International Journal of Computer Science and Business Informatics

IJCSBI.ORG

HOV-kNN: A New Algorithm to Nearest Neighbor Search in Dynamic Space

Mohammad Reza Abbasifard
Department of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran

Hassan Naderi
Department of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran

Mohadese Mirjalili
Department of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran

ABSTRACT
Nearest neighbor search is one of the most important problem in computer science due to its numerous applications. Recently, researchers have difficulty to find nearest neighbors in a dynamic space. Unfortunately, in contrast to static space, there are not many works in this new area. In this paper we introduce a new nearest neighbor search algorithm (called HOV-kNN) suitable for dynamic space due to eliminating widespread preprocessing step in static approaches. The basic idea of our algorithm is eliminating unnecessary computations in Higher Order Voronoi Diagram (HOVD) to efficiently find nearest neighbors. The proposed algorithm can report k-nearest neighbor with time complexity $O(knlogn)$ in contrast to previous work which was$O(k^2nlogn)$. In order to show its accuracy, we have implemented this algorithm and evaluated is using an automatic and randomly generated data point set.

Keywords
Nearest Neighbor search, Dynamic Space, Higher Order Voronoi Diagram.

1. INTRODUCTION
The Nearest Neighbor search (NNS) is one of the main problems in computer science with numerous applications such as: pattern recognition, machine learning, information retrieval and spatio-temporal databases [1-6]. Different approaches and algorithms have been proposed to these diverse applications. In a well-known categorization, these approaches and algorithms could be divided into *static* and *dynamic* (moving points). The

existing algorithms and approaches can be divided into three categories, based on the fact that whether the query points and/or data objects are moving. They are (i) static kNN query for static objects, (ii) moving kNNquery for static objects, and (iii) moving kNN query for moving objects [15].

In the first category data points as well as query point(s) have stationary positions [4, 5]. Most of these approaches, first index data points by performing a pre-processing operation in order to constructing a specific data structure. It's usually possible to carry out different search algorithms on a given data structure to find nearest neighbors. Unfortunately, the pre-processing step, index construction, has a high complexity and takes more time in comparison to search step. This time could be reasonable when the space is static, because by just constructing the data structure multiple queries can be accomplished. In other words, taken time to pre-processing step will be amortized over query execution time. In this case, searching algorithm has a logarithmic time complexity. Therefore, these approaches are useful, when it's necessary to have a high velocity query execution on large stationary data volume.

Some applications need to have the answer to a query as soon as the data is accessible, and they cannot tolerate the pre-processing execution time. For example, in a dynamic space when data points are moving, spending such time to construct a temporary index is illogical. As a result approaches that act very well in static space may be useless in dynamic one.

In this paper a new method, so called HOV-kNN, suitable for finding k nearest neighbor in a dynamic environment, will be presented. In k-nearest neighbor search problem, given a set P of points in a d-dimensional Euclidian spaceR^d ($P \subset R^d$) and a query point q ($q \in R^d$), the problem is to find k nearest points to the given query point q [2, 7]. Proposed algorithm has a good query execution complexity $O(knlogn)$ without enduring from time-consuming pre-processing process. This approach is based on the well-known Voronoi diagrams (VD) [11]. As an innovation, we have changed the Fortune algorithm [13] in order to created order k Voronoi diagrams that will be used for finding kNN.

The organization of this paper is as follow. Next section gives an overview on related works. In section 3 basic concepts and definitions have been presented. Section 4 our new approach HOV-kNN is explained. Our experimental results are discussed in section 5. We have finished our paper with a conclusion and future woks in section 6.

2. RELATED WORKS

Recently, many methods have been proposed for k-nearest neighbor search problem. A naive solution for the NNS problem is using linear search

method that computes distance from the query to every single point in the dataset and returns the k closest points. This approach is guaranteed to find the exact nearest neighbors [6]. However, this solution can be expensive for massive datasets. So approximate nearest neighbor search algorithms are presented even for static spaces [2].

