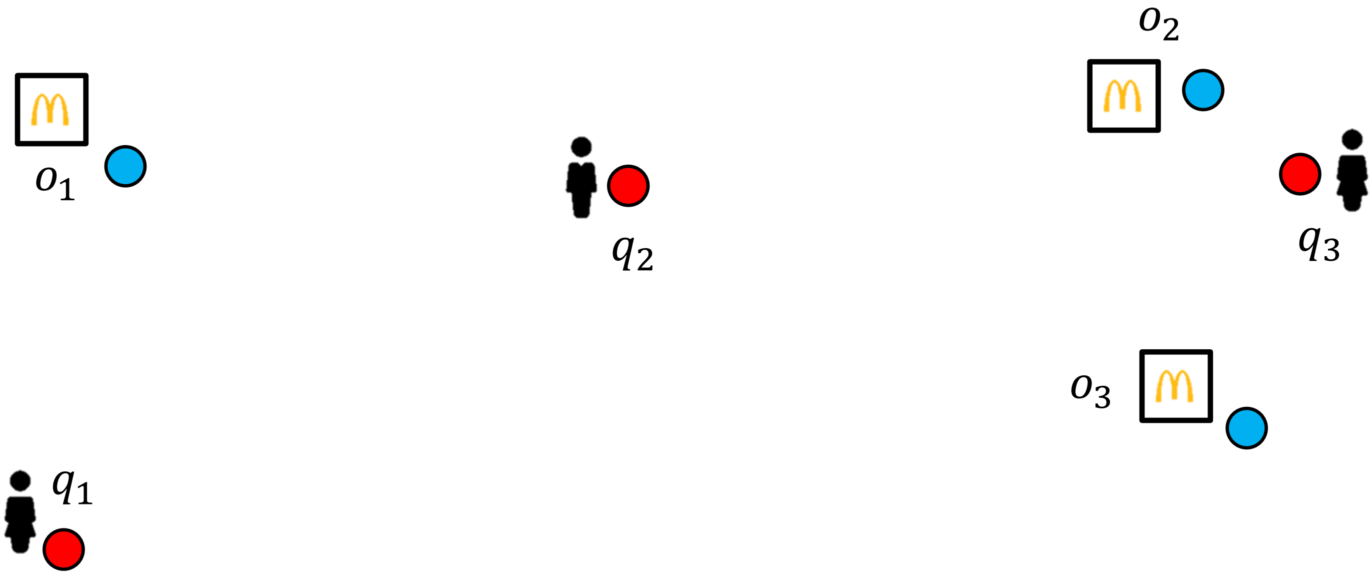
- Input: Object set $O \subseteq V$ (e.g. all restaurants)
- Input: Agent location $q \in V$ (e.g. a diner)
- kNN Query: What is the nearest object to q?
  - *By Euclidean Distance: o_2*
  - *By Network Distance: o_1*
- More accurate + versatile



Our Problem: Aggregate k Nearest Neighbours (AkNN)

- AkNN: Find the nearest object to ***multiple*** agents
  - Example: Three friends (agents) want to meet at a McDonalds (objects). Which object to meet at?

