APPROXIMATE K-NEAREST NEIGHBOUR BASED SPATIAL CLUSTERING USING K-D TREE

Dr. Mohammed Otair

Department of Computer Information Systems, Amman Arab University, Amman, Jordan
Otair@aau.edu.jo

ABSTRACT

Different spatial objects that vary in their characteristics, such as molecular biology and geography, are presented in spatial areas. Methods to organize, manage, and maintain those objects in a structured manner are required. Data mining raised different techniques to overcome these requirements. There are many major tasks of data mining, but the mostly used task is clustering. Data set within the same cluster share common features that give each cluster its characteristics. In this paper, an implementation of Approximate kNN-based spatial clustering algorithm using the K-d tree is proposed. The major contribution achieved by this research is the use of the k-d tree data structure for spatial clustering, and comparing its performance to the brute-force approach. The results of the work performed in this paper revealed better performance using the k-d tree, compared to the traditional brute-force approach.

KEYWORDS

Spatial data, Spatial Clustering, Approximate kNN, K-d tree, brute-force.

1. INTRODUCTION

Spatial data are data that have a location (spatial) characteristic [7] [26]. Spatial data are stored in databases called spatial databases, which contain spatial data type in its data model -side by side- with the ordinary non-spatial types. Spatial data are mainly required for Geographic Information Systems (GIS), whose information is related to geographic locations. GIS model support spatial data types, such as point, line, and polygon [25]. Spatial databases are database systems that manage spatial data. They are designed to manipulate both spatial information and the non-spatial attributes of that data. For the purpose of providing efficient and effective way to attain spatial data it is important to come up with indices (plural of index). When the data are based on multidimensional trees, these indices will meet success. Spatial database system is a system that was made to utilize and manipulate the non-spatial data and spatial data skilfully to realize the data [8]. The size of spatial data that are presented in these systems is also growing dramatically. The complication of spatial data and its databases mean that it is impossible for human beings to analyze such huge sizes of database that requires new techniques to be able to analyze and to discover their trends. Traditional databases need to have an additional space or dimension to store these data. In addition to its retrieval and analyzing such database would be too expensive or time consuming [8].

Spatial data mining is the operation of applying different mining methods to a spatial database to find non-trivial patterns from the spatial data [7]. One of the most important data mining methods is clustering. Clustering is the task of dividing the objects from spatial database into groups (clusters) in such a way that objects in one cluster are similar and share common features, while objects from different clusters are dissimilar [7] [24]. Clustering helps in discovering and

DOI: 10.5121/ijdms.2013.5108

understanding the natural structure and grouping the data in data set. Spatial clustering algorithms must be able to determine clusters of different dimensions, sizes, shapes and density [7] [24].

The K-D tree [14] is a structure of data was brought about by Jon Bentley in 1975. K-D tree and its versions remain the mostly used data structures for searching through multidimensional spaces; although it is somewhat old in age and had a several number of index structures. Based on The Digital Library of ACM, the paper [14] which came up with this data structure is one of the mostly known papers in computational geometry with 626 citations in early of 2013. The K-D tree has dual or binary search tree structure. It can be utilized effectively with nearest neighbour/range queries and searches on condition that the dimension is not too much. In problems of document analysis it has two typical dimensions, so that K-D tree can be a powerful for laying out problems of analysis. Unfortunately, K-D tree does not work properly with too much dimensions (because it needs to visit huge number of tree branches). Many algorithms came about to overcome these restrictions in order to carry out faster search [18] [22] [16]. Various improvements to k-nearest neighbour methods are possible by using proximity graphs [31].

As mentioned before, one of the earliest data structures proposed for the nearest neighbour problem that is still the most commonly used is the k-d tree [13] that is essentially a hierarchical decomposition of space a long different dimensions. In respect of a limited number of dimensions, its structure can be useful in the answering of some types of important queries such as the nearest neighbour query in linear space and logarithmic time. However, its performance would go down when the number of dimensions go more than two [20]. The main trouble for high dimensions in nearest neighbour search will sustain from the suffering of dimensionality, because either the processing time or the requirement of capacity expansion grows dramatically in d dimension. Finding a way to decrease complications of computing K-D tree search is highly interesting in these areas.

