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

**Tenindra Abeywickrama*, Muhammad Aamir Cheema^, Sabine Storandt#**

*NUS-Grab AI Lab, Monash University^, University of Konstanz #

## Taste it here ...

We study Aggregate k Nearest Neighbour (AkNN) queries in graphs:

- Setting: Graph $G = (V, E)$, Object Vertex Set $O \subseteq V$
- Input: Agent Locations $Q \subseteq V$ , Aggregate Function $f(x)$
- Definition: *Find the nearest object $o \in O$ by its aggregate distance computed to all agents by $f(x)$*

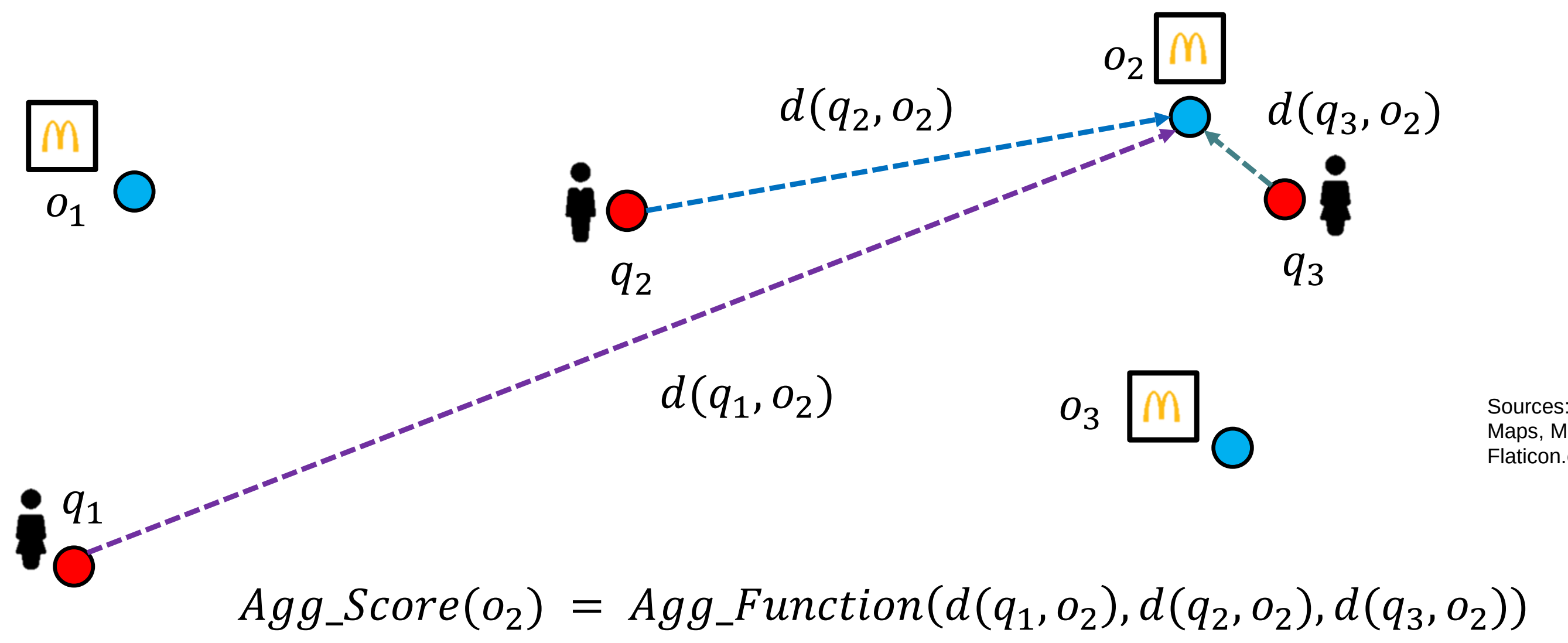
**Key Points:**

- Uses network distance in graph (more diverse + accurate metrics)
- Answered efficiently by using heuristics to retrieve candidates
- But existing heuristics only suitable for <u>single</u> agent
- Propose a new data structure to answer AkNN
- Significantly faster in practice but still lightweight

## Real-World Example

Three friends ($q_1, q_2, q_3$) want to meet at a McDonalds ($o_1, o_2, o_3$) convenient for everyone. Which McDonalds' should they meet at?

