Journal of Computer Science

Original Research Paper

Voronoi Partition to Support Data Search in Uncertain Database with *k*-Bound Filtering

Slamet Sudaryanto Nurhendratno, Sudaryanto and Solichul Huda

Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia

Article history
Received: 11-09-2020
Revised: 21-12-2020
Accepted: 21-12-2020

Corresponding Author:
Slamet Sudaryanto Nurhendratno
Faculty of Computer Science,
Dian Nuswantoro University,
Semarang, Indonesia
Email: slametalica301@dsn.dinus.ac.id

Abstract: With the development of mobile technology, one of its main functions is to have mobile navigation capabilities, this is one of the Location-Based Services (LBS). Mobile navigation is designed as a mobile device to be able to monitor access objects from users. Because of the mobile characteristic, the impact on the points of objects to be found always occur a not certain change. Efficient query processing is needed on a set of mobile data due to the movement of users. This movement has an impact on searching in an uncertain database. For this uncertain database, the important query method is Probabilistic k-Nearest Neighbor query (PkNN), which calculates the probability of the set of k objects to be closest to the given query point. Several studies have been conducted at this time in order to contribute to the search results in a mobile database and uncertain with the support of certain algorithms to produce better performance. In this study, we propose a method called voronoi partitioning to support searching in uncertain database (Partition threshold *k* Aggregate Nearest Neighbor query method-Partition_PANN). In designing the partition of the voronoi region, it starts by using voronoi local networks to calculate query answers for small areas around the request point. A query point that meets the threshold used as a value as a set of threshold queries. So, all of the query points that meet the thresholds are the basis for forming local network voronoi partitions. Thus, this method does not require preliminary calculations also evaluation of distances at each intersection. The purpose of this research is to improve the performance of queries in an uncertain database by making the aggregate process and trimming the probability value as one phase of the search algorithm. In the first stage, objects which not able to form the answers are filtered by calculating the minimum closed circle from the dataset of query which prepared for the trimming phase. The second step called probabilistic candidate selection, its cut significant set of candidates for inspected as the aggregate function of the nearest neighbor's demand. The remaining set is sent for verification which obtains possible answers at the lower and upper limits so that candidates whose probabilities are not less than the user-specified threshold are saved in the result set and returned to the user. We also examine the efficient data structures spatially which support this method. Our solution can be applied to uncertain data with an ever-changing probability density function.

Keywords: PkNN, Voronoi Partitions, Aggregate, Uncertain Database

Introduction

The advanced in implementing k-NN queries on spatial data objects are the implementation of mobile navigation in mobile devices such as the Global Positioning System (GPS). The use of global positioning system technology is one example of superior technology mobile devices that have the power of Location-Based Services (LBS) (Xuan *et al*., 2008). This technology has the advantage to continuously monitor the point or object of user, even though the mobile devices always experiencing movement both statically

and dynamically. Location-based services and cellular internet will be able to monitor the movement of object points due to changes that occur, it is commonly referred to as split nodes (Zhou *et al*., 2018). Many approaches have been developed to design the processing of split nodes, for example by applying conceptual models such as multidimensional indexes and query processing, while the type of approach which is often used for the spatial query is the *k* Nearest Neighbor search method (kNN search) (Kolahdouzan and Shahabi, 2004). The method of searching for objects in spatial data is continuing developing until today, the development of the search method is more on improving or the upgrading of the algorithm. Typical algorithms from the Nearest Neighbor (NN) query are at depth of the traversal and the selection of the best traversal is based on the Root tree (*R* tree). To develop methods demanding query requirements under different conditions, in recent years, research on NN queries has expanded to kNN queries (Papadias *et al*., 2004), group nearest neighbor query (Sultana *et al*., 2014), scalable nearest neighbor query (Meyerhenke *et al*., 2014), reverse nearest neighbor query (Sack and Urrutia, 2000), continuous reverse *k* nearest neighbors query, aggregate nearest neighbor query, strong neighborhood pair query, visible nearest neighbor query, nearest neighbor query on uncertain data and so on (Elmongui *et al*., 2013).

Research on the methods that have been implemented to help in the process of searching for certain spatial data objects, especially for searches on mobile data that is uncertain is very rapidly developing and is mostly based on the k-Nearest Neighbor (kNN) method. To facilitate the search for the Nearest Neighbor (kNN) with respect to moving demand points, research in this area focuses mostly on techniques that utilize pre-calculated network distances as partial results. One common example of this technique is the Network Voronoi Diagram (NVD). Thus, the query with the method k-Nearest Neighbor (kNN) is one of the most popular types of queries in location-based search services (Meyerhenke *et al*., 2008). The searching process in kNN starts from the user making a query or query kNN to the service provider for the object or objects closest to the query point to the user's location. With the advances in spatial databases, kNN queries have been upgraded from Euclidean space to road network environments, where users can use kNN queries to find objects based on network distance. Although the results of the kNN query have resulted in the closest distance and travel time, the method is still being developed and improved as in increasing the closest range so that a new method appears, namely *k*-Range Nearest Neighbor (kRNN) The main idea of kRNN is to find k-nearest objects by pulling each point on the road segment in the search for the area in search by users who are always on the move. This user movement has an impact on searching in an uncertain database. One important search method in an uncertain database is the *k*-Nearest-Neighbor query (PTk-ANN) probabilistic threshold method, the method calculates the probability of the set of *k* objects to be closest to the given query point. Some methods which developed in handling aspects of changing nodes or split nodes as the uncertainty of data are the Voronoi Continous *k*-Nearest Neighbor (VCkNN). There are also those who develop a search method that emphasizes the probability of object *k* to be searched as the closest object to a given query point such as the Probabilistic *k*-Nearest-Neighbor Query (k-PNN) method, Probabilistic Voronoi Diagram (PVD), Probabilistic Moving Nearest Neighbor (P-MNN) and others. Query methods with various approaches to uncertain data still have problems in search performance, namely the involvement of thresholds as a set of queries. So, it requires access to all data objects and network nodes, which means it is not suitable for large data sets in many real-life situations. The best method available for monitoring kNN results without initial preparation depends on executing snapshot requests on network nodes encountered by the query point. This method results in the repeated evaluation of distances on the same or similar set of nodes. So the query method cannot be applied to solve the KANN query problem on uncertain data directly. Therefore, we propose to use voronoi partitioned to support searching in uncertain database.

