International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:6, 2014

Searching *k*-Nearest Neighbors to be Appropriate under Gamming Environments

Jae Moon Lee

***Abstract*—**In general, algorithms to find continuous *k*-nearest neighbors have been researched on the location based services, monitoring periodically the moving objects such as vehicles and mobile phone. Those researches assume the environment that the number of query points is much less than that of moving objects and the query points are not moved but fixed. In gaming environments, this problem is when computing the next movement considering the neighbors such as flocking, crowd and robot simulations. In this case, every moving object becomes a query point so that the number of query point is same to that of moving objects and the query points are also moving. In this paper, we analyze the performance of the existing algorithms focused on location based services how they operate under gaming environments.

***Keywords*—**Flocking behavior, heterogeneous agents, similarity, simulation.

I. Introduction

THE process of finding the *k*-nearest neighbors (kNN) from a given point have been studied focusing on spatial databases. Most studies have indexed data objects using space access methods like KD-tree, R-tree [1]. Heuristics were used to reduce the searching space [2], [3], [8], [9]. Various methods to find the kNN under circumstances where the object is moving have also been studied [4]-[6], [9]. It is called as the continuous *k*-nearest neighbors (CkNN). These methods confront a time and space restraint in providing the necessary information to search the kNN.

Based on the Location Based Service (LBS) [7], [11], efficient methods to search the CkNN were proposed in [4]-[6]. In order to find the CkNN in moving objects, all conventional algorithms have used the grid structure. The CkNN monitoring method that uses object index and query index for moving objects have been proposed in [4]. Specifically, the overhaul method and the incremental method about the object index were suggested. A hierarchical version of the object index was also proposed to enhance the overall performance when moving objects are not evenly distributed. Another efficient algorithm was also brought up so called the conceptual partitioning method (CPM) [6]. The CPM creates a virtual set of cells inside a square that is close to the query points. This algorithm utilizes these virtual squares to reduce the number of accessing cells.

Conventional studies have been focused on algorithms that work efficiently in an environment where the query point is fixed to a moving object [4]-[6]. Also, the number of query points was also set to a small number compared to the number of objects [4]-[6]. This is because conventional studies have all been focused on LBS services. However, when the kNN is required for the agent to make a decision about the surrounding environment and the agent is a moving object like a robot, flocking and crow simulation in a game, then the kNN must be calculated with a certain basis [2], [3], [8], [10]-[12]. Thus, in these environments, each of the objects becomes a query point. Therefore the number of query points is identical to that of objects and the object and the query point moves simultaneously. This paper focuses on analyzing the performance of conventional algorithms in gaming environments such as the previously mentioned.

In Chapter III, conventional CkNN algorithms were introduced. Chapter III analyzes the performance of the algorithms in a game environment. Chapter IV focuses on the conclusions.

Jae Moon Lee is with the Dept. of Multimedia Engineering, Hansung University, 389 Samsundong Sungbukgu Seoul Korea (e-mail: jmlee@hansung.ac.kr).

II. Related Works

A. YPK-CkNN

All conventional algorithms used the grid structure when searching CkNN in moving objects. Using the grid is a very simple process. First, the space that one wishes to control must be divided into a grid, $m \times m$. Every moving object must be stored in the cell that includes its position. If the object is continuously moving and had moved to a different cell location, the object is deleted in the previous cell and is stored in the new cell. By doing so, near neighbors of a given query point is at a cell that stores the query point and neighboring cells. In order to search the kNN, we will only have to evaluate objects stored to the cell and the neighboring cells instead of all objects.