Issue this AkNN query: Which McDonalds minimises the SUM of road network distance over all friends?



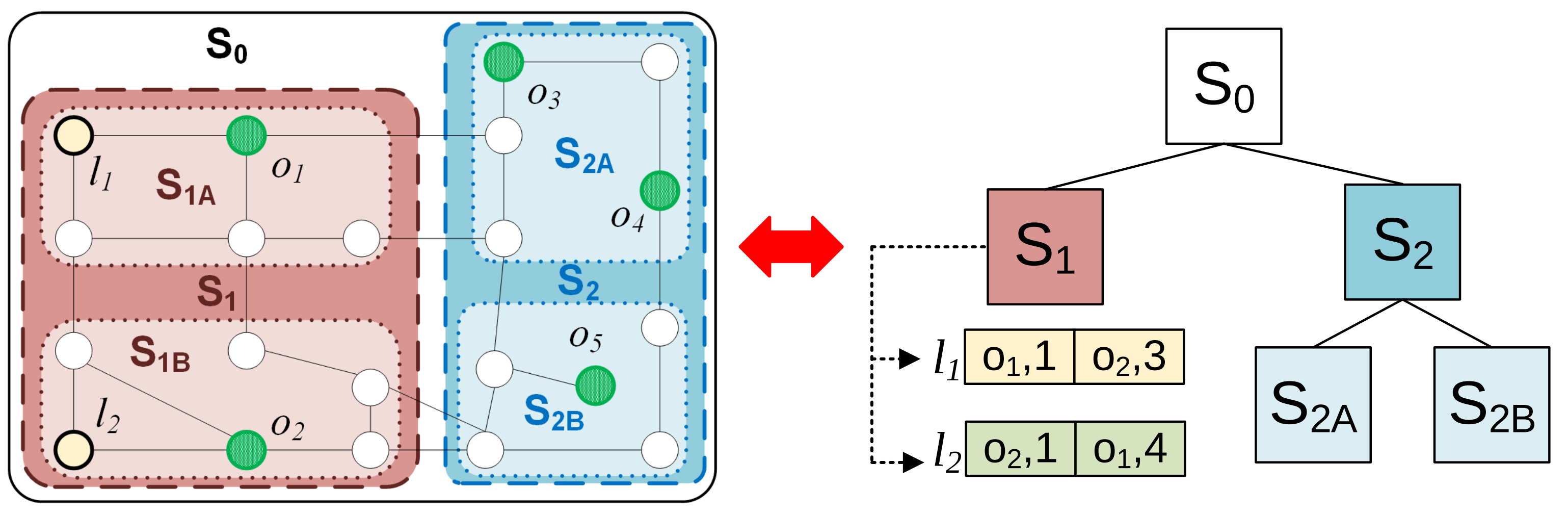
Sources: Google Maps, McDonalds, Flaticon.com

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

## Like it? Read the recipe

Single-agent heuristics rely on the intuition that the nearest objects are in the vicinity of the single agent. This is not true for multi-agent AkNN search, the "nearest" object by aggregate distance unlike to be near one agent! **Recursive hierarchical search is more intuitive**

But existing data structures are not efficient for graphs. We propose: **C**ompacted **O**bject-**L**andmark **T**ree (**COLT**), a hierarchical subgraph tree of the road network customised for object set $O$



## Benefits

COLT: An object-graph index that supports efficient hierarchical search perfectly suited for AkNN search