This paper will study the problem of finding the approximate k-nearest neighbour based spatial clustering using Kd-tree of a query point in a high dimensional (2D and 3D) space: given a database of n points in a d dimensional space, find approximate k-nearest neighbour of a query point. The use of k-d trees is a well known optimization to the kNN algorithm [34]. To work on in parallel of KNN Algorithm in CUDA by Gracia in [10] works as the starting point in KNN improvements is proved. The CUDA algorithm they brought about is a parallel application of the standard Brute Force KNN algorithm. They mentioned that the speedup was around 40 times faster than serial KNN using K-D trees and 100 times faster than executing of the implementation of serial CPU [10]. It was realized also that the points' dimension had quite little effect on total computation time [11]. One thing is taken for granted is that if the dimension of data is k, if a K-D tree has much more than 2k data points; then it can be useful. In respect of high dimensions, it usually goes to approximate nearest neighbour ANN searches. ANN is a great library for that and it is written in C++. Brute force, K-D trees search and other approximate approaches were implemented well by ANN. It aids automatically to mesh the parameters and change them flexibly.

2. RELATED WORK

Most of the variants of KNN algorithms are very slow in carrying out clustering work (a k-d tree is an example). The main performance problem of the current k-d tree which based on the nearest neighbour search algorithm is inflicted with reduction of performance due to curse dimension and that performance needed to be improved [15]. In this research, this problem is resolved by ANN algorithm in order to speed up the clustering process as well. A focus of this research is to improve performance of the KNN approach and to demonstrate its performance in a real-world problem. Another objective of this paper is to test the improvement performance of the existing K-d tree approach.

There are a large number of methods, techniques and algorithms that organize, manage, and maintain spatial objects in a structured manner. However, the most commonly used are:

2.1. K-Nearest Neighbour

T. M. Cover and P. E. Hart purpose k-nearest neighbour (kNN) in which nearest neighbour is calculated on the basis of value of k, that specifies how many nearest neighbours are to be considered to define class of a sample data point [30]. T. Bailey and A. K. Jain improve kNN which is based on weights [28]. The training set of points was given weights based on their distance that found in the sample data points. However, the requirements of the memory and complications of computing were still the most important anxieties [18]. To resolve memory problem, the size of data must be decreased.

The k-nearest neighbour join combines each point of one point set with its k nearest neighbours [5]. The general model of a KNN query is that the user gives many query types such a point query in multidimensional space and a distance metric for measuring distances between points in this space. The system is then tried to find the K closest or nearest answers in the database from the submitted query (i.e. query point). Generally distance metrics may include: Euclidean distance, Manhattan distance, etc. It is possible that a majority of the answers to a KNN query may be very similar to one or more of the other answers, especially when the data has clusters [1]. The kNN implementation can be done using box-decomposition trees (ball tree) [23] [29], k-d tree [19], these algorithms increase the speed of classical kNN algorithm. The k-Nearest Neighbour (kNN) algorithm has a wide range of applications in modern day computing.

The simplified variation of KNN algorithm is the implementing of Brute Force which consists of three phases. The first one is to compute the distances from each query point to every reference point of the training group of point. The second is to order these distances and choose the K objects that are the nearest from which. In the final phase, the categorizing process can be carried out. More formally, KNN finds the K closest (or most similar) points to a query point among N points in a d- dimensional attribute (or feature) space. K is the number of neighbours that are considered from a training data set and typically ranges from 1 to 20. The advantages of k Nearest Neighbour (kNN) can be summarized as in [30]: Training is very fast, Simple and easy to learn, Robust to noisy training data, and Effective if training data is large. However, there many disadvantages, as well: Biased by value of k, Computation Complexity, Memory limitation, being a supervised learning lazy algorithm i.e. runs slowly and easily fooled by irrelevant Attributes.

2.2. Spatial Indexing

To handle special data in an efficient way, as it is desired in geo-data implementations that are interested in spatial search on multidimensional spaces, such database system needs technique of index which aids in redeeming and getting data items quickly based on their spatial location [2]. The main important needs for these data structures are the ability to supply rapid reach to huge data volume and hold spatial relations such as settling in and neighbourhood for indexed things.

Several techniques were suggested to access spatial objects [33] like R-tree which considered as one of the mostly used hierarchical data structure. R-tree is used for indexing spatial objects efficiently that they have spatial multidimensional extent [2] [33]. There are several tree index structures like R* tree, R+ tree [17][33] and other similar structures to enhance the classical R-tree structure.