One of the main parts in NNS problem is data structure that is roughly used in every approach. Among different data structures, various tree search most used structures which can be applied in both static and dynamic spaces. Listing proposed solutions to kNN for static space is out of scope of this paper. The interested reader can refer to more comprehensive and detailed discussions of this subject by [4, 5]. Just to name some more important structures, we can point to kd-tree, ball-tree, R-tree, R*-tree, B-tree and X-tree [2-5, 8, 9].In contrast, there are a number of papers that use graph data structure for nearest neighbor search. For example, Hajebi et al have performed Hill-climbing in kNN graph. They built a nearest neighbor graph in an offline phase, and performed a greedy search on it to find the closest node to the query [6].

However, the focus of this paper is on dynamic space. In contrast to static space, finding nearest neighbors in a dynamic environment is a new topic of research with relatively limited number of publications. Song and Roussopoulos have proposed Fixed Upper Bound Algorithm, Lazy Search Algorithm, Pre-fetching Search Algorithm and Dual Buffer Search to find k-nearest neighbors for a moving query point in a static space with stationary data points [8]. Güting et al have presented a filter-and-refine approach to kNN search problem in a space that both data points and query points are moving. The filter step traverses the index and creates a stream of so-called units (linear pieces of a trajectory) as a superset of the units required to build query's results. The refinement step processes an ordered stream of units and determines the pieces of units forming the final precise result [9].Frentzos et al showed mechanisms to perform NN search on structures such as R-tree, TB-Tree, 3D-R-Tree for moving objects trajectories. They used depth-first and best-first algorithms in their method [10].

As mentioned, we use Voronoi diagram [11] to find kNN in a dynamic space. D.T. Lee used Voronoi diagram to find k nearest neighbor. He described an algorithm for computing order-k Voronoi diagram in $O(k^2 nlogn)$ time and $O(k^2(N-k))$ space [12] which is a sequential algorithm. Henning Meyerhenke presented and analyzed a parallel algorithm for constructing HOVD for two parallel models: *PRAM* and *CGM* [14]. In these models he used Lee's iterative approach but his model stake $O\left(\frac{k^2(n-k)logn}{p}\right)$running time and $O(k)$ communication rounds on a CGM

with $O(\frac{k^2(N-k)}{p})$ local memory per processor [14]. p is the number of participant machines.

3. BASIC CONCEPTS AND DEFINITIONS

Let P be a set of n sites (points) in the Euclidean plane. The Voronoi diagram informally is a subdivision of the plane into cells (Figure 1)which each point of that has the same closest site [11].

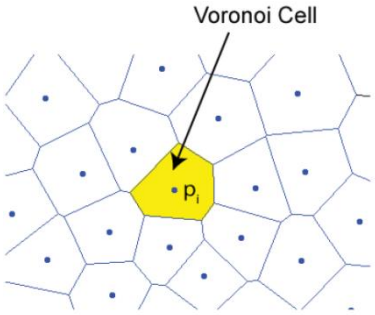

Figure 1.Voronoi Diagram

Euclidean distance between two points p and q is denoted by $dist(p,q)$:

$$dist(p,q) := \sqrt{(px-qx)^2+(py-qy)^2} \qquad (1)$$

Definition (Voronoi diagram):*Let $P=\{p_1,p_2,\dots,p_n\}$ be a set of n distinct points (so called sites) in the plane. Voronoi diagram of P is defined as the subdivision of the plane into n cells, one for each site in P, with the characteristic that q in the cell corresponding to site p_i if$dist(q,p_i) < dist(q,p_j)$ for each $p_j \in P$ with $j \neq i$ [11].*

Historically, $O(n^2)$*incremental* algorithms for computing VD were known for many years. Then $O(nlogn)$ algorithm was introduced that this algorithm was based on *divide and conquer*, which was complex and difficult to understand. Then Steven Fortune [13] proposed a plane sweep algorithm, which provided a simpler $O(nlogn)$ solution to the problem.