**Key Benefits:**

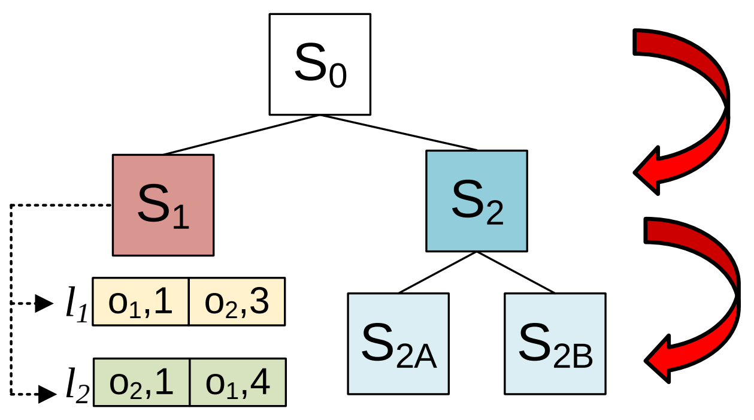
- Novel use of landmark lower-bounds (LLBs) for greater accuracy
- LLBs never previously used for hierarchical search until COLT!
- Landmarks are localized for each tree level => allows us to pinpoint best candidates
- Displays interesting property that makes it particularly efficient for AkNN search for convexity preserving aggregate functions
- Common functions such as SUM and MAX **<u>do</u>** preserve convexity
- Space/time complexity for pre-processing comes at modest cost in both theory and practice: still *lightweight*
- It is a generic graph data structure that can potentially be applied to other graphs and problems, e.g., shortest detour query

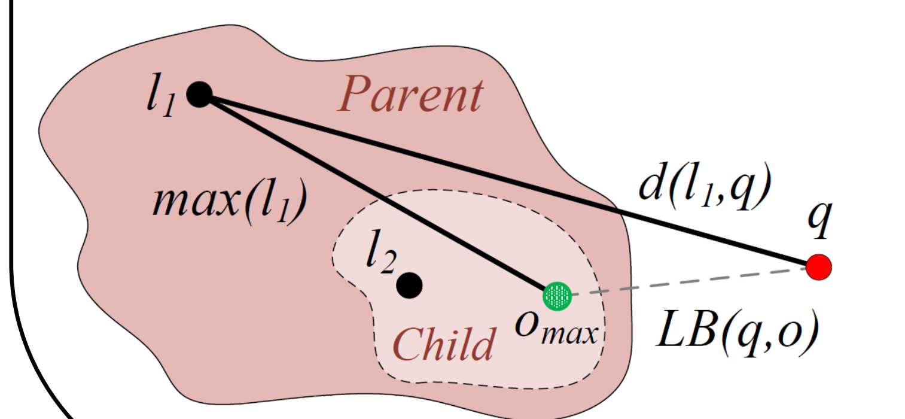
## Still hungry? Please have more

### Top-Down Recursive Search

We find AkNN candidates by conducting a top-down search from the root node in the tree. Each child represents a subgraph of the parent. COLT stores certain values in each tree node that allow a lower-bound aggregate score to be compute for all objects within the subgraph using the equation to the right. Children with the best aggregate score are evaluated recursively until the best candidate object is found

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





Accuracy of landmark lower-bounds increases as we delve deeper into the hierarchy (this is by design). At each level, the landmarks are more local to the objects it contains, leading to more accurate lower-bounds, allowing us to "pinpoint" the best candidates efficiently

### Experimental Results

- Experiments were conducted on real-world road network and POI datasets for United States
- COLT up to an order of magnitude faster than existing techniques!
- Performs better on dense POI sets because it better distinguishes close objects => i.e. a better heuristic
- COLT maintains improvement for varying parameters such as: number of results $k$, number of agents $|Q|$, the aggregate function, and machine independent heuristic efficiency
- Comes at small and/or comparable pre-processing cost in time and space