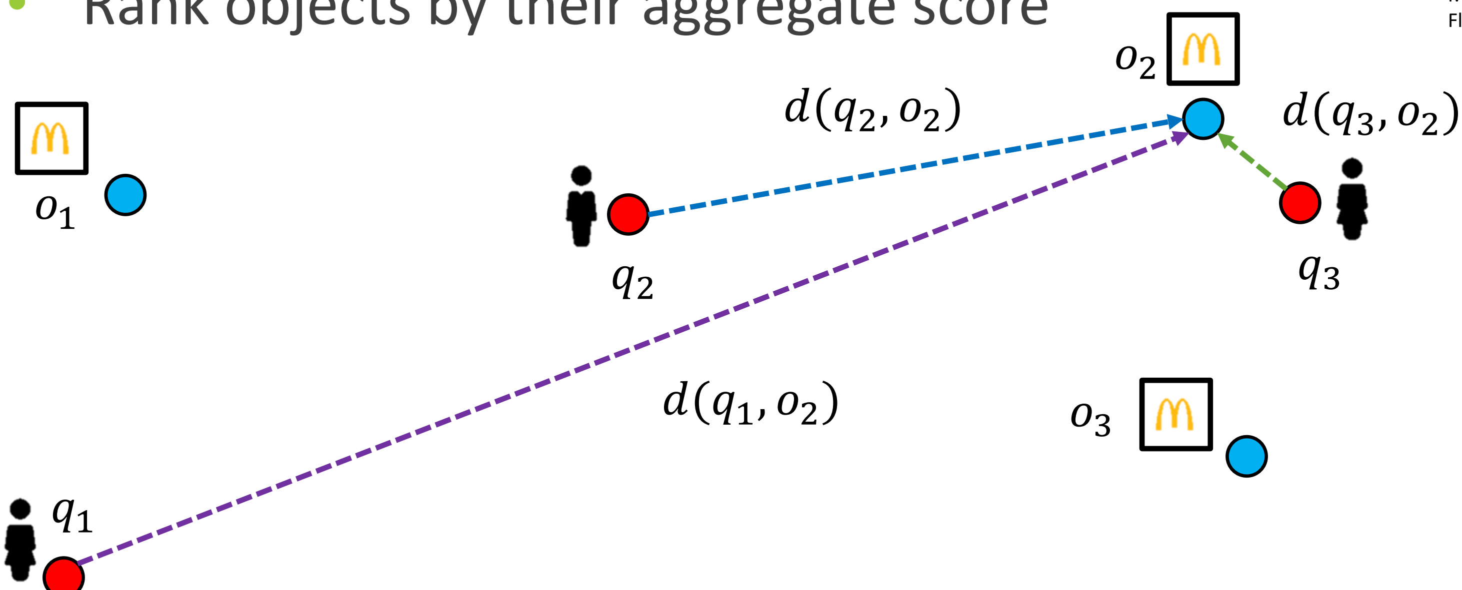
Sources: Google Maps, McDonalds, Flaticon.com



Our Problem: AkNN

- Input: Aggregate Function (e.g. SUM), Agent Set $Q \subseteq V$
- Aggregate individual distances from each agent
- Rank objects by their aggregate score