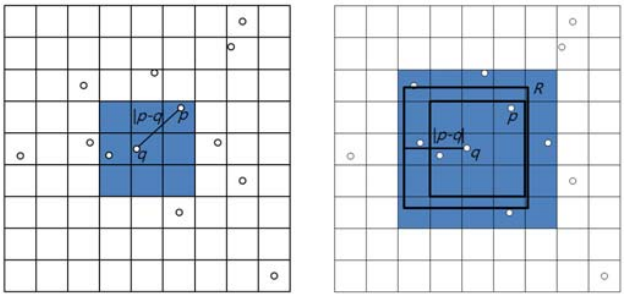

Fig. 1 Grid method in YPK-CkNN

YPK-CkNN, proposed in [4], is the most basic method to use the grid structure. Let's consider Fig. 1, where we assume that *k* is 2. As shown in the left of Fig. 1, the searching begins with the cell including query point q and sequentially expands to its neighboring cells until 2 objects are founded. Then it finds the

object *p* located furthest from *q* as shown in Fig. 1. It calculates the distance |*p*-*q*| between the two objects. Let *R* be the rectangle with the side length 2|*p*-*q*| centered at *q*. It evaluates the objects in the cells which are contained in *R* and overlapped with *R* and finds the *k*-nearest neighbors of the query point *q*.

overhaul(inputs: *q, G, k* output: *kNN(t)*)
1. By using the grid *G*, find the rectangle which centers the cell including *q* and contains at least *k* objects. 2. Find the object *p* which is in the rectangle and is farthest from *q*. 3. By using the grid *G*, find all the object in the rectangle with the side length 2|*p*-*q*| centered at *q*. 4. Find and return the *k*-nearest neighbors from the objects found in the step 3.

Fig. 2 YPK-Overhaul Algorithm

Fig. 2 is overhaul algorithm proposed in [4]. The advantage of this method is that while searching for the kNN, it only evaluates objects stored in cells that are adjacent to the query point. Reference [4] proposed additional methods to enhance the performance of the overhaul method; to find new kNN using the previous kNN in time *t*. It is important to note that the square in overhaul with an area of 2|*p*-*q*| is not the main focus, but that in the rectangle with the side length 2|*p*-*q*| centered at query point *q*, there is at least more than *k* number of objects. Using this fact, the incremental method in proposed [4] separately stores kNN evaluated in *t-1* until time *t*. This distance is then used to execute the step 3 and the step 4 of the overhaul method. The following is the algorithm explained.

incremental(inputs: *q, G, k, kNN(t-1)* output: *kNN(t)*)
1. Compute |*kNN(t-1)*-*q*|. 2. By using the grid *G*, find all the object in the rectangle with the side length 2|*kNN(t-1)*-*q*| centered at *q*. 3. Find and return the *k*-nearest neighbors from the objects found in the step 2.

Fig. 3 YPK-Incremental Algorithm

In this algorithm the kNN(*t-1*) is the kNN of *q* searched at time *t-1* and |*kNN(t-1)*-*q*| is the minimum radius of the circle that includes all objects in the kNN(*t-1*) in time *t*. The incremental method is simpler than the overhaul method. That is, the procedure to find |p-q| in the overhaul algorithm was replaced to computing |*kNN(t-1)*-*q*|. In most cases, the cost of the step 1 in Fig. 2 is less than that of the step 1 and 2 in Fig 2. When the moving speed of the objects is slow, this method shows great performance. However, when the moving speed of the objects are fast, the possibility of object to move further increases rapidly and thus the value of |*kNN(t-1)*-*q*| increases exponentially. In this case, the rectangle *R* shown in Fig. 1 is bigger and number of cells and objects to evaluate increases and consequentially the performance of the incremental method will be bad. Reference [4] has experimentally showed this fact.

B. CPM-CkNN

In order to reduce unnecessary access of the cells in the conventional YPK-CkNN method, [6] proposed a new method CPM-CkNN. CPM-CkNN has defined a DIR(direction) that includes various cells such as Fig. 4. In Fig. 4, U0, U1, ..., L2 are the DIR. The DIR has a level. As shown in the figure, the DIR level is incremented by 1 and starts with DIR 0 closest from the query point. In the CMP-CkNN method, every cell has a distance from the query point. This distance is determined as the closest distance between the query point and the random points in the cell. Thus, this distance is the closest, without exception of any objects included in the cell. The DIR also has a distance from the query point. This distance is determined as the closest distance among the cells included in the DIR. This distance is expressed as *mindist*<*DIRi, q*>.