In this paper, the KD-tree will be used for spatial indexing in a new KNN-based spatial clustering algorithm.

2.3. K-d Tree

A k-dimensional tree or a K-D tree is a commonly used data structure to organize several numbers of points in a multidimensional space called k dimensions. A searching process in K-D tree is a binary with additional imposed limitations on it. It is very helpful for some types of search queries such as: nearest neighbour and range. The root-cell of K-D tree illustrates the total volume of simulation. The rectangular of sub-volumes will be illustrated by the other cells which involve the moment of regions for: quadrupole, mass center, and mass.

K-D tree is one of the oldest used data structures for indexing in multi-dimensions. In K-D tree, each level divides the space into two divisions as represented in figure 2.1; the division takes place along one of the node dimension at the highest tree level and with other nodes' dimension at the subsequent level and so on, repetition through the entire dimensions. The division goes on in a way that at each nodes around one half of the points that are kept in the subtree stay on one side and other half stay on the other. The process of division stops when a node reaches less than a determined maximum number of points [32].

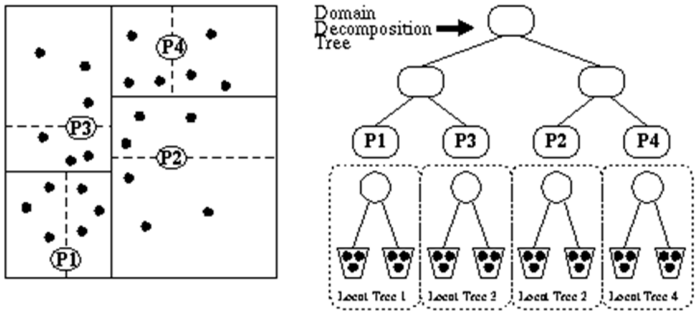

Figure 2.1: K-d Tree Partitioning (Source: http://www-hpcc.astro.washington.edu)

Despite of there are many different K-D tree variants were introduced; however, their functions is always to decompose space hierarchically into sort of modicum number of cells, each cell of them has a limited number of input objects). This will facilitate a speed technique to reach/process an input object by its position. It should go down and pass through the hierarchy until we access or reach the cell which includes the object. K-D trees are constructed by dividing point sets repeatedly along different dimensions. In this tree, each node is represented by a dimension that division the points into two set either left or right (or it could be up or down), each set of them will contain half number of points that derived from the root node. Once again these branches are decomposed into equal two equal parts, using partition through a distinct dimension. The decomposition process will terminate at log n levels or over, where each point has its own leaf node. The decomposing repeats via the several dimensions for the several tree levels, for the portion using the mid-point. K-d trees can work effectively with a limited number of dimensions but it may be unsuccessful when the number of dimensions become over three [27].

K-d tree nearest neighbour (kdNN) as described in [19] has many characteristics like: simple, fast, and provide completely balanced tree. At the other hand, it has several significant drawbacks such as: need extreme search, time consuming, and impulsively divide points into two equal parts which may miss out on data structure.

2.4. Spatial Clustering

Clustering is one of the most research areas that have been studied. Many clustering techniques and algorithms have been implemented and developed. The main categories of the clustering algorithms are:

2.4.1. Partitioning algorithms

In these algorithms build a portion of a database of several objects into a group of clusters, it builds the clusters within single step, and only one group of these clusters is built. Despite of some different clusters' groups may be built locally within the different algorithms. Even only one clusters group is produced, the user should type the number of targeted clusters. Each cluster within a group is represented using one of the objects' clusters which are located close to their cluster's center which is called k-means algorithms or using the center which is called k-medoid algorithms [12].

2.4.2. Hierarchical algorithms

Such types of clustering algorithms organize a tree data structure (called a dendrogram) of the clusters using: hierarchy structure or tree. The root in this tree is considered as a single cluster which involves all the spatial objects in the spatial area. The other nodes are considered as clusters with only one object. These algorithms operate repeatedly to achieve merging or splitting until a stopping condition is satisfied or the clustering process encompassed all objects. Hierarchical algorithms are classified into: divisive (from the root down to leaves) and agglomerative (from the leaves up to the root) algorithms, where divisive approach or agglomerative approach made up the dendrogram.