The voronoi diagram is an important branch of computational geometry, some important properties, such as the property of close neighbors, the largest empty circle property, the control span property, have attracted much attention by practical production and scientific research fields. For complex datasets, the voronoi diagram plays an important role in calculating the nearest neighboring point, largest empty circle, convex *n* point, minimum tree and other problems. The voronoi diagram and the delaunay triangulation double graph have also been widely applied to the reconstruction of geometric models, remote sensing satellites and others. The voronoi diagrams can express adjacent lateral relations of spatial data information and also have the basic character of vector and data models which uncertain and constantly changing continuously. Thus, this is an important part of the study of spatial data objects, especially in the field of GIS (Li-Ping *et al*., 2014).

In this study, the authors propose to expand the object search techniques of the kNN method, such as Probabilistic Aggregate Nearest Neighbor (PANN) query. This expansion will support implementations that have wide application services such as location-based services. Expansion of the search techniques by calculating the aggregate distance between the data

Points (P) and the Query set (Q) from each query result so that there will be three new functions (aggregate, sum, max and min) for each point. This aggregate value will be used as a collection of values in forming partitions in a particular region in the form of a voronoi diagram, where the query results for the nearest neighbor aggregate request depend on these different aggregate functions. Whereas data points that have the minimum aggregate distance from the data set *Q* the query is returned because they do not qualify as objects in the search in a particular Voronoi region partition. This is the background of my proposed search method (Partition_PANN) in searching for uncertain data by partitioning objects in certain areas to improve the performance of searching data in uncertain databases such as location-based services.

Related Work

Based on the literature that explained by (Papadias *et al*., 2004), the first research about ANN was conducted by (Papadias *et al*., 2005). There are three kinds of proposed KNN algorithm related to the environment and road network, namely Incremental Euclidean Restriction (IER), Threshold Algorithm (TA) and Concurrent Expansion (CE). It was explained that the IER algorithm utilizes the R tree and the *A** algorithm. Whereas the TA and CE algorithms are implemented by applying the *A** algorithm combined with a threshold to solve the ANN problem. A few years later research related to ANN was developed by (Zhu *et al*., 2010) by utilizing adjacent properties and pre-computational advantages from the voronoi diagram; the research proposes the kANN algorithm on the environment and the road network. In the algorithm described two important phases, namely the query phase and the trimming phase. The research also discusses strategies in expanding query points with the aim of effectively increasing query efficiency (Elmongui *et al*., 2013). Even his research also discusses the problem of spatio-temporal data flow management systems, such as the Continuous Aggregate Nearest Neighbor (CANN) query for moving and uncertain objects.

Several uncertain spatial database prototypes have also been developed nowadays (Spooner *et al*., 2004). In the study, two main classes of uncertainty models were assumed: Tuple uncertainty and attribute uncertainty. Where the uncertainty of the tuple recording the probability that the given tuple is part of a relationship (Park *et al*., 2015). While attribute uncertainty can represent inaccurate attribute values as uncertainty areas and pdfs which restricted in the region (Clark and Evans, 1955). The result of this research is the proposal of a formal prototype database to combine tuple uncertainty and attribute uncertainty (Tao *et al*., 2002).

Several studies related to evaluating Probabilistic Nearest-Neighbor (PNN) query on attribute uncertainty have also been conducted. PNN, which can be considered as 1-PNN, returns the probability of a single object to be closest to the given query point *q*. Wang *et al*. (2015) research, *R* tree-based indexing solutions for PNN have been presented. In this study, we developed an indexing solution (called k-bound filtering) on k-PNN.

Zhang *et al*. (2010), the probability of qualifying an object to meet PNN is obtained by changing the uncertainty of each object into two functions: Pdf and cdf the object's distance from the query point. They show how this conversion can be done for 1 D uncertainty (intervals) and 2 D uncertainty (circles and lines). The probability of qualification is then derived by evaluating the integral expression involving the pdf distance and cdf of several objects. We show how the k-PNN calculation can be done using distance pdf and cdf. Another method for evaluating PNN is proposed in (Sun *et al*., 2013), where each object is represented as a collection of points taken from continuous object pdfs. Recently, that the probability of an object in the database (called existential probability) is used to obtain lower and upper limits and pruning for the nearest neighbor (Wang *et al*., 2015). In the study, the author discusses how to efficiently retrieve data objects that have a minimum aggregate distance from a set point query.

To increase the evaluation of the probability of qualification for 1-PNN, have been proposed a 1-PNN variant that uses the probability threshold as an answering criterion and has developed an efficient verification method to obtain lower and upper limits of object qualification probabilities. These methods are not easy to use by k-PNN (with $k \geq 1$) for three reasons. First, the k-PNN evaluation faces an additional problem in examining a large number of k-part subsets. To deal with this problem, we developed a new method to reduce the number of subscriber candidates significantly. Second, probability bound verification is designed only for 1-PNN queries. We developed a new lower and upper bound calculation method for k-PNN queries. Third, the solution can only be used to handle pdf distances from candidate objects that are represented as random histograms (Nurhendratno *et al*., 2018). On the other hand, the author's technique is not limited to pdf (probability density function) histograms.

As far as our knowledge, there is little work that answers k-NN uncertain data questions. The main process of the k-NN algorithm is to submit a request that ranks the probability that each object is the closest neighbor to *q* and returns *k* object with the highest probability. Note that the ranking criteria are only based on the probability of each nearest neighbor object of *q*. This is not the probability that all objects returned in the query answer are the closest neighbor of k-query (kq). In

other words, the object k returned by (Sun *et al*., 2015) might not appear in a world that might be the same. On the other hand, the query studied in this study is "True" k-query with the nearest neighbor, where we consider the probability that the set of objects is *k* nearest neighbor *q*. In the discussion, (Sun *et al*., 2015) proposes an efficient index structure, called APLA-tree, to evaluate k-NN requests. They use the expected distance (under L1-norm) from the pdf uncertainty object of *q* as a ranking criterion. Thus, their k-NN requests are based on expected distances and have no probability in their answers.

Evaluation and indexing methods for probabilistic questions attribute uncertainty has been studied. This includes a range of queries, location-dependent queries, skyline queries and top-k queries. The issue of uncertainty has also been considered in the domain of biometric databases and access control. Recently, the problem of conditioning and cleaning probabilistic databases has been studied by many studies (Sudaryanto *et al*., 2019).

