Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring

Instructor: Cyrus Shahabi
Outline

• Introduction
  – Background, Definition, Motivation

• Related Work
  – Safe Regions, Approximation, YPK-CNN, SEA-CNN

• CPM
  – NN module, Data structure, Handling Updates, Multiple Updates

• ANNs

• Results

• Conclusion
Introduction::Definition

• **CNN**: Continuous Nearest Neighbor search
  – Snapshot: One line query (CNN paper)
  – Continuous: A series of queries and a monitoring system

• **CkNN**: the $k^{th}$ first CNN results
  – Application: Continuously locating 2/more nearest gas stations while driving in a road

**This Paper**
Introduction::Motivation

• **CPM**: Conceptual Partitioning and Monitoring

• Enhancing the performance and memory consumption in CNN searches

• Extend to highly dynamic environments

• Extend for other types of queries (e.g. ANN)
Related Work

- **Snapshot**: using an offline algorithm, all results are computed at once given the whole input

- **Monitoring**: The client continuously asks for NN and a monitoring system on server should be optimized for such a case.
Related Work::Safe Regions

• Moving Query, Static Objects
  – Zhang et. al.: Defines a region around query point (Voronoi cell) where re-computation is not necessary

• Static (Range) Query, Moving Objects
  – Q-index: a list of updates that influence a query is being kept using an R-tree
    • Utilizing the idea of “safe region”
    • Updates only occurs when objects exit this region
Related Work::Approximation

- Koudas et al.: e-approximation kNN over streams of points

  “The returned \( k^{th} \) NN lies at most \( e \) distance units farther from \( q \) than the actual \( k^{th} \) NN of \( q \)”

- Flexible with memory: more memory, smaller \( e \)

- Applied in both snapshot and continuous \( kNN \)
Related Work:: YPK-CNN

- **YPK-CNN[YPK05]:**
  - Objects are assumed to fit in main memory
  - Objects indexed with regular grid cells with fixed size $\delta \times \delta$
  - Applies the updates directly to grids and re-evaluates queries every $T$ time units

- First time queries: a 2 step NN search (more on this later)
- When object moves:
  - Returning queries; update/re-sort points inside the query region
- When Query moves:
  - Start from scratch.
YPK-CNN: First time queries

- **NN Module**: Starts with a rectangle around \( q \), then doubles the nearest distance and creates another box and continues till it finds \( k \) neighbors.
YPK-CNN: First time queries

- **NN Module**: Starts with a rectangle around $q$, then doubles the nearest distance and creates another box and continues till it finds $k$ neighbors.

Step 1:
Visit cells in squares around, until the 1st objects are found.
YPK-CNN: First time queries

**NN Module:** Starts with a rectangle around $q$, then doubles the nearest distance and creates another box and continues till it finds $k$ neighbors.

Step 1:
Visit cells in squares around, until the 1st objects are found.
YPK-CNN: First time queries

- **NN Module**: Starts with a rectangle around \( q \), then doubles the nearest distance and creates another box and continues till it finds \( k \) neighbors.

Step 2:
Search the cells around with side length \( R \).

\[
R = 2 \times d + \delta \\
d = \text{dist}(q, p_1)
\]
YPK-CNN: First time queries

- **NN Module**: Starts with a rectangle around q, then doubles the nearest distance and creates another box and continues till it finds nearest neighbors.

Step 2:
Search the cells around with side length R.

\[ p_2 \text{ is the NN of } q \]
YPK-CNN: Updates

- **Update Handling:** Assume $p_2$ moves.
YPK-CNN: Updates

- **Update Handling:** Assume $p_2$ moves to $p_2'$. The new searching rectangle is defined by $R$.

$$d_{\text{max}} = \text{dist}(q, p_2')$$

$$R = 2 \times d_{\text{max}} + \delta$$
YPK-CNN: Updates

- **Update Handling**: Assume $p_2$ moves. The new searching rectangle is defined by $R$.