Instead of partitioning the space into regions according to the closest sites, one can also partition it according to the k closest sites, for some $1 \leq k \leq n-1$. The diagrams obtained in this way are called higher-order Voronoi diagrams or HOVD, and for given *k*, the diagram is called the order-k Voronoi diagram [11]. Note that the order-1 Voronoi diagram is nothing more than the standard VD. The order-(n−1) Voronoi diagram is the farthest-point Voronoi diagram (Given a set P of points in the plane, a point of P has a cell in the farthest-point VD if it is a vertex of the convex hull), because the Voronoi cell of a point p_i is now the region of points for which p_i is the farthest site. Currently the best known algorithms for computing the

order-k Voronoi diagram run in $O(nlog^3n + nk)$ time and in $O(nlogn + nk2^{clog^*k})$ time, where c is a constant [11].

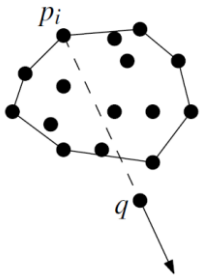

Figure 2. Farthest-Point Voronoi diagram [11]

Consider *x* and *y* as two distinct elements of *P*. A set of points construct a cell in the second order Voronoi diagram for which the nearest and the second nearest neighbors are x and y. Second order Voronoi diagram can be used when we are interested in the two closest points, and we want a diagram to captures that.

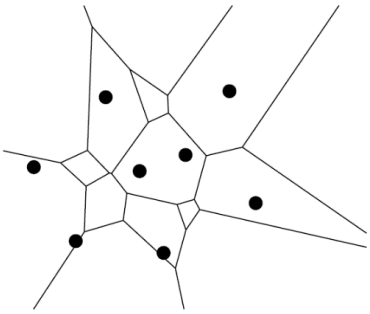

Figure 3.An instant of HOVD [11]

4. SUGGESTED ALGORITHM

As mentioned before, one of the best algorithms to construct Voronoi diagram is Fortune algorithm. Furthermore HOVD can be used to find k-nearest neighbors [12]. D.T. Lee used an $O(k^2nlogn)$ algorithm to construct a complete HOVD to obtain nearest neighbors. In D.T. Lee's algorithm, at first the first order Voronoi diagram is obtained, and then finds the region of diagram that contains query point. The point that is in this region is defined as a first neighbor of query point. In the next step of Lee's algorithm, this nearest point to the query will be omitted from dataset, and this process will be repeated. In other words, the Voronoi diagram is built on the rest of points. In the second repetition of this process, the second neighbor is found and so on. So the nearer neighbors to a given query point are found sequentially.

However we think that nearest neighbors can be finding without completing the process of HOVD construction. More precisely, in Lee's algorithm each time after omitting each nearest neighbor, next order of Voronoi diagram is made completely (edges and vertices) and then for computing a neighbor performs the search algorithm. In contrast, in our algorithm, the vertices of Voronoi diagram are only computed and the neighbors of the query are found during process of vertices computing. So in our algorithm, the overhead of edge computing to find neighbors is effectively omitted. As we will show later in this paper, by eliminating this superfluous computation a more efficiently algorithm in term of time complexity will be obtained.

We use Fortune algorithm to create Voronoi diagram. Because of space limitation in this paper we don't describe this algorithm and the respectable readers can refer to [11, 13]. By moving sweep line in Fortune algorithm, two set of events are emerged; site event and circle event [11]. To find k nearest neighbors in our algorithm, the developed circle events are employed. There are specific circle events in the algorithm that are not actual circle events named false alarm circle events. Our algorithm (see the next section) deals efficiently with real circle events and in contrast doesn't superfluously consider the false alarm circle event. A point on the plane is inside a circle when its distance from the center of the circle is less than radius of the circle. The vertices of a Voronoi diagram are the center of encompassing triangles where each 3 points (sites) constitute the triangles. The main purpose of our algorithm is to find out a circle in which the desired query is located.