The methods we mentioned above are all for certain data, however, large data that is uncertain is processed in a real application. Examines GNN requests for uncertain data and proposes spatial pruning methods and probabilistic pruning methods to trim data points and improve query efficiency. According to the data which is uncertain, researches methods for resolving various requests, nearest neighbor requests and the most regular neighbor query problems. The above research is all done to query the uncertain conditions of the dataset (Chen *et al*., 2013).

Problem Definition

Spatial databases with attribute uncertainty can be divided into two groups, namely existential uncertainty and location uncertainty. Existential uncertainty can represent a certain point of location, but existence is uncertain. Location uncertainty where the data point is certain, but the location is uncertain. All of these possibilities will always be found within a certain probability scale.

In this study, we study Probabilistic k-Nearest Neighbor (k-PNN) Query for databases with attribute uncertainties. This query returns non-zero probabilities (called qualification probability) from each set of *k* objects to be the closest neighbors of a certain point *q*. Given an uncertain D database of *n* uncertain objects, where $D = \{o1,..., on\}$, k-PNN queries can be defined as follows.

The algorithm we propose uses the voronoi diagram as the basic framework. Voronoi diagrams have many geometrical properties that can greatly improve the performance of a range search query processing in a particular partition. Voronoi diagram is the decomposition of space and fields according to the position of a set of discrete points (sites). Each site produces Voronoi Polygon (VP) which involves all points closer to the site than the others. VP is formed by a set of Voronoi edges which is a subset of locus points that are the same distance from two adjacent sites. The intersection of these edges for a site is called voronoi vertex. The Voronoi diagram definition is.

Definition

Given a set of discrete objects $\wp = (P_1,..., P_n)$ $(n > 1)$ in the road network, $WP(P_i) = \{\forall hal \mid d_n(hal, P_{saya}) d_n(hal, P_j)\}$ $(i, j \in i_n$ dan $i = j)$. Voronoi diagram for $\wp$ is:

$$VD(\wp) = \sum_{k=1}^{n} VP(Pi)$$

This definition shows that for each point in the $VP(P_i)$, the distance to P_i must be smaller than other generators, then $VP(P_i)$ is called the voronoi polygon which is related to P_i.

NVD is a special type of voronoi diagram that is built on a spatial network with composition as a spirit to partition. Decomposition as the basis of this partition is based on the connection of nodes or discrete objects rather than Euclidean distances. In NVD, the voronoi polygon transforms into a set of road segments called Network Voronoi Polygon (NVP) and the polygon edge also shrinks to several midpoints, called border points, from the road network connection between two interesting objects.

Figure 1 shows an example of NVD partitions. Besides interesting objects (*P*), NVD also includes several road network intersections (*n*) and border point (b). According to the voronoi diagram property, from the border point to a pair of adjacent objects the same distance (e.g., $Dis(b_7, P_1) = dis(b_7, P_3)$), then we only need to use the Dijkstra algorithm in one voronoi polygon to get the distance from the generator to its border. The distance between objects can be calculated by selecting the minimum distance to shared boundary and multiplying this value (e.g., MIN $(dis(P_3, P_1)) = 2*dis(b_7, P_3) = 2*dis(b_7, P_1)$).

Since all distances from the border point to the generator or other border points can be calculated using the Dijkstra algorithm and stored in a database, when a range query is issued, all the required distances can be taken from the database rather than calculated online. So, with the help of NVD, VRS and VCR can process range queries more efficiently than traditional range search methods.

DOI: 10.3844/jcssp.2020.1753.1764

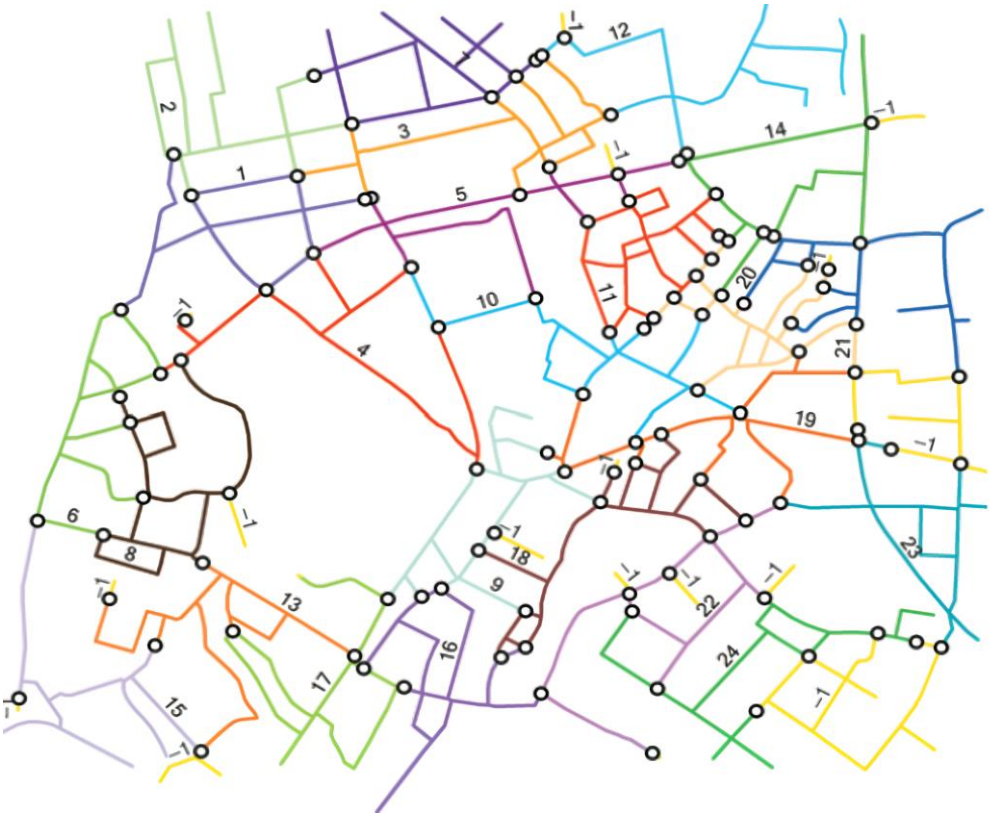

Fig. 1: NVD partitions into the same cell (sub-trees of the same total length except for boundary cells (indicated by cell number-1); circles on the network indicate boundary points)