2.4.3. Density-based algorithms

Density is the concept that based on all of these algorithms; they use a mechanism of density-connected points. These algorithms have several characteristics, such as its ability to detect noise, clusters of arbitrary styles, and they need a density variable as a termination criterion [12].

2.4.4. Grid-based algorithms

In the Grid-based algorithms, instead the number of data objects they depend on the grid size. They use a single regular grid mesh to split the whole domain problem into multiple cells. Each cell represents the data objects using a series of statistical features from the objects. These algorithms the grid cells to achieve the clustering, instead to perform the clustering on the database directly. In compare with categories of other clustering algorithms, Grid-based algorithms have the shortest run time [12].

The run time is too inactive for most of the clustering algorithms especially with huge databases. To resolve this problem or drawback, many techniques have been proposed to enhance the clustering algorithms. For instance, the researchers in [9] discussed the benefits of the use of clustering in spatial problems using R*-tree. Moreover, based-focusing techniques were proposed in [4] [9], they proceed in the following phases: (1) constructing a sample of the database that is came from each R*-tree data page and (2) implementing the clustering algorithm only to that sample.

This paper introduces a new KNN- based spatial clustering algorithm using the Kd-tree, which will be described in the following section.

3. APPROXIMATE K-NEAREST NEIGHBOUR BASED SPATIAL CLUSTERING

This paper is concerned with the problem of Approximate KNN based spatial clustering. The concept is based on clustering spatial points that are the most nearest and have similar properties into one cluster. In order to compute the nearest neighbours, a simple brute-force search can be used. However, in order to handle large volumes of spatial data organized in high dimensions, brute force will be too slow. Therefore, the need arises for other techniques that are based on indexing spatial data into a data structure that can be used to answer nearest neighbours queries.

In this paper, the k-d tree is chosen as the data structure to index spatial points. In order to evaluate the efficiency of the k-d tree for achieving this purpose, several experiments have been performed to evaluate and compare the efficiency of the proposed method when applied on various data size, various dimensions, and multiple k values. Implement approximate k-nearest-neighbour (kNN) search using a brute force approach as well as with the help of the kd-tree will be used to reach of the main objective of this research (i.e. to speed-up K-nearest neighbour searches). Then, the obtained results will be presented, discussed, and analyzed in the next section.

3.1. Brute Force Algorithm

Brute-force search or exhaustive search is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. The simple sequence of operations for the brute force algorithm is shown below (taken from [10]):

1. Calculate all the distances between the query point and reference points
2. Sort these distances.
3. Select the *k* reference points with the smallest distances, then categorization vote by *k* nearest objects.
4. Iterate all the previous steps from 1 to 3 for all query points.

3.2. K-d Tree Algorithm

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyper-plane that divides the space into two parts, known as half-spaces. Points to the left of this hyper-plane represent the left subtree of that node and points right of the hyper-plane are represented by the right subtree. The hyper-plane direction is chosen in the following way: every node in the tree is associated with one of the k-dimensions, with the hyper-plane perpendicular to that dimension's axis. So, for example, if for a particular split the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right subtree. In such a case, the hyper-plane would be set by the x-value of the point, and its normal would be the unit x-axis [14].

The nearest neighbor search (NN) algorithm aims to find the point in the tree that is nearest to a given input point. This search can be done efficiently by using the tree properties to quickly eliminate large portions of the search space. Searching for a nearest neighbor in a k-d tree proceeds as in Appendix-A.

3.3. Approximate Nearest Neighbour (ANN)

An approximation may be agreeable to access the nearest neighbour in some implementations which needs to return spatial objects. In other words, this type of algorithms doesn't ensure to access the exact nearest neighbour every time, in order to enhance the savings in the memory and speed. To find the exact nearest neighbour in these algorithms in most cases, but this mainly based on the training or test dataset. The searching process of the algorithms that use the approximate nearest neighbour technique could involve: best bin first, locality-sensitive hashing and neutral box-decomposition [21].

The k-d tree is searched for an approximate nearest neighbour. The point is returned through one of the arguments, and the distance returned is the squared distance to this point. The method used for searching the k-d tree is an approximate adaptation of the search algorithm described by Friedman, Bentley, and Finkel as in [13].