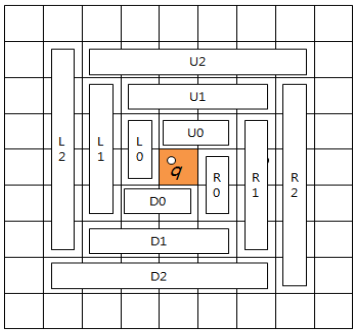

Fig. 4 Conceptual partitioning method

The algorithm suggested in [6] uses heap data structure about cell and DIR. Thus, according to their distance, it stores cell or DIR that are potentially searching cell in the heap and when needed, it is deleted from the heap to evaluate necessary objects. Fig. 5 is the NN_Computation algorithm suggested in [6].

NN_Computation(inputs: *q, G, k* output: *kNN(t)*)
1. Insert the cell <*Cq*, 0> to MinHeap. 2. For four directions based on cell *Cq*, insert <DI*R0*, *mindist(DIR0, q)*> to MinHeap. 3. Remove the root from MinHeap. 3.1 If the root is a cell, store each object in the cell to *bestNN* and compute *best_dist* if necessary. 3.2 Otherwise, insert all the cells in *DIR(i)* and *DIR(i+1)* to MinHeap. 3.3 If the distance of the root in MinHeap is greater than *best_dist*, return. Otherwise go to the step 3.

Fig. 5 CPM-NN_computation Algorithm

In Fig. 5, Cq is the cell to contain the query point q, and *bestNN* is a priority queue that stores *k* objects. The initial value of *best_dist* is infinite and when *k* objects are stored in *bestNN* in the step 3.1, *best_dist* is the minimum radius that includes all objects in *bestNN*. In [6], every cell used in the step 3.1 of Fig. 5 were made to maintain the <*visit_list*> for every fixed query point. Thus, from hereafter, one may access the cell using the

<*visit_list*> instead of the MinHeap. Because this does not use additionally any heap data structure when the query point is fixed, the performance is enhanced. Also in [6], each query point maintains an influence region. This minimizes the calculation cost by investigating objects that enter the influence region and exit out of it. However in [6], for all moving query points only the NN_computation algorithm must be repeatedly used disregarding all influence regions or <*visit_list*>

III. Performance Analysis in a Gamming Environment

Through experiments, this paper analyzes the performance of the previously introduced three algorithms in a gaming environment; where all moving objects become the query point. For this experiment C++ was used to implement the three algorithms. Intel i7 CPU and a 8GB memory computer with Windows 7 was used for this experiment. The parameters applied to the experiment are like the following.

TABLE I
Parameters Used in Experiments

Parameter	Range
Number of Objects (n)	100, 500, 1000, 5000, 10000
Number of query Points	same to number of objects
Number of Neighbors(k)	2, 4, 8, 16, 32, 64
Velocity of Objects(v)	slow, normal, fast

The size of the space used for this experiment was fixed to a 1000x1000 pixel area. The speed of the objects were set as slow, normal, and fast as shown in Table I. The speed of slow, normal, and fast were defined as each 4 pixels, 20 pixels, 100 pixels per frame. The experiment data was generated by creating random numbers for the 2 dimensional space for each object. These objects were then programmed to move to a certain direction in a defined speed and fixed time interval. In the performance analysis, overhaul method, incremental method and CPM were named as YOH, YIN, CPM respectively. In every experiment each algorithm was repeated 100 times, and the running time was measured by seconds as the unit of measurement. For instance, if YOH is 76, it indicates the sum of time elapsed in a certain environment where YOH was repeated 100 times and its unit is also seconds.

The first experiment is the performance comparison according to the change in number of objects. Thus, we will measure the change in performance according to the change of the number of objects such as shown in Table I. Here, the number of neighboring agents was fixed to 16 and the speed of the objects was fixed to slow and normal. Fig. 6 illustrates the results of the experiment. In cases where the value of n is small, the incremental method shows the greatest performance. However, rapid decrease of performance was observed as the number of objects increased. In general, YIN has worse performance than the other algorithms when the speed of objects is fast [4], [6]. It is because the number of objects contained in the circle with the radius $|kNN(t\text{-}1)\text{-}q|$ centered at q increases. This is the same reason why YIN has bad performance when the number of objects increases. That is, if the number of objects increases in the fixed size of the space, there will be more objects per area and the number of objects that can be calculated through the step 2 of Fig. 3 will surpass greatly the number of k. The performance of the overhaul and CPM has the difference when the number of objects is 2500. As shown in Fig. 6, when the number of objects is less than 2500, CPM performs better, whereas the overhaul method outperforms when the number of objects increases.