As the proposed algorithm does not need pre-processing, it's completely appropriate for dynamic environment where we can't endure very time consuming pre-processing overheads. Because, as the readers may know, in k-NN search methods a larger percent of time is dedicated to constructing a data structure (usually in the form of a tree). This algorithm can be efficient, especially when there are a large number of points while their motion is considerable.

4.1 HOV-kNN algorithm

After describing our algorithm in the previous paragraph briefly, we will elaborate it formally in this section. When the first order Voronoi diagram is constructed, some of the query neighbors can be obtained in complexity of the Fortune algorithm (i.e.$O(nlogn)$). This fact forms the first step of our algorithm. When the discovered circle event in *HandleCircleEvent* of the Fortune algorithm is real (initialized by the variable "*check*" in line 6 of the algorithm, and by default function *HandleCircleEvent* returns "*true*" when circle even is real) the query distance is measured from center of the circle. Moreover, when the condition in line 7.i of the algorithm is true, the three points that constitute the circle are added to *NEARS* list if not been added

before (function *PUSH-TAG (p)* shows whether it is added to *NEAR* list or not).

```
1)  Input : q , a query
2)  Output: list NEARS, k nearest neighbors.
3)  Procedure :
4)  Initialization :
5)  NEARS ={}, K nearest neighbors
      , Check = false, MOD = 0, V = {} (hold Voronoipoints）;
6)  Check = HandleCircleEvent()
7)  If check= true, then -- detect a true circle event.
    i)   If distance(q , o) < r  Then
         (1)  If PUSH-TAG(p1) = false , Then
              (a)  add  p1  to NEARS
         (2)  If  PUSH-TAG (p2) =  false , Then
              (a)  add p2 to NEARS
    ii)  If  PUSH-TAG(p3) = false, Then
              (a)  add p3 to NEARS
```

Real circle events are discovered up to this point and the points that constitute the events are added to neighbor list of the query. As pointed out earlier, the preferred result is obtained, if "k" inputs are equal or lesser than number of the obtained neighbors a$O(nlogn)$complexity.

```
8)  if  SIZE (NEARS) >= k , then
        a.  sort (NERAS )  - - sort NEARS by distance
        b.  for  i = 1 to k
                i.  print (NEARS);
9)  else if SIZE (NEARS) = k
                ii.  print(NEARS);
```

The algorithm enters the second step if the conditions of line 8 and 9 in the first part are not met. The second part compute vertices of Voronoi sequentially, so that the obtained vertices are HOV vertex. Under sequential method for developing HOV [12], the vertices of the HOV are obtained by omitting the closer neighbors. Here, however, to find more neighbors through sequential method, loop one of the closest neighbor and loop one of the farthest neighbor are deleted alternatively from the set of the point. This leads to new circles that encompass the query. Afterward, the same calculations described in section one are carried out for the remaining points (the removed neighbors are recorded a list named *REMOVED_POINTS*). The calculations are carried out until the loop condition in line 5 is met.

```
10) Else if (SIZE(NEARS) < k )
        c.  if  mod MOD 2 = 0 , then
                i.  add nearest_Point to REMOVED_POINT ;
                ii.  Remove(P,nearest_Point);
        d.  if  mod MOD 2 = 1 , then
```

i. add farthest_Point to REMOVED_POINT ;
ii. Remove(P,nearest_Point);

11) Increment MOD ;
12) produce line 6 to 9 from part1 for remind points P ;
13) Repeat until k >= SIZE _ LIST (NEARS) + SIZE _ LIST (REMOVED_POINT) ;
14) PRINT (NEARS) ;

Should the number of neighbors be less than required number of neighbors, the algorithm starts the third part. At this part, Voronoi vertices and their distance from query are recorded in a list. As explained for the first part of the algorithm, the Voronoi vertices in the Fortune algorithm and their distance to the query are enough to check realization of the condition of line 8. The vertices and their distance to the query are recorded. Following line will be added after line 7 in the first part:

add pair(Voronoi_Vertex ,distance_To_Query) to List V

Moreover, along with adding input point to the list of the neighbors, their distance to the query must be added to the list.