The algorithm operates recursively. When first encountering a node of the k-d tree we first visit the child which is closest to the query point. On return, we decide whether we want to visit the other child. If the box containing the other child exceeds 1/(1+eps) times the current best distance, then we skip it (since any point found in this child cannot be closer to the query point by more than this factor.) Otherwise, we visit it recursively. The distance between the queried point and a box is calculated actually (not approximated as is overwhelmingly calculated in the classical k-d tree), by additional distance updates, according to the methodology that proposed by Mount and Arya [3].

The main entry points to the ANN search sets things up and then call the recursive routine another search routine. This is a recursive routine which performs the processing for one node in the k-d tree. There are two versions of this virtual procedure, one for splitting nodes and one for leaves. When a splitting node is visited, we determine which child to visit first (the closer one), and visit the other child on return. When a leaf is visited, we compute the distances to the points in the buckets, and update information on the closest points. Some trickery is used to incrementally update the distance from a k-d tree rectangle to the query point. This comes about from the fact that which each successive split, only one component along the dimension that is split) of the squared distance to the child rectangle is different from the squared distance to the parent rectangle.

ANN is a library written in the C++ programming language to support both exact and approximate nearest neighbour searching in spaces of various dimensions. It was implemented by David M. Mount of the University of Maryland and Sunil Arya of the Hong Kong University of Science and Technology. ANN (pronounced like the name \Ann") stands for the Approximate Nearest Neighbour library. ANN is also a testbed containing programs and procedures for generating data sets, collecting and analyzing statistics on the performance of nearest neighbour algorithms and data structures, and visualizing the geometric structure of these data structures.

Retrieving the nearest neighbour object to a submitted query accurately (when there are too much dimensions) seems to be is not an easy task. It is most probably to response a query by a classical brute-force steps of calculating the distances between each one of the dataset points and the point being queried; however, this will make calculating process is very slow for some applications that need a too many number of queries be tested on the same circumstances and the data set at the same time. To resolve this problem, a set of data-points should be pre-processed into data structures that help in answering the queries of nearest neighbour. Several numbers of data structures have been introduced for this endeavour [6].

One of the drawbacks with searching in actual nearest neighbour is that for all methods (exclude the classical brute-force search) are space or the processing time grows dramatically as a function of dimension. Therefore, most these methods are not considerably result better search than classical brute-force, except when the number of dimensions are very limited. However, Mount, Arya et al. shown that if the user have the capacity to afford a small percentage of searching errors(i.e. it could return a point that is considerably far away from the point being queried instead the actual nearest); after that a considerable enhancements could be achieved at the running time. ANN is considered as a tool for answering two cases: approximate or exact for the nearest neighbour queries [6].

ANN was established as a testbed for a type of searching algorithms that adopt the nearest neighbour, or those partially based on perpendicular space decompositions. These algorithms include: box-decomposition trees and k-d trees. The ANN library provides different techniques to construct such search structures. This library also offers two methods to search into these structures [6], they are: priority search and standard tree-ordered search.

4. EXPERIMENTAL EVALUATION AND RESULTS ANALYSIS

The Approximate KNN–based spatial clustering method has been tested through several experiments using the k-d tree. The nearest neighbour spatial distance was computed using the ANN library, which was modified to perform spatial clustering. The experiments have been performed on a 2.8 GHz PC, with Intel Pentium 4 processor and 256 MB memory. The ANN library has been compiled under visual C++ .Net 2003.

The experiments were conducted for computing 1, 2, 3, 4 and 5 nearest neighbours, with data sets of sizes 0.5 and 1 MB assuming the spatial points are organized in 2 and 3 dimensions.

The first set of experiments has been performed to compare the performance of both, K-d tree and brute-force, for a 0.5 MB data set in a 2-dimentional space. The execution time required by both approaches was approximately the same. Better performance was achieved when the k-d tree was used for larger number of K nearest neighbours. Figure 4.1 shows the execution time required by K-d tree when 1, 2, 3 and 4 nearest neighbours are computed. However, for 5 nearest neighbours, better performance was observed using the K-d tree.