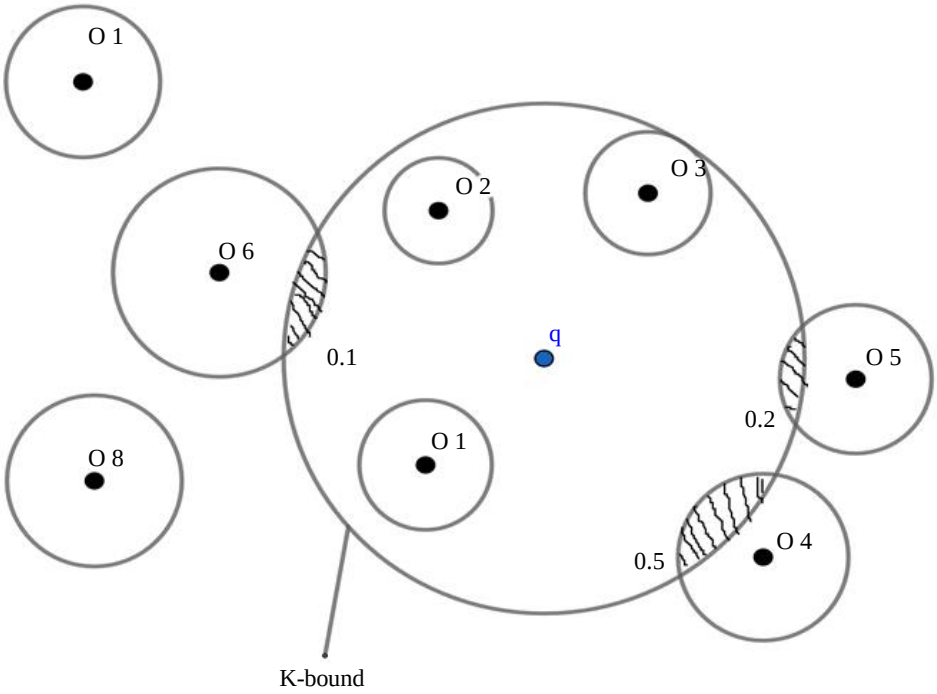

Fig. 2: Voronoi partition former threshold

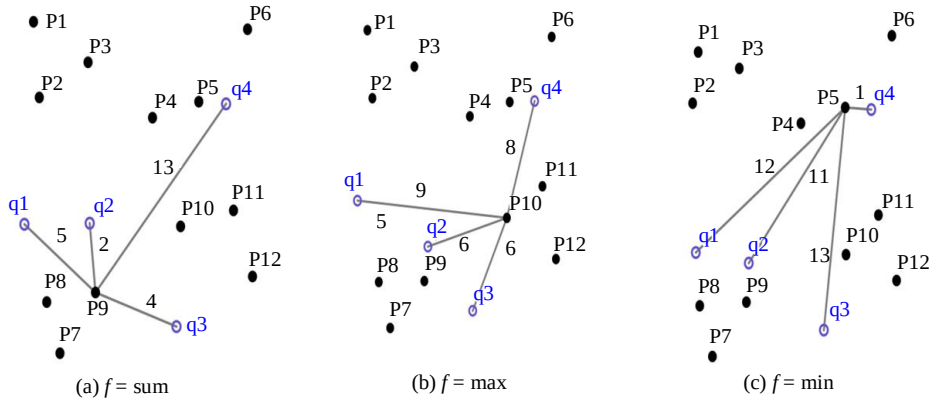

Fig. 3: Sample *k* Aggregate Nearest Neighbor (kANN) queries in object spatial databases

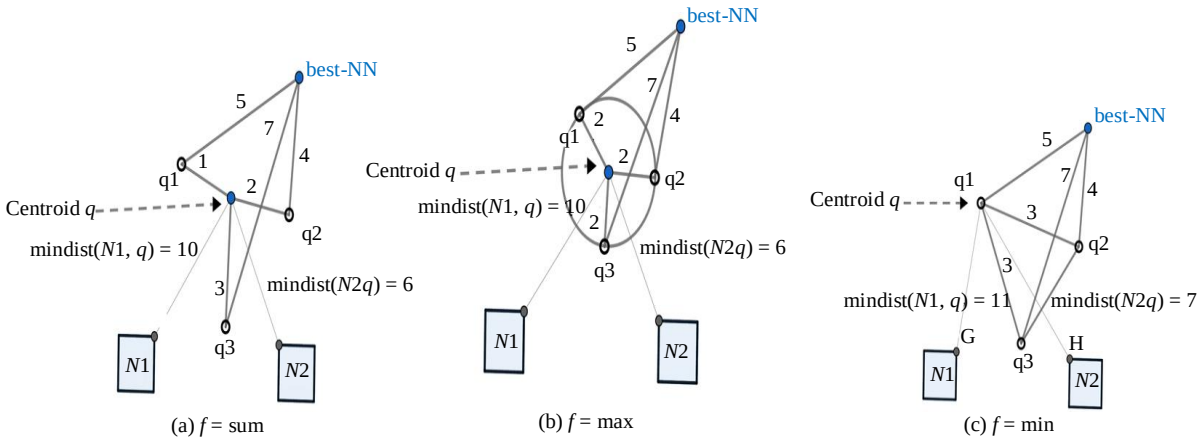

Fig. 4: After pruning of node in partitioned spatial database

Voronoi-Partition Design to Support Searching in Unceratain Database

The k-aggregate probabilistic threshold for closest neighbor query problems includes three phases. In section pre processing phase, we propose a grouping algorithm, to process the partitioning of data from a data set into a specific data group. Query algorithm by searching for partitions to minimize the average distance from minimum closed circles and single-point cluster requests. At the processing phase we propose an algorithm for process the effects of the results set by the dataset variations. In section filtering phase, for the three types of aggregate functions, we propose related pruning strategies to cut points that cannot be results and add qualified points to the candidate set. At last, in section refinement phase, we propose a phase improvement algorithm. For data points in the candidate set, calculate the probability to be a query result and add a set whose probability is greater than the threshold into the result set. Figure 2 illustrates if there is a distribution of *n* points on a two-dimensional plane, as a collection of point objects $P = \{p_i, ..., p_n\}$ and there are *n* number of query points $Q = \{q_i, ..., q_n\}$, then the query results can be calculated through the aggregate distance between the data point *p* and the query set *Q*. Thus there will be three sums of the aggregate function, sum, max and min of each point

Pre Processing Phase

At the pre-processing stage illustrated in Fig. 2, the data is processed from the data acquisition into a dataset. In the dataset, data normalization is performed then the Voronoi diagram will be made. From the normalization of these data, it was done by eliminating data uncertainty which is not needed in this study. Phase in normalizing include the process of grouping data into groups, ranging from insignificant data to data that is significant to the threshold value. This process repeats itself at each point to determine centroid polygons, creates polygon voronoi and generates datasets in voronoi diagrams. The results of normalization can be continued to the processing phase and used by the Partition_MinCircle algorithm to search.