Using these two lists (after being filled, the lists can be ranked based on their distance to query) the nearest point or Voronoi vertices is obtainable. The nearest point can be considered as the input query and the whole process of 1st and 2nd parts of the algorithm is repeated until required number of neighbors is achieved. Finally, to have more number of neighbors, the method can be repeated sequentially over the closer points to the query. This part of the algorithm has the same complexities of the two other sections as the whole process to find the preliminary query is repeated for the representatives of the query.

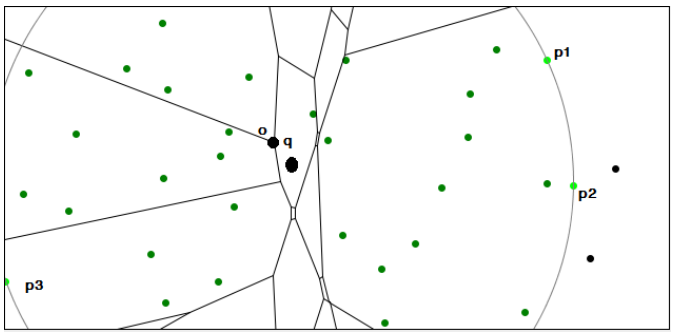

Figure 4.implementation of HOVD

In Figure 4 "*o*" is a vertex of Voronoi and a center point of circle event that is created by p_1, p_2 and p_3. Based on algorithm the circle that encompasses the query, add p_1, p_2 and p_3 points as neighbors of query to the neighbors' list. Here *k* is near to *n*, so by computing higher order of Voronoi, the circle will be bigger and bigger. Thus farther neighbors are added to query neighbors' list.

4.2 The complexity of HOV-kNN

As mentioned before, HOV-kNN algorithm has a time complexity lesser than the time complexity of D.T. Lee's algorithm. To show this fact, consider the presented algorithm in the previous section. Line 13 explains that the main body of algorithm must be repeated k times in which "k" are the number of neighbors that should be found. In each repetition one of the query's neighbors are detected by algorithm and subsequently eliminated from dataset. The principle part of our algorithm that is the most time consuming part too is between lines 6 and 9. This line recalls modified Fortune algorithm which has a time complexity$O(nlogn)$. Therefore the overall complexity of our algorithm will be:

$$\sum_{i=1}^{k} O(nlogn) = O(nlogn)\sum_{i=0}^{k} 1 = kO(nlogn) = O(knlogn) \qquad (2)$$

In comparison to the algorithm introduced in [12] (which has a time complexity$O(k^2nlogn)$) our algorithm is faster k times. The main reason of this difference is that Lee's algorithm completely computes the HOVD, while ours exploits a fraction of HOVD construction process. In term of space complexity, the space complexity of our algorithm is the same as the space complexity of Fortune algorithm: $O(n)$.

5. IMPLEMENTATION AND EVALUATION

This section introduces the results of the HOV-kNN algorithm and compares the results with other algorithms. We use Voronoi diagram which is used to find k nearest neighbor points that is less complicated. The proposed algorithm was implemented using C++. For maintaining data points vector data structure, which is one of the C++ standard libraries, was used. The input data points used in the program test were adopted randomly. To reach preferred data distribution, not too close/far points, they were generated under specific conditions. For instance, for 100 input points, the point generation range is 0-100 and for 500 input points the range is 0-500. To ensure accuracy and validity of the output, a simple kNN algorithm was implemented and the outputs of the two algorithms were compared (equal input, equal query). Outputs evaluation was also carried out sequentially and the outputs were stored in two separate files. Afterward, to compare similarity rate, the two files were used as input to another program.

