Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring

By: Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias
SIGMOD Conference, 2005

Presented by: Kaveh Shahabi
CS 599: Geospatial Information Management - Fall '09
Sep 16th, 2009

Introduction

- Background, Definition, Motivation
- Related Work
  - Safe Regions, Approximation, YPK-CNN, SEA-CNN
  - CPM
    - NN module, Data structure, Handling Updates, Multiple Updates
  - ANNs
  - Analysis
    - Analytical, Qualitative
  - Results
  - Conclusion

Outline

Introduction

- NN: Finding the nearest neighbor to a query point in space
- Applications in GIS, Vision, Database, etc.
- kNN: returns top k nodes closest to the query point.

Introduction: Definition

- CNN: Continuous Nearest Neighbor search
  - Snapshot: One line query (B1 paper)
  - Continuous: A series of queries and a monitoring system
- CkNN: the kth first CNN results
  - Application: Continuously locating nearest gas stations while driving in a road

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

- Zhang et. al.: Defines a region around query point (Voronoi cell or expiry time) were re-computation is not necessary
- Q-index: a list of updates that influence a query is being kept using an R-tree
- MQM: Each object has a resident domain assigned by the server

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 $q$”
- Is flexible with memory: more memory smaller $e$
- Both snapshot and continuous $ekNN$

YPK-CNN

- Yu et al. [YPK05]: regular grid cells with fixed size $\delta \times \delta$ as index
- Applies the updates directly and re-evaluates queries every $T$ time units
- First time queries; a 2 step NN search
- Returning queries; update/re-sort points inside the query region

YPK-CNN

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

\[
R = 2 \times d_{max} + \delta
\]
**YPK-CNN**

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

\[
R = 2 \times d_{\text{max}} + \delta
\]

- **Update Handling**: Assume \( p_2 \) moves. Now \( d_{\text{max}} \) is max distance of previously discovered neighbors.

\[
R = 2 \times d_{\text{max}} + \delta
\]

**SEA-CNN**

- **SEA-CNN**: Exclusively focuses on monitoring without any first-time NN module. It also handles the special case of no neighbor node moving out.

- Uses circles instead of rectangles.
- Circle radius is the distance of the \( k^{th} \) NN.
- If there is no node moving out then special case otherwise similar to YPK-CNN.
- **SEA-CNN.** Exclusively focuses on monitoring without any first-time NN module. It also handles the special case of no neighbor node moving out.

  - Uses circles instead of rectangles.
  - Circle radius is the distance of the k-th NN.
  - If there is no node moving out then special case otherwise similar to YPK-CNN.

- **SEA-CNN.** Exclusively focuses on monitoring without any first-time NN module. It also handles the special case of no neighbor node moving out.

  - Uses circles instead of rectangles.
  - Circle radius is the distance of the k-th NN.
  - If there is no node moving out then special case otherwise similar to YPK-CNN.
**CPM::NN Module**

- Same grid cell with fixed size index structure.
- Uses circles to search cells (rectangles).
- If \( \text{min\_dist} \) of a cell (rectangle) is larger than or equal to the distance of the discovered node (\( k_{th} \) NN) then omit the cell.
- Terminates after discovering \( k \) NNs.

**NAÏVE APPROACH**

**RECTANGLES**

- Main contribution is the rectangle shaped cells on the grid to index objects.

**Lemma:** Each rectangle \( \text{min\_dist} \) increases by \( \delta \) from one level to the upper level.

**Insert each rectangle starting from lower level into a heap with its \( \text{min\_dist} \). Same with cells. De-heap and extract them and add them to visit list.**

**Insert level zero into heap**

**De-heap \( C_q \) \( \to \) empty**

**De-heap \( U_0 \) \( \to \) 2 cells**

**Insert each rectangle starting from lower level into a heap with its \( \text{min\_dist} \). Same with cells. De-heap and extract them and add them to visit list.**

**Insert level zero into heap**

**De-heap \( C_q \) \( \to \) empty**

**De-heap \( U_0 \) \( \to \) 2 cells**
CPM::NN Module
- Insert each rectangle starting from lower level into a heap with its min_dist. Same with cells. De-heap and extract them and add them to visit list.

De-heap until the first non-empty cell = Cₚ₁
Level = 1
best_dist = dist(pₘ, q) = 1.7

CPM::NN Module
- Insert each rectangle starting from lower level into a heap with its min_dist. Same with cells. De-heap and extract them and add them to visit list.

The next item in heap (R₀) has key lower than 1.7 so it de-heaps

CPM::NN Module
- Insert each rectangle starting from lower level into a heap with its min_dist. Same with cells. De-heap and extract them and add them to visit list.

Continue inserting (re-heap) 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. Same with cells. De-heap and extract them and add them to visit list.

de-heap cells until it hits Cₚ₁ dist(pₘ,q) = 1.3
At this point the algorithm will stop because heap root node is Cₚ₄ which has key larger than 1.3

CPM::Data Structure
- For each query the heap, closed (visited) list, kth NN distance, and the NNs are being kept
- For each cell only the objects inside and the associated queries are being kept

CPM::Handling Updates
- If an object moves in to a query region (circle with radius best_dist) then best_NNs just need to be re-ordered including the new object
If it moves out then query need re-computation. Re-computation will continue from previous heap until next NN with distance lower than heap root node key.

---

If it moves out then query need re-computation. Re-computation will continue from previous heap until next NN with distance lower than heap root node key.

---

If it moves out then query need re-computation. Re-computation will continue from previous heap until next NN with distance lower than heap root node key.

---

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?

---

The general solution is to keep a list of new nodes that entered a query region (I) and the outgoing ones (O). When a query returns, 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.

---

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 \( adist(p, Q) = \sum_{q \in Q} dist(p, q) \)."

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

- **ANN with CPM**: make a 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.

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)

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

- Figure 6.2: CPU time versus $N$ and $n$
- Figure 6.3: Performance versus $\delta$
- Figure 6.4: CPU time versus object and query speed
Results

- 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

Conclusion

Questions?

Thank you for your attendance