Processing Phase

The difficulty of ANN is how to group the closest neighboring points to be partitioned from several demand points in the case of different aggregate functions. In this study, we propose the Partition_MinCircle (*Q*) algorithm to calculate the minimum closed circle of a group of query points. By using the center of circle *O* to represent the distribution of query points on several partition groups. In conducting partitioning, the data collected will be clustered in *k* data groups. Each *k* data group and is used to be the center of the query points on that partition:

Algorithm - Partition_MinCircle

Input: A dataset of point Q = {*q*1, *q*2,…*qn*}
Output: Min_cover circle centers *{c1…ck}* implicitly dividing *X* into *k cluster/*center of minimum covered circle*/*
choose initial MinCircle (Q) ← ∅/*minimum covered circle*/
choose initial (O) ← ∅/* center of minimum covered circle*/
choose initial (d) ← ∅/* diameter of minimum covered circle*/
take two-point *qi*, *qj* from *Q*
d ← dist(*q*1, *qj*)
O ← {*c*1,…*ck*} dividing *Q* into *k*
Make the circle *C*1 which is centered at *O*1 and has a diameter *d*1
Remove *qi*, *qj* from *Q*
MinCircle (*Q*) ← C1
O ← *O*1
For I = 1 to N do
    find the closest center *Ck* ϵ *C* to instance *xi*
    for each q ϵ *Q*
        d ← dist(*qn*,*O*) + *d*/2
        O ← On
        MinCircle(*Q*) ← *Ck*
    End for
    assign instance *xi* to set *Ck*
    for *i* = 1 to *k* do
        set *ci* to be the center of mass of all points in *Ci*
    end for
end for
Return *O*;
End.

In the algorithm model above, the first step is to take two points from the first *Q* query dataset and calculate the diameter distance of each circle *C*1. Let the *O*1 shows the center of the circle. Delete *qi* and *qj* from the query data set, the currently minimum closed circle and its center are assigned to MinCircle (*Q*) dan *O*, respectively. In the while-loop, the algorithm assesses whether there is an exit *q* not inside or in the MinCircle circle (*Q*). If *q* does not exist, the MinCircle (*Q*) is the minimum closed circle that we calculated. Return *O* and algorithm are terminated. If *q* exists, then run the loop statement. In the of the loop statement, first of all access to the *qm* point which has the longest distance from the center of circle *O* that we have calculated. Then calculate the distance between *qm* and *O* then add the current minimum closed circle radius and this distance is set to diameter *d*. Create a *Cm* circle that is centered on *Om* and has a diameter *d*. The currently minimum closed circle and its center are set to MinCircle (*Q*) and *O* and this loop ends. Repeat this procedure until the for-loop cycle conditions are incorrect and then the algorithm is terminated. Return the center of circle *O*.

Filtering Phase

The filtering phase is part of the aggregation implementation by trimming the points in certain criteria. The pruning strategy is related to trimming the points that can't be a payoff and adding the points that qualify into the candidate pool. This step in filtering starts from the processing phase of calculating the nearest neighboring points of several demand points in the case of different aggregate functions. The aggregate step is an effort to reduce computational complexity in the query process on ANN. Next is a refinement phase for the data points in the candidate set, calculating the probabilities to be the query result and adding the set whose probability is greater than the threshold to the result set.

Aggregate Function f = Sum

When the aggregate function is summed, data points *k* with the minimum total distance to the query data set are returned. Centroid calculations are performed in the initialization phase to filter out data points that cannot be answered. In this sum function, *q* minimize the function a $\text{dist}(q,Q)=\sum_{i=1}^{n}|qqi|$, for example (*x*, *y*) become *q* coordinate and (*xi*, *yi*) become qi query point coordinate. Because the partial derivative of the a dist function (*q*, *Q*) concerning the independent variables *x* and *y* is zero at *q* centroid, we have the following equation for the basic algorithm as follows:

$$\frac{\partial adist(q,Q)}{\partial x}=\sum_{i=1}^{n}x=\frac{(x-xi)}{\sqrt{(x-xi)^2+(y-yi)^2}}=0$$

$$\frac{\partial adist(q,Q)}{\partial y}=\sum_{i=1}^{n}x=\frac{(y-yi)}{\sqrt{(x-xi)^2+(y-yi)^2}}=0$$

By modifying geometric centroid coordinates in order to get a fast approach:

$$x = x - n\frac{\partial asist(q-Q)}{ax} whiley = y - n\frac{\partial adist(q-Q)}{\partial x}$$

where, *n* is the number or number of steps, with the number of processes repeated until the adist function (*q*, *Q*) is closed to the minimum value. For the same technique with different initial geometric centroid coordinates to capture weights, i.e.:

$$x = \left(\frac{1}{\sum_i^n wi}\right).\sum_{i=1}^n wi.xi\ while$$

$$y = \left(\frac{1}{\sum_{j=i}^n wi}\right).\sum_{i=1}^n wi.yi$$

Aggregate Function f = Max

When the aggregate function is max, the data point k with the minimum-maximum distance to the query data set is returned. For max function, centroid (minimalize a dist function $(q,Q) = \max_{i=1}^n |qqi|$) corresponds to the center of the smallest circle containing all the points in *Q*. This is also known as the minimum closing loop problem, where various algorithms get the right answer. In our implementation, we use a random incremental algorithm with expected linear time (to several query points). So, for weighted max, we can use the same centroid as in the unweighted case.