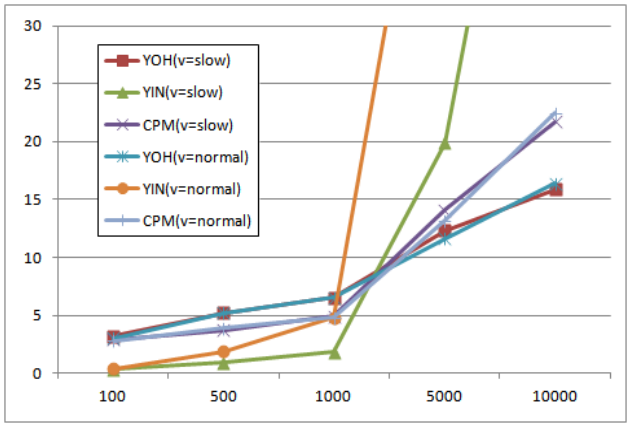

Fig. 6 Performance comparison versus n with k=16 and v= slow, normal

TABLE II
Performance Comparisons Versus k with n=1000, 5000 And v=normal

k	n=1000			n=5000		
	YOH	YIN	CPM	YOH	YIN	CPM
2	1.1	1.5	1.4	2.9	22.3	5.4
4	2.3	2.0	2.1	4.3	30.9	7.2
8	3.8	3.1	3.2	7.1	42.4	9.5
16	6.5	4.8	4.8	11.6	58.8	13.2
32	11.2	7.2	7.9	20.2	143.6	18.5
64	18.8	10.9	14.0	35.0	176.3	34.0

The second experiment is the performance comparison according to the change in number of neighboring objects. The number of objects was fixed as 1000 and 5000, and the speed of objects was fixed to normal. The results of the experiment are illustrated in Table II. When the number of objects is 1000 and the value of k is small, the performance of all three methods is similar. As the value of k increases, the incremental method displays the best performance, followed by the CPM. When the number of objects is 5000, the incremental method shows the worst performance. As mentioned in the previous analysis, this is because as the number of objects per area increases, more objects detected in the step 2 of Fig. 3. When the number of objects is 5000 and as k increases, the performance of CPM and YOH is similar.

The last experiment is the performance comparison according to the change in speed of objects. As shown in Table I, the speed of objects was divided into slow, normal, and fast. Fig. 7 explains the results of this experiment, where the value of k is fixed as 16 and the values of n are fixed to 100 and 1,000. As one can know from the characteristics of the algorithms, YOH and CPM are not related to the speed of objects. Thus, as shown in Fig. 7, YOH and CPM have constant values. On the

other hand, YIN illustrates a rapid decrease in performance as the speed of the objects rises. This is because the value of $|kNN(t-1)-q|$ in Fig. 3 increases. However, in the case that the value of n is 100, notice that even if the speed of objects is fast, the running time of YIN also is constant. It means that YIN has always the greatest performance when the number of objects is low.