Sources: Google Maps, McDonalds, Flaticon.com

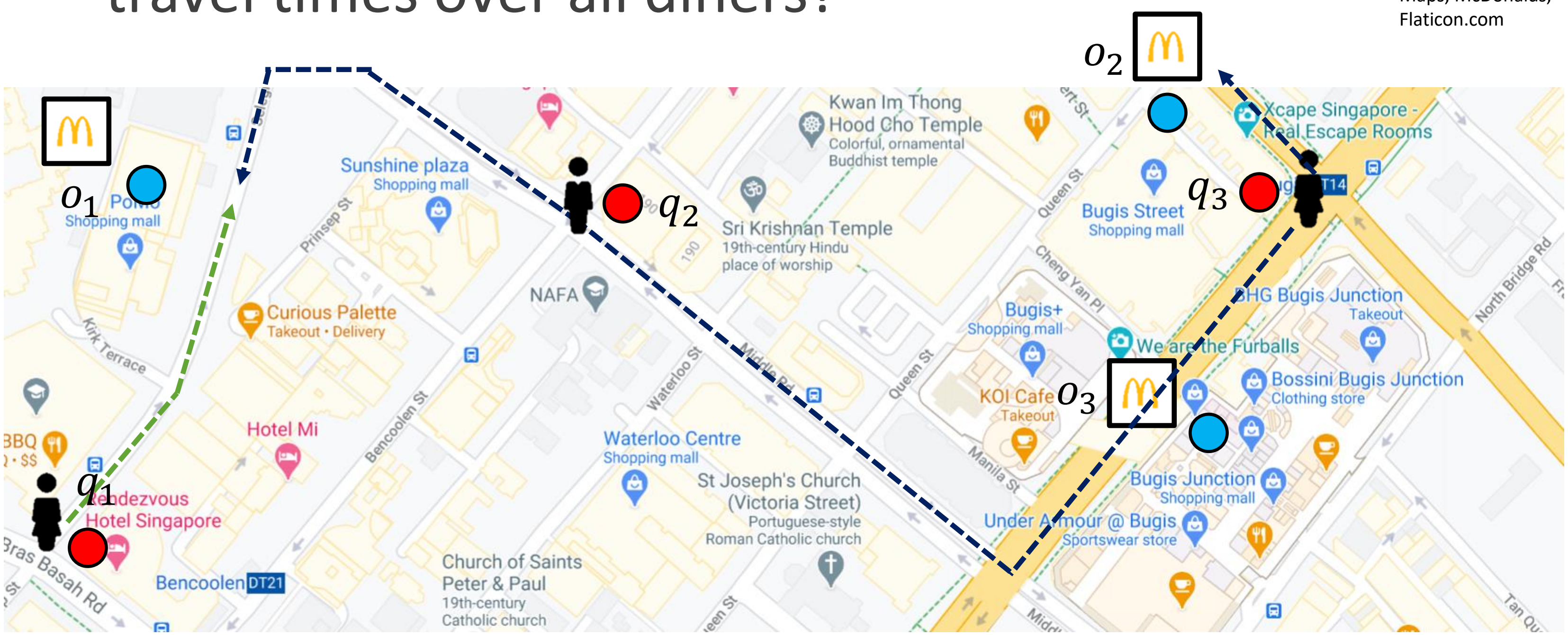


$$Agg\_Score(o_2) = Agg\_Function(d(q_1, o_2), d(q_2, o_2), d(q_3, o_2))$$

Our Problem: AkNN

- Still using network distance for accuracy/versatility
- Example: Which McDonalds minimises the SUM of travel times over all diners?

Sources: Google Maps, McDonalds, Flaticon.com

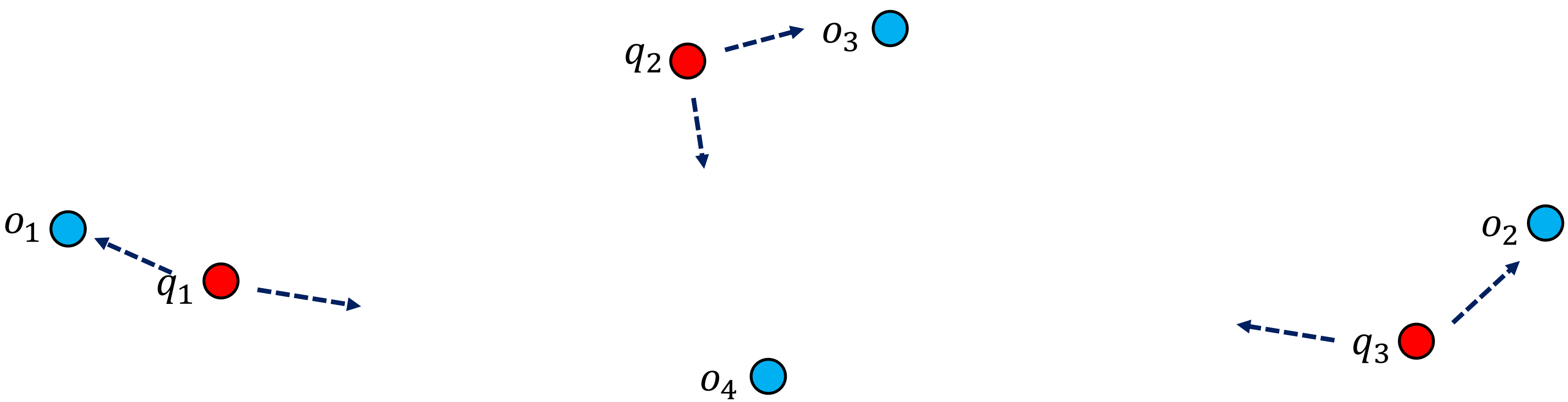


Motivation

- Inefficient to compute distance to every object
- Typical Solution: heuristically retrieve likely candidates until all results found
- But existing heuristics are either:
  - (a) borrowed from kNN => not suitable for AkNN
  - (b) not accurate enough for network distance