Aggregate Function f = Min

When the aggregate function is min, *k* data points that have a minimum distance to the query dataset are obtained. The min aggregate function is easier than the other two, so we use the method as considering centroid as a query point to get query results. By considering the min aggregate function, one of the query points can be chosen as centroid because it points to adist $(q,Q) = \max_{i=1}^n |qqi| = 0$. Among the alternatives, we choose the one that minimizes $\max_{i=1}^n |qiqj|$ (i.e., query points with the lowest maximum distance from any point in Q). The reason for this choice will become clear after the trimming of the trimming strategy. In a weighted min scenario, among *n* possible choices for *q*, we choose the query point *qi* with the maximum weight *wi*.

Algorithm - Filltering_Min_Candidate

Input: set of query point $Q = \{q1,q2,\ldots qn\}$, uncertain dataset $p = \{p1,p2,\ldots pn\}$

Output: node min_candidate circle centers *{c1…ck}, Partition_PANN query candidate case of f = sum, f = main, f = max*

Begin;

Sum_candidate ← ∅/*initialize the query candidate set*/
Max_candidate ← ∅/*initialize the query candidate set*/
Min_candidate ← ∅/*initialize the query candidate set*/
H ← ø, Can_visit ← ∅/*initialize the queu*/
Visit ← ø
q ← min_circle (Q)/*process the query dtaset*/
r ← the location of *q* in the uncertain Voronoi diagram
if *r* ϵ SUV(*pi*)
    Can_visit ← *pi* end its *k* level generation points
    P_ANN ← *pi* the single generation points *p*
    Min_candidate ← *pi* end its *k* level generation points
Else
    P_ANN ← any of the multiple generations on point
    Can_visit ← generation point set of MUV(*Pi*, *Pj*) where *q* is located in *k* level generation points
    Min_candidate ← generation point set of MUV(*Pi*, *Pj*) where *q* is locate in *k* level generation points
    Generation point
End if
For each *C*1 ϵ Can_visit do
    If (count (*C*, *q*) ≤ *k*)
        dist_min(*q*, *Ci*) ← dist(*q*, *Ci*)-*ri*
        dist_max(*q*, *Ci*) ← dist(*q*, *Ci*)+*ri*
    else
        prune *Ci*
        max_candidate ← *pi*
        min_candidate ← generation point set of MUV (*Pi*, *Pj*) where *q* is located in *k* level generation points
    end if
    return min_candidate
    return max_candidate
    return sum_candidate
end for
end;

How the voronoi partition algorithm work starts with initializing the candidate and assigning several variables and setting Min_Candidate values and calls the initial algorithm to get the center of the circle *q* from the minimum closed circle. Next is to determine the center point of the query *q*, to get an uncertain location in the Voronoi diagram. If point *q* is in SUV (*pi*), add a single generation point and *k* level of the adjacent generation point into the partition through the candidate set value Min_Candidate. If the *q* point is in the MUV partition (*pi*, *pj*), add some generation points and *k* generation point levels that are adjacent to the candidate set Min_Candidate. Min_Candidate filter algorithm ends with Min_Candidate as a candidate set, this process is repeated until there are no more search candidates.

For each aggregate function, in the refinement phase, we first initialize the results set by Partition_PANN_Result. Process the query data set according to the candidate filtering algorithm and obtain the *q* center from the query dataset. Candidates assigned by Partition_PANN_Candidate from kANN are obtained by the appropriate algorithm in the filtering phase. For data points in the candidate set, count all *s* sets composed of data points *k* and calculate probabilities probable (s) of them as a result set. Compare the probability with the user-specified *T* threshold and return the result {*s*, prob (s)} which the probability is greater than the threshold.

Refinement Phase

In the refinement phase, data points that cannot be query results based on the previous two phases will be truncated and get a candidate pair on the appropriate partition. After that, in the refinement phase, the probability of the data set *k* data points in the candidate set will be calculated and compare the probabilities with the user-defined Threshold (T). Finally, the location verification will be obtained in order to get results that are consistent with the determination of objects in the partition. After verification, the object is stored in set *S* and requires further processing, which has the proper qualification probability calculation. The main idea is to treat the probability of an object as the sum of the probability of qualifications in the partition. By using probability bonding information in each partition, the probability of an answer can be gradually calculated. This process is repeated for the next partition until we can decide whether *S* should be included in the answer. As shown in our experiment, "incremental improvement" usually faster than the probability of computing directly doing numerical integration on partitions faster than at [0, *fk*], which has a larger integration area.

Experimental Result

In this part of the experiment, we evaluate the effectiveness of the proposed Partition_PANN algorithm in this study. The uncertain data set (*P*) comes from a simulation data set and the *Q* query data set is randomly generated. This uncertain diagram will be indexed in the VR-Tree form. In the initial stages voronoi diagrams that are not certain will be built first. In conducting this experiment, the algorithm will evaluate the time of the query, the cost of I/O and the trimming ratio of three things. The experimental results are obtained from the average query results by executing 50 times.

In evaluating the effect of query time processing based on the size of the data points in the case of aggregate and max number functions. The abscissa represents the size of the data point and the ordinate represents the query time (Nurhendratno *et al*., 2018). It is shown from the numbers that keep the other query conditions unchanged, the query time of the three algorithms all increase with a gradual increase in data point size. Because more distance needs to be calculated when the size of the data point is large (Nurhendratno and Sudaryanto, 2017). The number of data points in the candidate set and the complexity of the calculation of the probability value also increases and leads to an increased inquiry time.

The computer used in this experiment is a computer with CPU Pentium I5 2.7 GHz, Windows 7, 4 GB memory and MySQL as a database server. The dataset in this experiment was synthesized using software produced by the spatial data generator. A dataset consisting of 6,000 objects are used. The capability in VR-tree is set in 3 K size per leaf node and the maximum capability of VR-tree node is 50 entries. For disk caching which related to memory buffers, the LRU algorithm is used. The memory buffer only stores pages related to the VR-tree root initially and can load up to 64 pages.

Experiment

In this study, we compare the algorithm that the author developed, namely Partition_PANN with the PTkANN algorithm and PANN algorithm, this algorithm utilizes the main idea (Papadias *et al*, 2005). Call the algorithm repeatedly in (Papadias *et al*., 2005) for *k* times to execute the PTkANN query. In this experiment, the measured time cost is the average cost of 100 queries with the same dataset but with different query points generated randomly. Our results are summarized in Table 1.