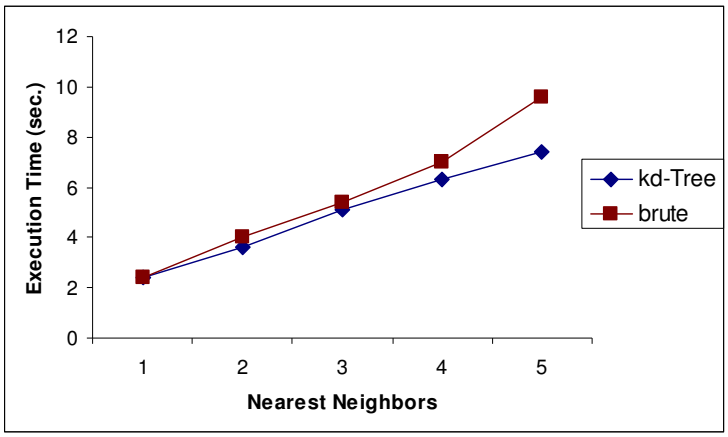

Figure 4.1: K-d Tree vs. brute-force Performance (0.5 MB data set, 2-D)

The next set of experiments has been conducted for the same data set size (0.5 MB), however; in a 3-dimensional space. It is obvious from figure 4.2 that as the number of nearest neighbours increases; the K-d tree shows better performance.

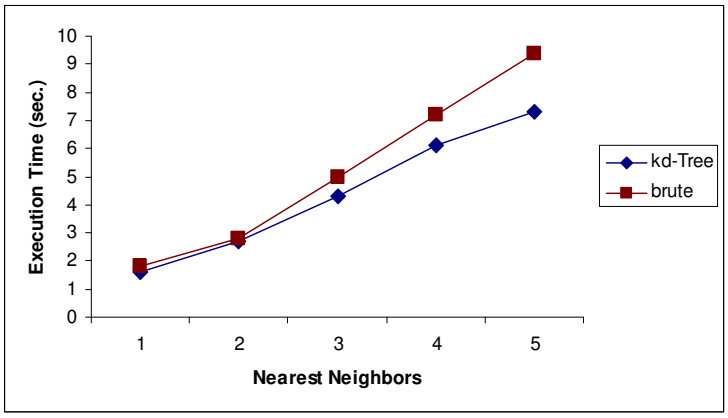

Figure 4.2: K-d Tree vs. brute-force Performance (0.5 MB data set, 3-D)

With larger volumes of data, an enhanced performance was achieved by K-d tree. The enhancement becomes clearer for data organized in 3 dimensions. These results are clear from figures 4.3 and 4.4.

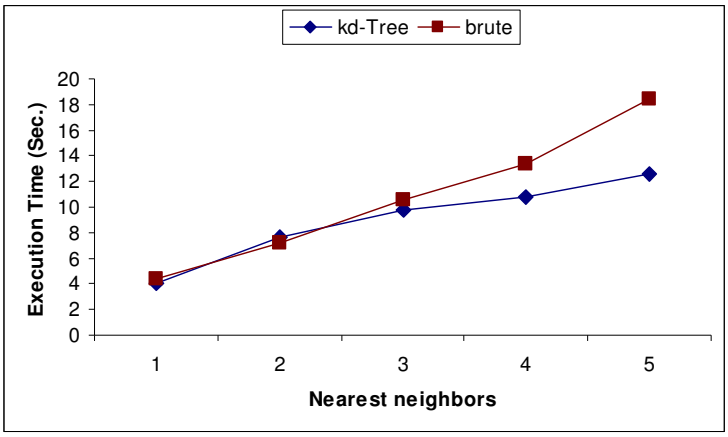

Figure 4.3: K-d Tree vs. brute-force Performance (1 MB data set, 2-D)

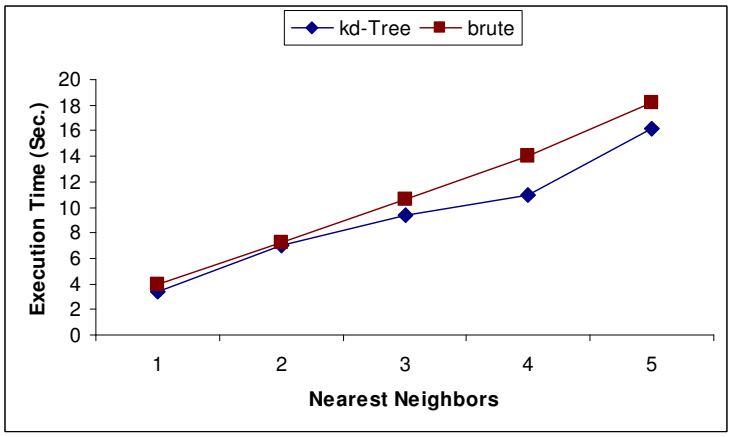

Figure 4.4: K-d Tree vs. brute-force Performance (1 MB data set, 3-D)

