Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring

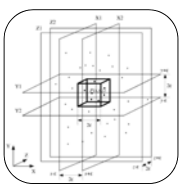
By: Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias
SIGMOD Conference, 2005

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

Outline

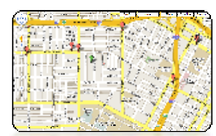
Introduction::Background

- **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 k^{th} 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

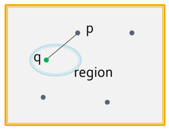
- **Snapshot:** using an offline algorithm, all results are computed at once given the whole input
- **Monitoring:** The client continuously asks for NN and a monitoring system on server should be optimized for such a case.

Method	Query	Memory	Processing	Result
Q-index	Range	Main	Distributed	Exact
MQM	Range	Main	Distributed	Exact
Mobieyes	Range	Main	Distributed	Exact
SINA	Range	Disk	Centralized	Exact
DISC	NN	Main	Centralized	Approximate
YPK-CNN	NN	Main	Centralized	Exact
SEA-CNN	NN	Disk	Centralized	Exact

Table 2.1: Properties of monitoring methods

Related Work::Safe Regions

- Zhang et. al.: Defines a region around query point (Voronai 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*

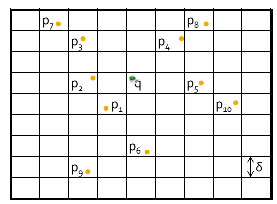
YPK-CNN

- **Yu et al. [YPK05]:** regular grid cells with fixed size δ x δ 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.

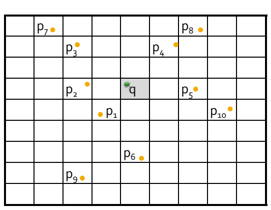
$R = 2 \times d_{max} + \delta$



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.

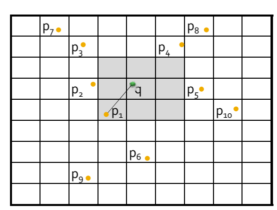
$R = 2 \times d_{max} + \delta$



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.

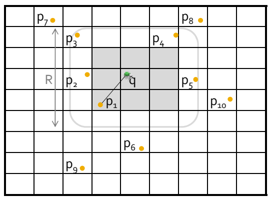
$R = 2 \times d_{max} + \delta$



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.

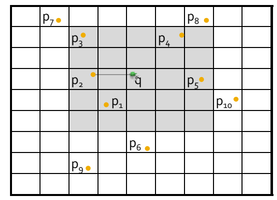
$R = 2 \times d_{max} + \delta$



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.

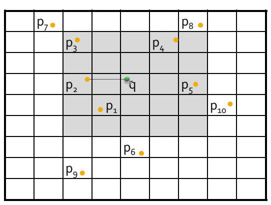
$R = 2 \times d_{max} + \delta$



YPK-CNN

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

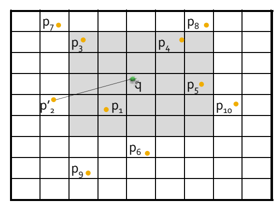
$R = 2 \times d_{max} + \delta$



YPK-CNN

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

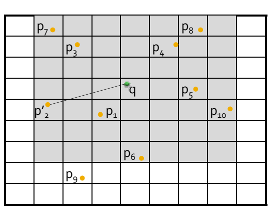
$R = 2 \times d_{max} + \delta$



YPK-CNN

- **Update Handling:** Assume p_2 moves. Now d_{max} is max distance of previously discovered neighbors. Only one rectangle needed.

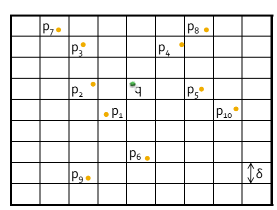
$R = 2 \times d_{max} + \delta$



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.