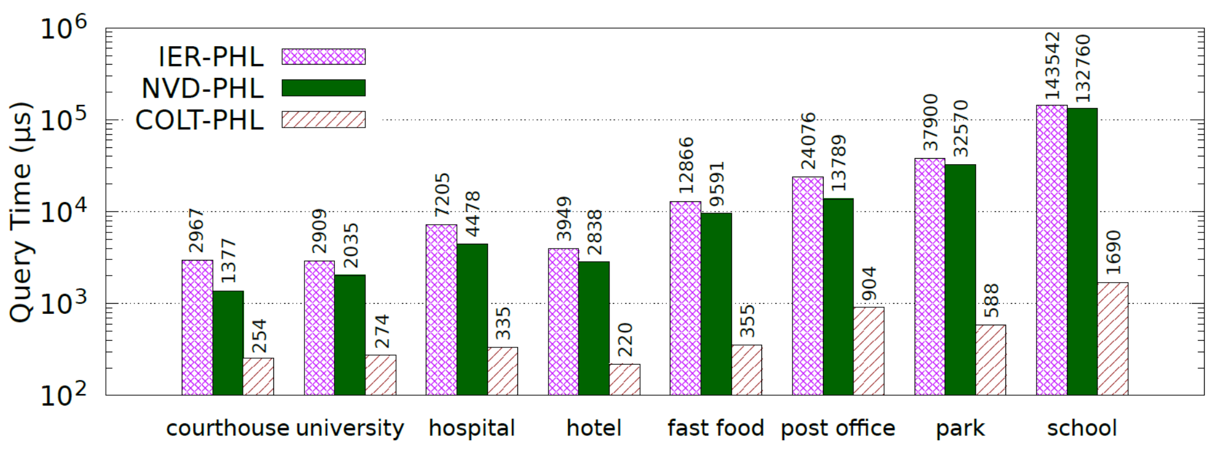
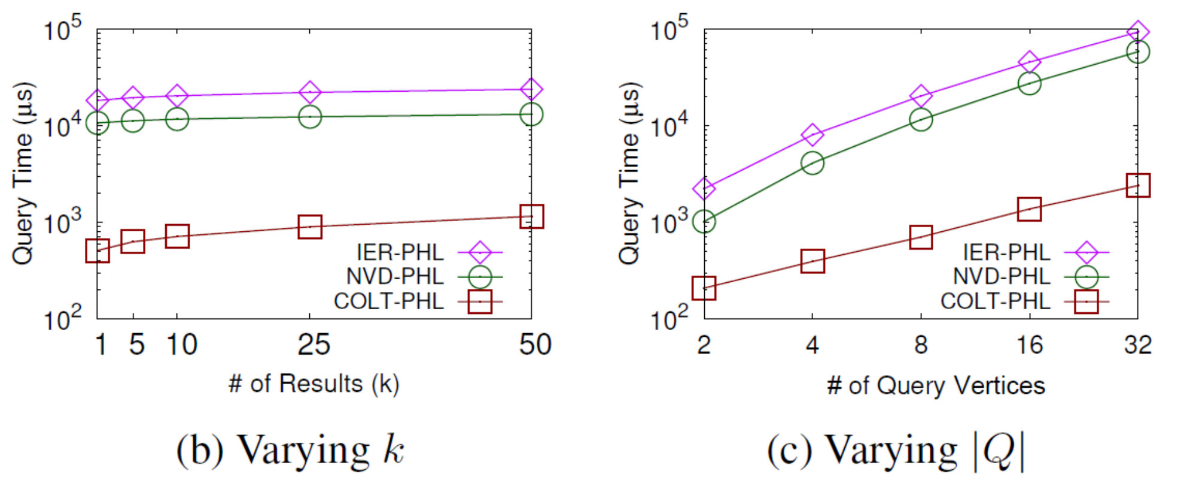


Figure 4: Performance on different real-world POI sets



(b) Varying $k$

(c) Varying $|Q|$

Figure 5: Performance for $max$ function