As seen in the previous figures (4.3 and 4.4), the K-d Tree algorithm needs less number of seconds in comparing with brute-force especially when using a large volumes of data regardless the number of dimensions (i.e. if it is in 2D or 3D).

5. CONCLUSIONS

The major contribution achieved by this research is the use of the approximate k-nearest neighbour with k-d Tree data structure for spatial clustering, and comparing its performance to the brute-force approach. The results of the work performed in this paper revealed better performance using the k-d Tree, compared to the traditional brute-force approach. The efficiency of the data structure primarily depends on a particular implementation and data set. A poorly balanced tree will mean we have to search way more data than we need to.

FUTURE WORK

As a future work, other data structures can be used to achieve spatial clustering. The results obtained can be compared to this paper's results.

ACKNOWLEDGMENT

The Author would like to thank Sama Al-Momani and Zeinab Jaradat (My Students) for their help in the experiments that done in this research.

REFERENCES

[1] Anoop Jain , Parag Sarda , & Jayant R. Haritsa, (2003) "Providing Diversity in K-Nearest Neighbour Query", Tech. Report TR-2003-04.

[2] Antonin Guttman, (1984), "R-Trees: A Dynamic Index Structure for Spatial Searching". SIGMOD Conference, 47-57.

[3] Arya & Mount, (1993) "Algorithms for fast vector quantization", Proc. of DCC '93: Data Compression Conference, eds. J. A. Storer and M. Cohn, IEEE Press, 381-390.

[4] Beckmann N., Kriegel H.-P., Schneider R., Seeger B., (1990) "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles" , Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, pp. 322-331.

[5] Christian Böhm, (2002) "Powerful Database Primitives to Support High Performance Data Mining", Tutorial, IEEE Int. Conf. on Data Mining.

[6] David M. Mount , (2010) "ANN Programming Manual", University of Maryland.

[7] Dunham M., (2002) "Data Mining: Introductory and Advanced Topics", New Jersey, Prentice Hall.

[8] Erica Kolatch, (2001) "Clustering Algorithms for Spatial Databases: A Survey", citeseerx.ist.psu.edu.

[9] Ester M., Kriegel H.-P., Xu X, (1998) "Incremental Clustering for Mining in a Data Warehousing Environment", Proceedings of the 24th VLDB Conference.

[10] Garcia, V., Debreuve, E., and Barlaud, M., (2008) "Fast k nearest neighbor search using GPU", IEEE Computer Society Conference, 1-6.

[11] Graham Nolan, (2009) "Improving the k-Nearest Neighbour Algorithm with CUDA", Honours Programme, The University of Western Australia.

[12] J. Sander, M. Ester, H. Kriegel, and X. Xu, (1998) "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications". Journal of Data Mining and Knowledge Discovery, Vol. (2), Issue (2), 169-194.

[13] J.L. Bentley, Friedman, J.H., Finkel, R.A., (1977) "An algorithm for finding best matches in logarithmic expected time", ACM Transactions on Mathematical Software 3(3), 209–226.

[14] J.L. Bentley, (1975) "Multidimensional binary search trees used for associative searching", Comm. ACM, 18(9):509 517.

[15] Jaim Ahmed, (2009) "Efficient K-Nearest Neighbor Queries Using Clustering With Caching", Master Thesis, The University of Georgia.

[16] Marius Muja and David G. Lowe, (2009) "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration", in International Conference on Computer Vision Theory and Applications (VISAPP'09).

[17] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. (1990) "The R*-tree: an efficient and robust access method for points and rectangles". ACM SIGMOD, pages 322-331.

[18] Nitin Bhatia, Vandana, (2010) "Survey of Nearest Neighbor Techniques", International Journal of Computer Science and Information Security, Vol. 8, No. 2.

[19] R. F. Sproull, (1991) “Refinements to Nearest Neighbor Searching”, Technical Report, International Computer Science, ACM (18) 9, pp 507-517.