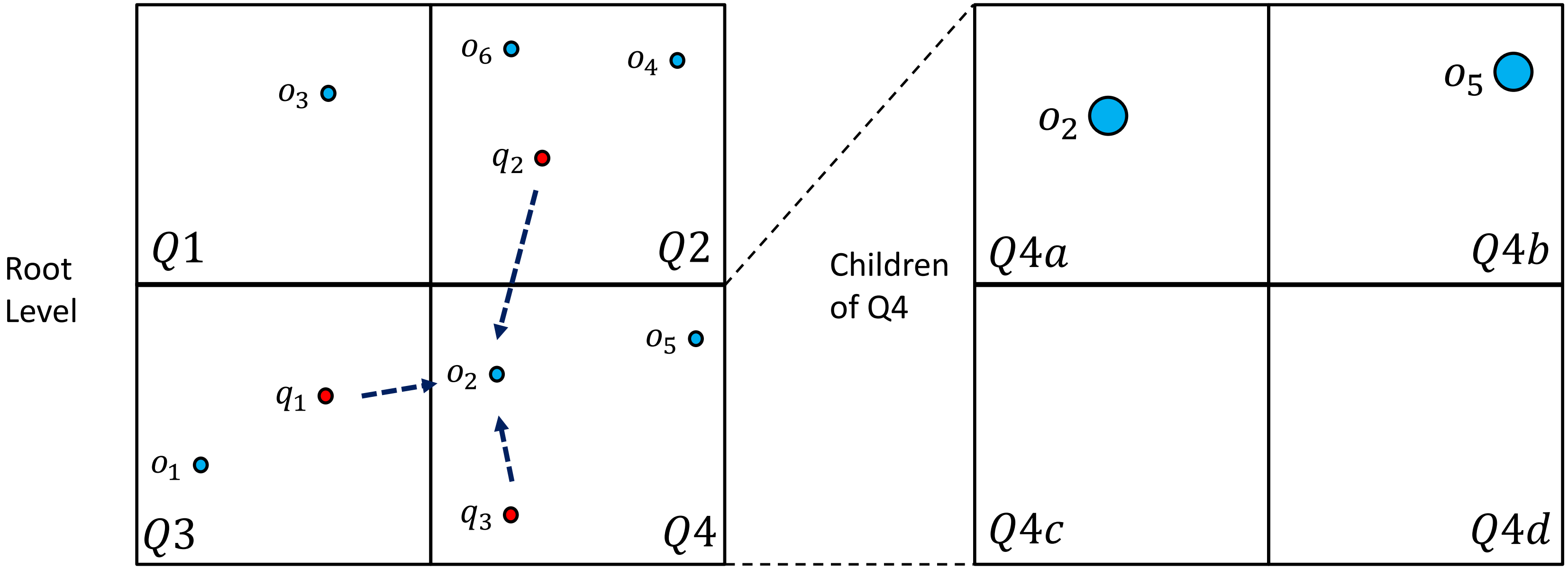
Expansion Heuristics

- Borrowed from kNN search heuristics: expand from each query vertex
- But best AkNN candidates unlikely to be <u>near any one query vertex</u>



Hierarchical Search Heuristic

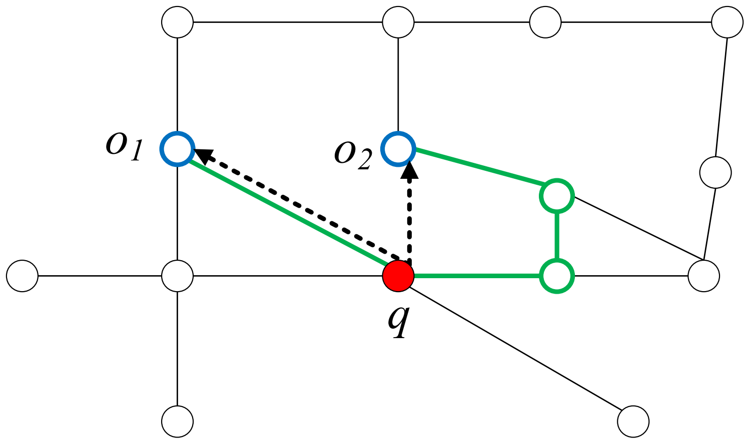
- Divide space to group objects => recursively
- Search "promising" regions top-down (recursively)
- Pinpoint best candidate anywhere in space



Hierarchical Search

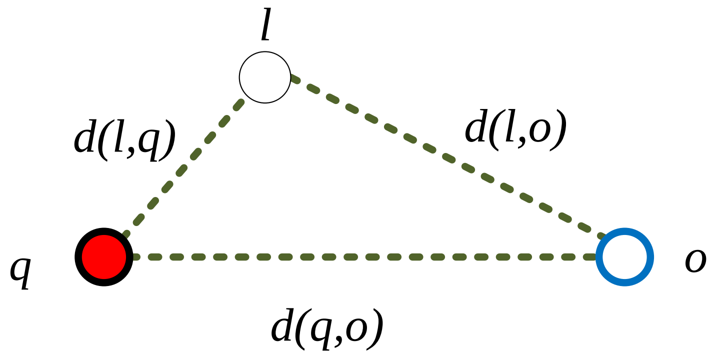
- How do we decide which regions are “promising”?
- Use lower-bound score for all objects in a region
- Past Work: R-tree + Euclidean distance lower-bound
- Not accurate for road network distance

Data structure needed for accurate hierarchical lower-bound search in graphs



Landmark Lower-Bounds

- Pre-compute distances from *landmark* vertices
- Use triangle inequality to compute lower-bound
- Only allows small numbers of landmarks (space cost)
- Not suitable for hierarchical search