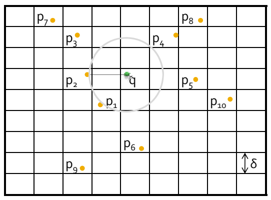
  - Uses circles instead of rectangles
  - Circle radius is the distance of the k^{th} NN
  - If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

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

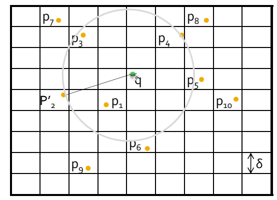
- Uses circles instead of rectangles
- Circle radius is the distance of the k^{th} NN
- If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

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

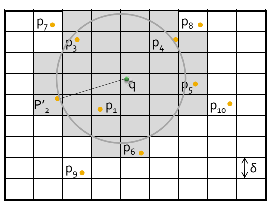
- Uses circles instead of rectangles
- Circle radius is the distance of the k^{th} NN
- If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

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

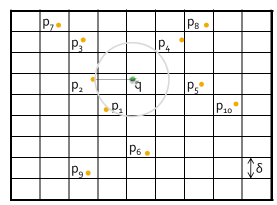
- Uses circles instead of rectangles
- Circle radius is the distance of the k^{th} NN
- If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

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

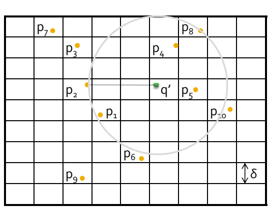
- Uses circles instead of rectangles
- Circle radius is the distance of the k^{th} NN
- If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

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

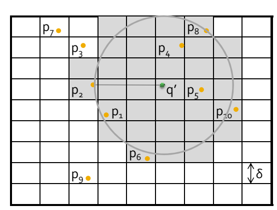
- Uses circles instead of rectangles
- Circle radius is the distance of the k^{th} NN
- If there is no node moving out then special case otherwise similar to YPK-CNN



SEA-CNN

- **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

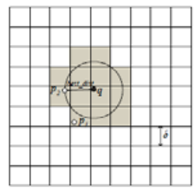


CPM::NN Module

- Same grid cell with fixed size index structure.
- Uses circles to search cells (rectangles).
- If *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.

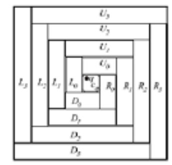
CPM::NN Module

NAÏVE APPROACH



RECTANGLES

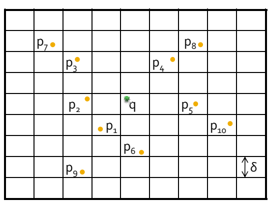
- Main contribution is the rectangle shaped cells on the grid to index objects



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.

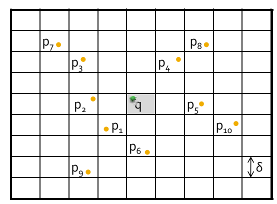
Lemma: each rectangle min_dist increases by δ from one level to the upper level.



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.

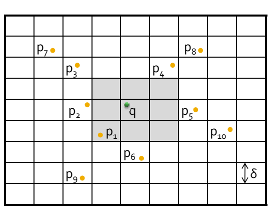
Insert level zero into heap



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.

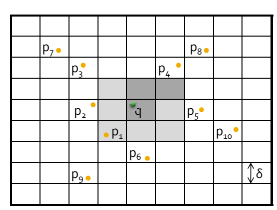
Insert level zero into heap



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.

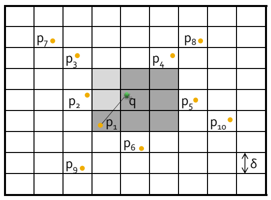
De-heap C_q -> empty
De-heap U_0 -> 2 cells



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.

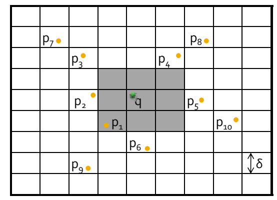
De-heap until the first non-empty cell -> Cp_1
Level = 1
best_dist = dist(p_1,q) = 1.7



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.

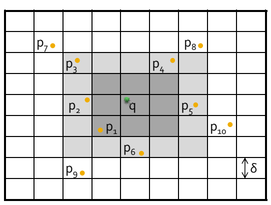
The next item in heap (R_0) has key lower than 1.7 so it de-heaps



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 (en-heap) rectangles of level one in the heap. Then extract again from top and re-insert cells

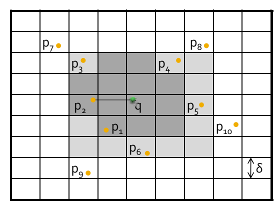


CPM::NN Module

- 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.