[20] Rina Panigrahy, (2006) "Nearest Neighbor Search using Kd-trees", citeseerx.ist.psu.edu.

[21] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Wu, (1998) "An optimal algorithm for approximate nearest neighbor searching", Journal of the ACM, 45(6):891-923.

[22] S. Dhanabal, S. Chandramathi, (2011) "A Review of various k-Nearest Neighbor Query Processing Techniques", International Journal of Computer Applications, Volume 31– No.7.

[23] S. N. Omohundro, (1989) “Five Ball Tree Construction Algorithms”, Technical Report.

[24] Shekhar S, Zhang P, Huang Y, Vatsavai R., (2003) "Trends in Spatial Data Mining". Department of Computer Science and Engineering, University of Minnesota, Minneapolis.

[25] Shekhar S, Zhang P., (2004) "Spatial Data Mining: Accomplishments and Research Needs". University of Minnesota. GIS Science.

[26] Shekhar S., Chawla S., (2003) "Spatial Databases: A tour". Pearson education Inc, Upper Saddle River, New Jersey.

[27] Steven S. Skiena, (2010) "The Algorithm Design Manual" , 2nd Edition, Stony Brook, NY 11794-4400.

[28] T. Bailey and A. K. Jain, (1978) “A note on Distance weighted k-nearest neighbor rules”, IEEE Trans. Systems, Man Cybernatics, Vol.8, pp 311-313.

[29] T. Liu, A. W. Moore, A. Gray, (2006) “New Algorithms for Efficient High Dimensional Non-Parametric Classification”, Journal of Machine Learning Research, pp 1135-1158.

[30] T. M. Cover and P. E. Hart, (1967) “Nearest Neighbor Pattern Classification”, IEEE Trans. Inform. Theory, Vol. IT-13, pp 21-27.

[31] Toussaint GT (2005). "Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining". International Journal of Computational Geometry and Applications 15 (2): 101–150.

[32] William R. Mark, Gordon Stoll, (2006) "Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic", Warren Hunt, IEEE Symposium on Interactive Ray Tracing.

[33] Yu-Chen Fu, Zhi-Yong Hu, We1 Guo, Dong-Ru Zhou, (2003) "QR-tree: a hybrid spatial index structure", Proceedings of the Second International Conference on Machine Learning and Cybernetics.

[34] Zhou, K., Hou, Q., Wang, R., and Guo, B. (2008) "Real-time kd-tree construction on graphics hardware". ACM Trans. Graph. 27, 5, 1-11.

Author

Mohammed A. Otair is an Associate Professor in Computer Information System at Amman Arab University-Jordan. He received his B.Sc. in Computer Science fro IU-Jordan and his M.Sc. and Ph.D in 2000, 2004, respectively, from the Departme of Computer Information Systems-Arab Academy. His major interests are Mobi Computing, Data Mining and Databases Neural Network Learning Paradigms, Wel computing, E-Learning. He has more than 29 publications.

Appendix-A

Searching for a nearest neighbor in a *k*-d tree [Source: Wikipedia]

1. Starting with the root node, the algorithm moves down the tree recursively, in the same way that it would if the search point were being inserted (i.e. it goes left or right depending on whether the point is less than or greater than the current node in the split dimension).
2. Once the algorithm reaches a leaf node, it saves that node point as the "current best"

3. The algorithm unwinds the recursion of the tree, performing the following steps at each node:
    A. If the current node is closer than the current best, then it becomes the current best.
    B. The algorithm checks whether there could be any points on the other side of the splitting plane that are closer to the search point than the current best. In concept, this is done by intersecting the splitting hyper-plane with a hyper-sphere around the search point that has a radius equal to the current nearest distance. Since the hyper-planes are all axis-aligned this is implemented as a simple comparison to see whether the difference between the splitting coordinate of the search point and current node is less than the distance (overall coordinates) from the search point to the current best.
        I. If the hyper-sphere crosses the plane, there could be nearer points on the other side of the plane, so the algorithm must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
        II. If the hyper-sphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.
4. When the algorithm finishes this process for the root node, then the search is complete.

Generally the algorithm uses squared distances for comparison to avoid computing square roots.

Additionally, it can save computation by holding the squared current best distance in a variable for comparison.