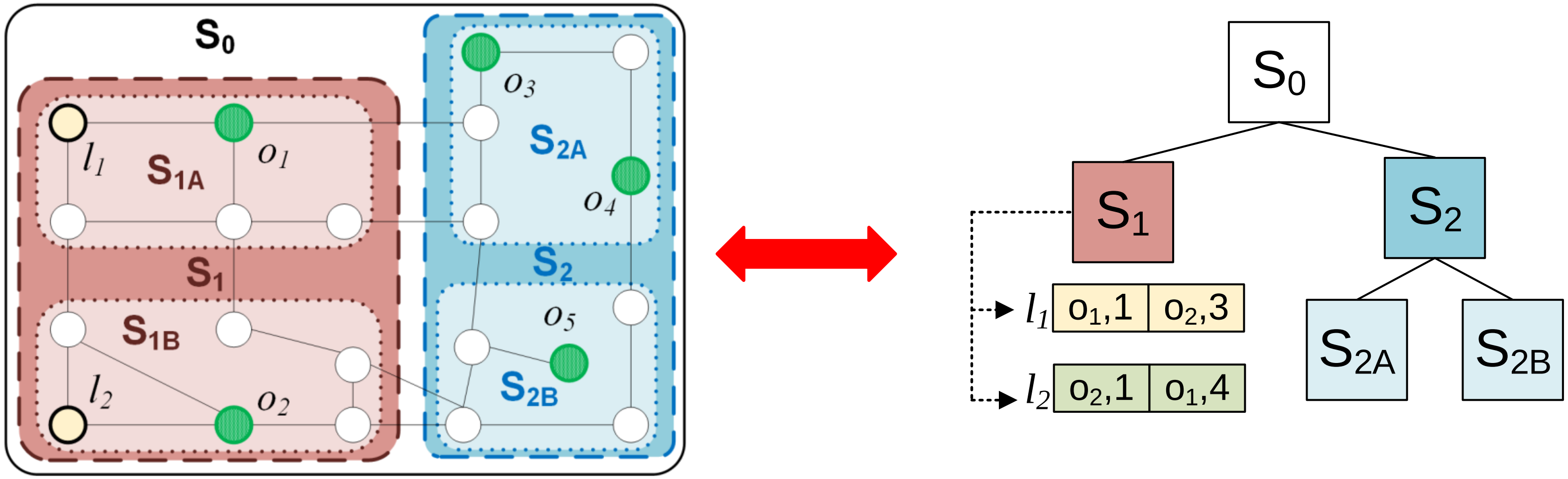


$$d(q,o) \leq |d(l,o) - d(l,q)|$$

Choose tightest LLB over a set of multiple landmarks

Compacted-Object Landmark Tree (COLT) Index

- Partition graph recursively => subgraph tree
- Choose localised landmarks in every subgraph
- Compact based on object set O



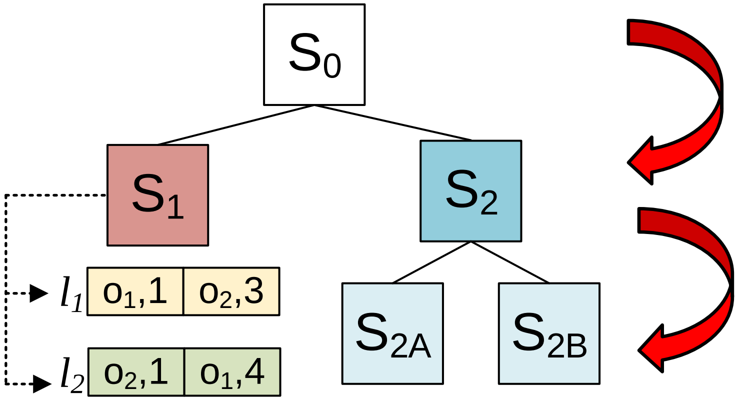
COLT

- Non-leaf + leaf nodes stores
  - M^-: min distance to any object in subgraph from landmark
  - M^+: max distance to any object in subgraph from landmark
- Enables accurate lower-bound for any tree node

$$LB_{l_i}(n_C, q) = \begin{cases} d(l_i, q) - M^+ & \text{if } d(l_i, q) \geq M^+ \\ M^- - d(l_i, q) & \text{if } d(l_i, q) \leq M^- \\ 0 & \text{else} \end{cases}$$