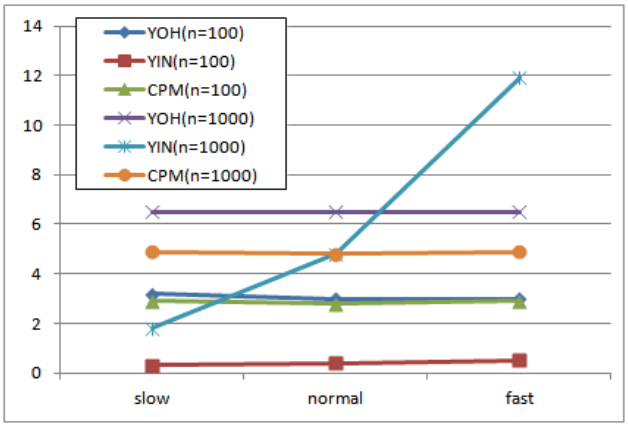

Fig. 7 Performance comparison versus v with k=16 and n=100, 1,000

Analyzing the results of the previous three experiments, the YIN method shows greatest performance when there is less number of objects. However, when the number of objects is high, the overhaul method displays best performance. This partially contradicts the results in [6]. It is possibly due to the difference in the number of query points. In gaming environments, the number of moving objects is not infinitely increased. Also, in gaming environments, the calculation for the movement of the objects is not only time consuming, but also it also is very time consuming for the rendering of each object. Thus, there are not many cases where over 1,000 objects are applied. Therefore in gaming environments, the incremental method is most applicable when the objects are slow. If not, the CPM method and the YOH method are applicable.

IV. Conclusions

Finding the k-nearest neighbor problem is studied throughout various fields of researches. Location based services are a representative example. Focusing on these services, many studies have been conducted. Although most of these cases are researched for moving objects, query points of them are mostly fixed, not moving. Thus, it is a different environment from gaming environments such as flocking, crowd simulation, and robot where query points are also moving. For the algorithms which are originally developed to suit location based services, this paper showed which algorithm is best applicable to gaming environments. The experimental analysis showed that the YPK-incremental method is most appropriate in cases where the speed of objects is slow or the number of objects is low, and the CPM method is more applicable when the moving object is fast. Finally, YPK-overhaul method outperforms rather than the others when the number of objects is high.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A2001091).

References

[1] Guttman, A., "R-trees: a dynamic index structure for spatial searching", ACM SIGMOD Rec., 14(2), 1984, pp. 47-57.
[2] Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model", SIGGRAPH, 21(4), 1987, pp. 25-34.
[3] Mat Buckland, "Programming Game AI by Example", ISBN 1556220782, Wordware Publications, 2005.
[4] Yu, X., Pu, K.Q., Koudas, N., "Monitoring k-Nearest Neighbor Queries over Moving Objects", 21st Int. Conf. on Data Engineering, 2005, pp. 631-642.
[5] Xiong, X., Mokbel, M.F., Aref, W.G., "SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-Temporal Databases. 21st Int. Conf. on Data Engineering, 2005, pp. 643-654.
[6] Mouratidis, K., Hadjieleftheriou, M., Papadias, D., "Conceptual Partitioning: an Efficient Method for Continuous Nearest Neighbor Monitoring", Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 634-645.
[7] Junglas, I.A., Watson, R.T., "Location-based services", Commun. ACM, 51(3), 2008, pp. 65-69.
[8] Jae Moon Lee, "An Efficient Algorithm to Find k-Nearest Neighbors in Flocking Behavior", Information Processing Letters, 110, 2010, pp. 576-579.
[9] Jun Min Park, Jae Moon Lee, "Performance Analysis for finding continuous k-nearest neighbors of moving objects under game environments", Proc. of Korea Game Society, Spring, 2013.
[10] Jae Moon Lee, Young Mee Kwon, "A Model of Pursuing Energy of Predator in Single Predator-Prey Environment", Journal of Korea Game Society, 13(1), 2013, pp. 41-48.
[11] Sangchul Kim and Zhong Yong Che, "A Method for Tennis Swing Recognition Using Accelerator Sensors on a Smartphone", Journal of Korea Game Society, 13(2), 2013, pp. 29-38.
[12] Seongdong Kim, Jae Moon Lee, Varun Ramachandran and Seongah Chin, "A biological simulation game using Prey-Predator model", Information - An international interdisciplinary journal, 16(4), 2013, pp.2607-2618.

Jae Moon Lee received M.S. (1988) and Ph.D.(1992) degree from the Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea. He is currently a professor at department of multimedia engineering, Hansung University, Seoul, Republic of Korea. He has published several research papers in the areas of flocking systems, evolutionary computation, machine learning and AI.