Table 1 shows that the query processing time of the Partition_PANN algorithm is less than the other two approaches. It is easy to see that query processing time is from Partition_PANN algorithm grows very slowly, especially when $n<4.000$ which unlike the other two approaches increases sharply as n, the size of the dataset, increases from Table 1. The experimental results show that the Partition_PANN algorithm for the 1 NN query outperforms the previous method in runtime. This advantage stands out, especially in large datasets. This advantage comes from the excellent VR-tree data structure is used which makes the role of Voronoi diagrams brought to the full processing time in 1 NN requests.

Experiment

For each *k* value, we perform 100 kNN queries with different query points but on the same randomly selected dataset consisting of 6,000 objects, using Partition_PANN, PANN and PTkANN, each with *k* varying from 2 to 200. We present an average execution time of 100 runs of the kNN query for each *k* value and each algorithm, as shown in Table 2.

From Table 2, it is easy to see that the PTkANN and PANN times are between 2,01 and 2,2 times as much as

the Partition_PANN algorithm. We can see that the execution time of Partition_PANN algorithm grows very slowly when *k* increases, *k*<20.

From the experiment it can be seen that the performance of the Partition_PANN algorithm will be better and simpler because *k* increases when k>40. However, the performance is much faster than the other two methods. This phenomenon arises because the candidate size specified by *AG1* (*p*) *U,...*, *UAGi* (*p*) which is used to search for (*i* + 1) - the nearest neighbor grows rapidly for a greater value of *I*, where *p* is the closest neighbor from the request point. The experimental results show that the Partition_PANN algorithm outperforms the previous method in processing time, especially for values smaller than *k* concerning the size of the dataset.

From Table 3, the pruning effect on *f* = sum that the execution time of the Partition_PANN algorithm is always better for than other two approaches. We can also see that the difference in execution time of Partition_PANN, PTkANN and PANN becomes wider as the size of the dataset increases (from 4 *k* to 1,024 k). The experimental results show that the Partition_PANN algorithm performs better than the previous method in execution time.

Experiment

We compare the performance of Partition_PANN with two other methods that can be applied in the case that the pruning path *f* = sum is a line segment. Therefore the pruning path which considered is limited to line segments in our experiment. In this experiment, using an average of 6% for each Query Area (QA), the length of the query line segment, we performed 40 queries with different query line segments on the same randomly selected dataset consisting of 100 nodes.

From Table 4, the effect of pruning on f = max is that the execution time of the Partition_PANN Algorithm is always better the other two approaches. We can also see that the difference in the execution time of Partition_PANN, PTkANN and PANN becomes wider as the dataset size increases (from 4k to 1,024k). The experimental results show that the Partition_PANN algorithm has better performance than the previous method in execution time.

Experiment

We compare the performance of Partition_PANN with two other methods that can be applied in the case that the pruning path *f* = max is a line segment. Therefore the pruning path which considered is limited to line segments in our experiment. In this experiment, using an average of 6% for each Query Area (AQ), the length of the query line segment, we performed 40 queries with different query line segments on the same randomly selected dataset consisting of 100 nodes.

From Table 5, the effect of pruning on *f* = min that the execution time of the Partition_PANN algorithm is always better than the other two approaches. We can also see that the difference in execution time of Partition_PANN, PTkANN and PANN becomes wider as the size of the dataset increases (from 4 k to 1,024 k). The experimental results show that the Partition_PANN algorithm performs better than the previous method in execution time.

Table 1: The execution time of Partition_PANN versus PTkANN and PANN for different data size

Time(sec)	1.000	2.000	3.000	4.000	5.000	6.000
PTkANN	0,0091	0,0109	0,0201	0,0211	0,0262	0,0298
PANN	0,0062	0,0081	0,0090	0,0103	0,0156	0,0177
Partition_PANN	0,0035	0,0039	0,0043	0,0080	0,0088	0,0098

Table 2: The execution time of Partition_PANN versus PTkANN and PANN for different *k* point

Time(sec)	4 k point	10 k point	20 k point	40 k point	80 k point	100 k point
PTkANN	0,0097	0,0111	0,0151	0,0201	0,0232	0,0268
PANN	0,0086	0,0104	0,0133	0,0160	0,0206	0,0254
Partition_PANN	0,0040	0,0051	0,0063	0,0103	0,0167	0,0191

Table 3: CPU cost by fixing A_Q to 6% of the data space and the number k of retrieved ANNs (*f* = sum)

Time(sec)	4 k	16 k	64 k	256 k	1.024 k
PTkANN	0,0139	0,0411	0,0610	0,0921	0,1210
PANN	0,0096	0,0181	0,0195	0,0600	0,0969
Partition_PANN	0,0044	0,0049	0,0054	0,0080	0,0108

Table 4: PU cost by fixing A_Q to 6% of the data space and the number k of retrieved ANNs (*f* = max)

Time(sec)	4 k	16 k	64 k	256 k	1.024 k
PTkANN	0,0189	0,0461	0,0677	0,0952	0,1265
PANN	0,0109	0,0192	0,0231	0,0620	0,0981
Partition_PANN	0,0051	0,0068	0,0075	0,0080	0,0113

Table 5: CPU cost by fixing A_Q to 6% of the data space and the number k of retrieved ANNs (f = min)

Time(sec)	4 k	16 k	64 k	256 k	1.024 k
PTkANN	0,0119	0,0128	0,0151	0,0192	0,0215
PANN	0,0088	0,0101	0,0135	0,0168	0,0196
Partition_PANN	0,0038	0,0044	0,0051	0,0078	0,0101

Experiment

We compare the performance of Partition_PANN with two other methods that can be applied in the case that the pruning path f = min is a line segment. Therefore the pruning path which considered is limited to line segments in our experiment. In this experiment, using an average of 6% for each Query Area (AQ), the length of the query line segment, we performed 40 queries with different query line segments on the same randomly selected dataset consisting of 100 nodes.