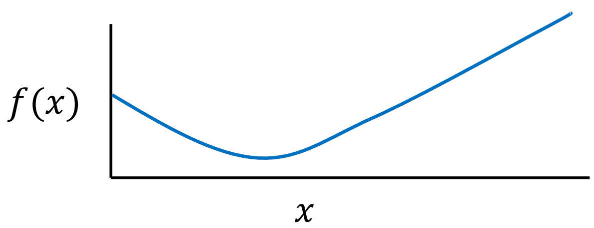
Hierarchical Traversal in COLT

- Top-down search from root node
- Compute lower-bound for child using equation
- Recursively evaluate child with best score



Hierarchical Traversal in COLT

- Leaf nodes store Object Distance List
- Find object with minimum aggregate lower-bound
- Interestingly common functions preserve convexity!
- Easily found using modified binary search



Object	o_4	o_2	o_5	o_1	o_3
Distance	2	5	7	8	12

Experimental Setup

- Dataset: US Road Network Graph from DIMACS
  - $|V| = 23{,}947{,}347$ vertices, $|E| = 57{,}708{,}624$ edges
  - Real-World POIs from OSM for US
- Comparison against IER and NVD
  - IER: hierarchical search using Euclidean heuristic
  - NVD: state-of-the-art expansion heuristic

Query Time: Real-World POIs

- COLT up to an order of magnitude faster!
- COLT performs better on dense POI sets
- Heuristics is less important on sparse POI sets

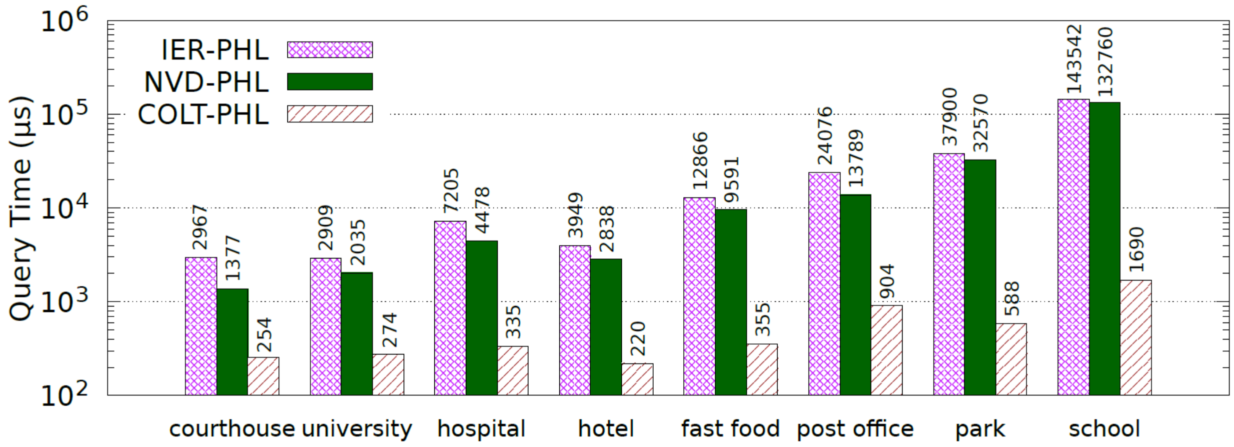
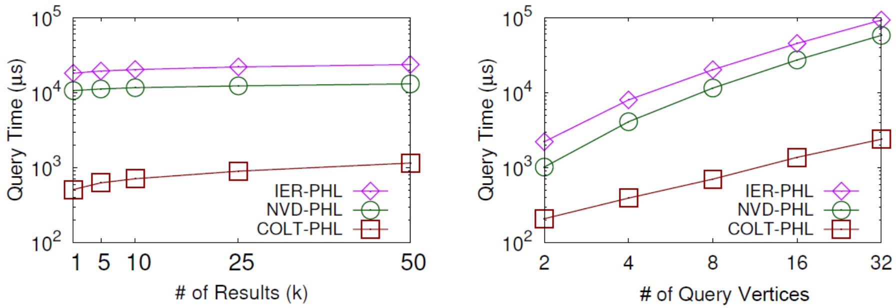


Figure 4: Performance on different real-world POI sets

Sensitivity Analysis

- COLT maintains improvement for
  - Varying parameters (k, number of agents)
  - Varying aggregate functions (MAX, SUM)
  - Heuristic efficiency metrics
- Comes at a lightweight pre-processing cost



(b) Varying k (c) Varying $|Q|$

Figure 5: Performance for max function

Thank You!

Questions?