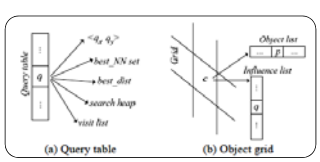
de-heap cells until it hits Cp_2
dist(p2,q) = 1.3

At this point the algorithm will stop because heap root node is Cp_4 which has key larger than 1.3



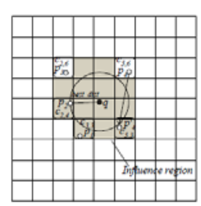
CPM::Data Structure

- For each query the heap, closed (visited) list, k^{th} 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

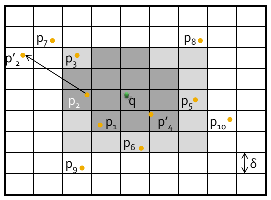
- If an object moves in to a query region (circle with radius best_dist) then best_NNs just need to be re-ordered including the new object



CPM::Handling Updates

- 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

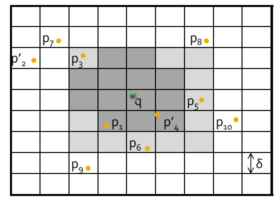
Initials previous heap and visit list with updated objects



CPM::Handling Updates

- 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

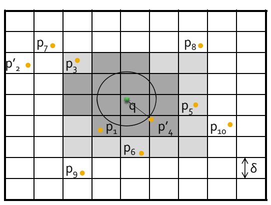
Re-visit cells first from visit list and then put into original heap until it hits Cp'4



CPM::Handling Updates

- 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

Since there are no more non-empty cells in this circle the search will terminate with no more addition to the visit list

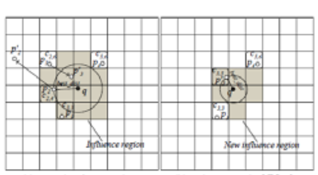


CPM::Multiple Updates

- 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). 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.

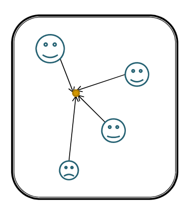


(a) p_2 and p_3 issue updates (b) p_3 becomes the NN of q

Other types of Queries (ANN)

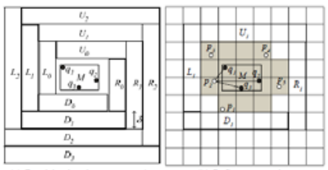
- **Aggregate Nearest Neighbor: "**Given a set of query points Q = {q1,q2,…,qm}, a sum ANN query continuously reports the data object p that minimizes adist(p,Q) = ΣqiЄQ dist(p,qi)".

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.


(a) Partitioning into rectangles (b) Influence region

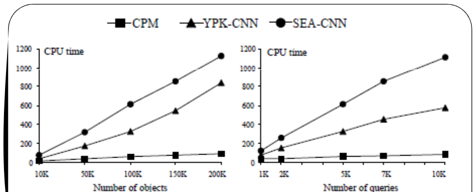
Analysis

- 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 δ = 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)

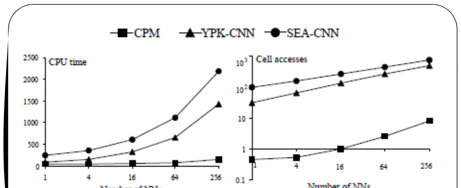
Results



(a) Effect of N (b) Effect of n

Figure 6.2: CPU time versus N and n

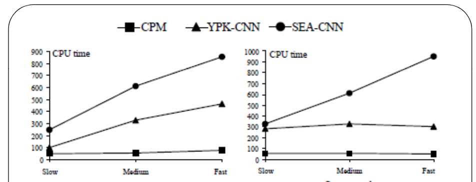
Results



(a) CPU time (b) Cell accesses

Figure 6.3: Performance versus k

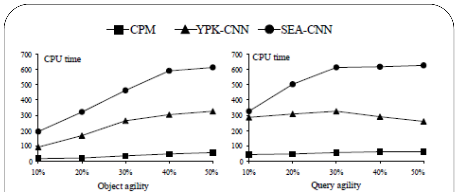
Results



(a) Effect of object speed (b) Effect of query speed

Figure 6.4: CPU time versus object and query speed

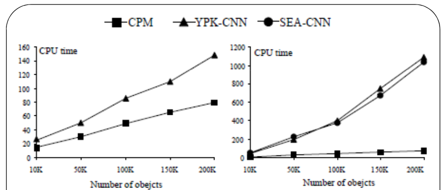
Results



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

Figure 6.5: CPU time versus object and query agility

Results



(a) Constantly moving queries (b) Static queries

Figure 6.6: CPU time for constantly moving and 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

Thank you for your attendance

Questions?