\[ d_{\text{max}} = \text{dist}(q, p_2') \]
\[ R = 2 \times d_{\text{max}} + \delta \]

This rectangle guarantees to contain at least one object ($p_2'$).
YPK-CNN: Updates

- **Update Handling:** Assume \( p_2 \) moves. The new searching rectangle is defined by \( R \).

\[
\begin{align*}
    d_{\text{max}} &= \text{dist}(q, \ p_{2}') \\
    R &= 2 \times d_{\text{max}} + \delta
\end{align*}
\]

YPK collects all the objects in this rectangle to identify new NN.
Related Work:: SEA-CNN

- **SEA-CNN[XMA 05]**
  - Objects are stored in secondary memory, indexed with a regular grid.
  - Exclusively focuses on monitoring
  - Assume the initial NN result is available.

- **Key features:**
  - Uses circles instead of rectangles to define “answer region”
  - Circle radius is the distance of the $k^{th}$ NN
SEA-CNN: Object Moving

• **SEA-CNN.** Exclusively focuses on monitoring without any first-time NN module.

• Initial “answer region”

  If there is no node moving out:
  1) Nodes only moves within this region
  2) Some nodes enter this region

• Collect all nodes in this region to find NN
SEA-CNN: Object Moving

- **SEA-CNN.**: Exclusively focuses on monitoring without any first-time NN module.

If there are nodes moving out:

\[ p_2 \rightarrow p_2' \]

new \( r = d_{\text{max}} \)

\[ d_{\text{max}} = \text{dist}(q, p_2') \]

(the previous NN that moves furthest)
SEA-CNN: Object Moving

- **SEA-CNN.**: Exclusively focuses on monitoring without any first-time NN module.

If there are nodes moving out:
- $p_2 \rightarrow p_2'$

- Update “answer region”

- Collect all nodes in this region to find NN
SEA-CNN: Query Moving

- **SEA-CNN.**: Query updates Process

- Initial $r = \text{dist}(q, p_2)$
SEA-CNN: Query Moving

- **SEA-CNN.**: Query updates Process.
  - When q moves to q’

  - Initial $r = \text{dist}(q, p_2)$
  - New $r’ = r + \text{dist}(q, q’)$
SEA-CNN: Query Moving

- **SEA-CNN.**: Query updates Process.
  - When q moves to q’

- New r’ = r + \text{dist}(q, q’)

- Update “answer region”

- Collect all nodes in this region to find NN
**CPM::NN Module**

**SEA-CNN Drawbacks:**
- Hard to maintain “answer regions” for circles on grids

**CPM:**
- Using the rectangle shaped cells on the grid to index objects
CPM: NN Module

- Notation:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_dist(c,q)</td>
<td>Minimum distance between cell c and query q</td>
</tr>
<tr>
<td>best_NN</td>
<td>The list of k best NNs found so far</td>
</tr>
<tr>
<td>best_dist</td>
<td>The distance of k-th NN in list best_NN</td>
</tr>
<tr>
<td>δ</td>
<td>The width of each grid</td>
</tr>
</tbody>
</table>

- Lemma: each rectangle min_dist increases by δ from one level to the upper level.

  e.g. \( \text{mid\_dist}\ (L_1, q) = \text{mid\_dist}\ (L_0, q) + \delta \).
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

Current Heap:
- Insert rectangles of level zero into heap

Insert rectangles of level zero into heap

Current Heap:
- Insert rectangles of level zero into heap

(0,0)
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

De-heap
C(4,4)

Empty -skip

Visited List: \{c(4,4)\}
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

De-heap Uo

1. Add all the cells in Uo
2. Add next level rectangle (i.e., U1)

De-heap Next?