Conclusion

In this study, we research extensively about queries with the basic algorithm kNN and ANN using the Voronoi diagram. Because of the unpopular use of data in many searching applications, uncertainty management has become an important topic in the database community, so we are developing a new data structure. We began to study a useful basic query, i.e., k-NN Query (T-k-PNN) probability threshold for database uncertainty. Different from the right database, evaluating T-k-PNN requires probability information and doing expensive numerical integration. In this study, we propose a probabilistic calculation of the aggregate k algorithm of TkANN neighbor demand based on the voronoi partition. In the process of querying the nearest neighbor demand, this algorithm includes three phases. First, proceed the query dataset by calculating the minimum closed circle. Then, use the appropriate pruning strategy to process the dataset and get the candidate set. Therefore, we propose various pruning techniques by considering distance and probability constraints and grouping nodes in certain Voronoi partition groups. As a result, the method we named as Partition_PANN, the result of the experiment is compared with the 2 methods of PTkANN and PANN shown by our experimental results, with the k-bound filtering technique, many objects that do not meet the requirements can be trimmed. The number of k-subsets can be partially reduced significantly by the Partition_PANN algorithm. We can demonstrate efficient calculations for the lower and upper probability limits with the help of partition information. We will learn how this technique can be extended to support other queries, for example, reverse-neighbor queries. Based on experiments with a limited dataset, we have proved the efficiency of the algorithm which we propose in this study. In the future, we intend to examine the verification problem by expanding the query area to obstructed spaces in an uncertain database.

Acknowledgement

This research received full support and funding sponsorship from the Ministry of Research and Technology-Higher Education of the Republic of Indonesia. Planning, operational coordination and supervision are carried out by LL-DIKTI Region VI of Central Java. The authors express their gratitude for their attention and guidance.

Funding Information

The authors should acknowledge the funders of this manuscript and provide all necessary funding information.

Author's Contributions

Slamet Sudaryanto Nurhendratno: Is involved in the concept of developing a search method, testing and analyzing results.

Sudaryanto: Was involved in programming and comparison of results between other methods.

Ethics

Authors will be responsible for the ethics and originality of the work that has been published.

References

Chen, P., Gu, J., Lin, X., & Tan, R. (2013). Group Nearest Neighbor Queries over Uncertain Data in Location Based Services. International Journal of Hybrid Information Technology, 6(2), 117-128.

Clark, P. J., & Evans, F. C. (1954). Distance to nearest neighbor as a measure of spatial relationships in populations. Ecology, 35(4), 445-453.

Elmongui, H. G., Mokbel, M. F., & Aref, W. G. (2013). Continuous aggregate nearest neighbor queries. GeoInformatica, 17(1), 63-95.

Kolahdouzan, M., & Shahabi, C. (2004). Continuous k-nearest neighbor queries in spatial network databases. Proc. of STDBM.

Li-Ping, Z., Ji-qIao, Z., & Song, L. (2014). Research on methods of construction of Voronoi diagram and nearest neighbor query in constrained regions. Computer Science, 41(9), 220-224.

Meyerhenke, H., Monien, B., & Sauerwald, T. (2008, April). A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. In 2008 IEEE International Symposium on Parallel and Distributed Processing (pp. 1-13). IEEE.

Meyerhenke, H., Sanders, P., & Schulz, C. (2014, June). Partitioning complex networks via size-constrained clustering. In International Symposium on Experimental Algorithms (pp. 351-363). Springer, Cham.

Nurhendratno, S. S., & Budiman, F. (2017). Design model integration and syncronization between surveillance units to support data warehouse epidemiology. Journal of Theoretical & Applied Information Technology, 95(3).

Nurhendratno, S. S., Sudaryanto, Budiman, F., & Setyowati, M. (2018). Query Optimization on Distributed Database Dengue Fever by Minimizing Attribute Involvement. J. Comput. Sci., 14(4), 466-476.

Papadias, D., Shen, Q., Tao, Y., & Mouratidis, K. (2004, April). Group nearest neighbor queries. In Proceedings. 20th International Conference on Data Engineering (pp. 301-312). IEEE.

Papadias, D., Tao, Y., Mouratidis, K., & Hui, C. K. (2005). Aggregate nearest neighbor queries in spatial databases. ACM Transactions on Database Systems (TODS), 30(2), 529-576.

Park, J. H., Chung, C. W., & Kang, U. (2015). Reverse nearest neighbor search with a non-spatial aspect. Information Systems, 54, 92-112.

Sack, J. R., & Urrutia, J. (Eds.). (1999). Handbook of computational geometry. Elsevier.

Spooner, P. G., Lunt, I. D., & Briggs, S. V. (2004). Spatial analysis of anthropogenic disturbance regimes and roadside shrubs in a fragmented agricultural landscape. Applied Vegetation Science, 7(1), 61-70.

Sudaryanto, S., Fikri, B., & Maryani, S. (2019). Query optimization on distributed health database dbd for supporting data center with materialized view and minimizing attribute involvement. Journal of Theoretical and Applied Information Technology, 97(11).

Sultana, N., Hashem, T., & Kulik, L. (2014, November). Group nearest neighbor queries in the presence of obstacles. In Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 481-484).

Sun, W. W., Chen, C. N., Zhu, L., Gao, Y. J., Jing, Y. N., & Li, Q. (2015). On efficient aggregate nearest neighbor query processing in road networks. Journal of computer science and technology, 30(4), 781-798.

Tao, Y., Papadias, D., & Shen, Q. (2002, January). Continuous nearest neighbor search. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases (pp. 287-298). Morgan Kaufmann.

Wang, W., Xu, J., Xu, M., Zheng, N., & Ge, E. (2015, November). Probabilistic Group Nearest Neighbors query based on Voronoi diagram. In Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics (pp. 36-41).

Xuan, K., Zhao, G., Taniar, D., & Srinivasan, B. (2008, December). Continuous range search query processing in mobile navigation. In 2008 14th IEEE International Conference on Parallel and Distributed Systems (pp. 361-368). IEEE.

Zhang, Y., Lin, X., Zhu, G., Zhang, W., & Lin, Q. (2010, March). Efficient rank based knn query processing over uncertain data. In 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010) (pp. 28-39). IEEE.

Zhou, Y., Du, J., Yang, Y., & Shi, W. (2018). Location Nearest Neighbor Query Method Based on Voronoi Map. Journal of Beijing University of Technology, (2), 11.

Zhu, L., Jing, Y., Sun, W., Mao, D., & Liu, P. (2010, November). Voronoi-based aggregate nearest neighbor query processing in road networks. In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 518-521).