Visited List: \{c(4,4), c(4,5)\}
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

De-heap until find the first non-empty cell -> Cp₁

Update: best_dist = dist (p₁,q) = 1.7
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

Continue with rectangles of level one in the heap. Then extract again from top and re-insert cells.
CPM::NN Module

- Insert each rectangle starting from lower level into a heap with its min_dist (from q). Same with cells. De-heap and extract them and add them to visit list.

de-heap cells until it hits \( C_p_2 \)

\[
dist(p_2, q) = 1.3
\]

At this point the algorithm will stop, why?

Update:
best_dist = dist \( (p_2, q) \) = 1.3
CPM::NN Module

• Influence Region for q
  – The set of cells that intersect the circle centered at q with radius best_dist

Now:
best_dist = dist
(p₂,q) = 1.3

Influence region:
The blue cells
For each query, we keep:
1) Position
2) best_NN + best_dist
3) Search heap
4) Visited cell list

For each cell, we keep:
1) the objects inside it
2) the queries where the cell is in its influence region
CPM::Handling Updates

• If an object moves into a query’s influence region
  – best_NNs list just need to be re-ordered including the new object
CPM::Handling Updates

• If the previous NN moves out then query need re-computation.
  – Re-computation will utilize the previous heap and visited list

To search for objects:

1) Start with the visited list
2) Then, go to the heaps
CPM::Handling Updates

• If the previous NN moves out then query need re-computation.
  – Re-computation will utilize the previous heap and visited list

Benefits:

1) Avoid computing min_dist between q and visited cells
2) Reduce the heap operations (en-heap, de-heap)
CPM::Handling Updates

- If the previous NN moves out then query need re-computation.
  - Re-computation will utilize the previous heap and visited list

After re-computation:

1) $p'_4$ is found
2) Visited list and heap should be updated
CPM::Handling Updates

• If a query is moving
  – Delete the query together with all the data structures (e.g., influence region)
  – Re-compute NN from scratch.
CPM::Multiple Updates

• The mentioned approach is not efficient because:

  – Updates may cancel each other
  – We may have more updates than queries
  – When to re-compute? Timestamp, trig by updates, trig by returning query?
CPM::Multiple Updates

- The general solution is to keep a list of new nodes that entered a query region (I) and the outgoing ones (O).
  - if |I| >= |O| then it means we still have enough NNs in best_NN to be able to re-order, else query needs re-computation.
Other types of Queries (ANN)

- **Aggregate Nearest Neighbor**: “Given a set of query points $Q = \{q_1, q_2, \ldots, q_m\}$, a sum ANN query continuously reports the data object $p$ that minimizes $a_{dist}(p, Q) = \sum_{q_i \in Q} \text{dist}(p, q_i)$”.

**In simple English**: where should we all meet minimizing the total traveling distance?
Other types of Queries (ANN)

- **ANN with CPM**: make an MBR around all query points and then have the rectangles around them. The only difference is instead of distance we use sum of distances as the heap key.

(a) Partitioning into rectangles  
(b) Influence region
Analysis

• The authors of the paper analytically calculated the time and space complexity of each operation with the assumption of uniformly distributed objects and arbitrary query points.

• They also qualitatively compared it with YPK-CNN and SEA-CNN.

• Later they matched these claims with experimental results.
Results

• They showed $\delta = 1/128$ (of the grid) is the optimal cell size using experimental results.

• Almost no effect from number of objects and queries

• Results from object and query speed

• And object / query agility (percentage of objects that move within a timestamp)
Results

(a) Effect of $N$

(b) Effect of $n$
Results

(a) CPU time

(b) Cell accesses

Cell Access: No. of cell access per query per time stamp
Results

(a) Effect of object speed

(b) Effect of query speed
Results

(a) Effect of object agility ($f_{obj}$)  (b) Effect of query agility ($f_{qry}$)
Results

(a) Constantly moving queries
(b) Static queries
Conclusion

- CNN algorithm with minimal overhead for repeated queries
- Monitoring system
- Useable for ANN queries
- Can handle user-constrained NN search (e.g. specific region)
- No knowledge about moving objects and speed is required
References


• A presentation by Penny Pan in csci587 Fall’2010