The evaluation was also conducted in two steps. First the parameter "k" was taken as a constant and the evaluation was performed using different points of data as input. As pictured in Figure 5, accuracy of the algorithm is more than 90%. In this diagram, the number of inputs in dataset varies between 10 and 100000. At the second step, the evaluation was conducted with different values of k, while the number of input data was stationary. Accuracy of the algorithm was obtained 74% while "k" was between 10 and 500 (Figure 6).

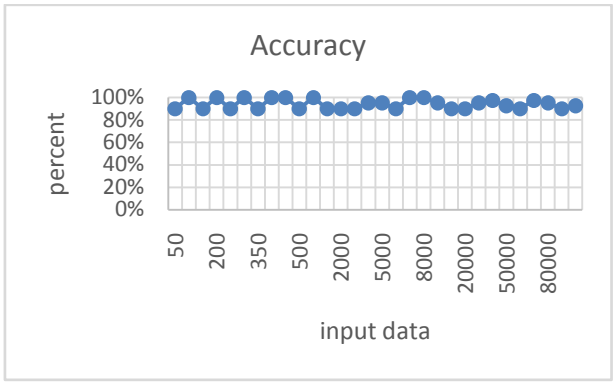

Figure 5. The accuracy of the algorithm for constant k and different points of data as input

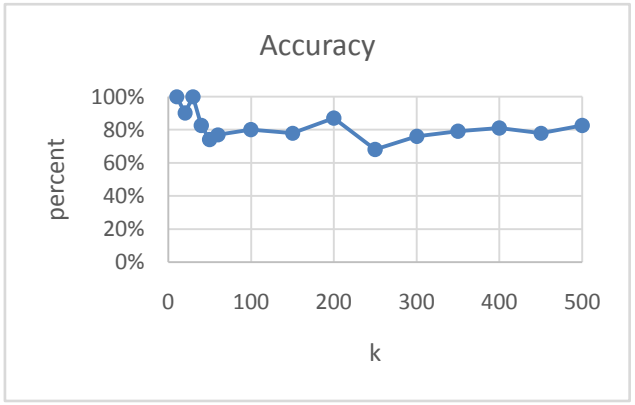

Figure 6. The accuracy of the algorithm for variable k and constant data as input

6. CONCLUSION AND FUTURE WORK

We have introduced a new algorithm (named HOV-kNN) with time complexity $O(knlogn)$ and computing order k Voronoi diagram to find k nearest neighbor in a set of N points in Euclidean space. The new proposed algorithm finds k nearest neighbors in two stages: 1) during constructing the first order Voronoi diagram, some of the query neighbors can be obtained in complexity of the Fortune algorithm; 2) computing vertices of Voronoi sequentially. Because of eliminating pre-processing steps, this algorithm is significantly suitable for dynamic space in which data points are moving. The experiments are done in twofold: 1) constant number of data points while k is variable, and 2) variable number of data points while k is constant. The obtained results show that this algorithm has sufficient accuracy to be applied in real situation. In our future work we will try to give a parallel version of our algorithm in order to efficiently implementation a parallel machine to obtain more speed implementation. Such an algorithm will be appropriate when the numbers of input points are massive and probably distributed on a network of computers.

REFERENCES

[1] Lifshits, Y.Nearest neighbor search: algorithmic perspective, *SIGSPATIAL Special*. Vol. 2, No 2, 2010, 12-15.

[2] Shakhnarovich, G., Darrell, T., and Indyk, P.Nearest Neighbor Methods in Learning and Vision: Theory and Practice, The MIT Press, United States, 2005.

[3] Andoni, A.Nearest Neighbor Search - the Old, the New, and the Impossible, Doctor of Philosophy, Electrical Engineering and Computer Science, Massachusetts Institute of Technology,2009.

[4] Bhatia, N., and Ashev, V. Survey of Nearest Neighbor Techniques, *International Journal of Computer Science and Information Security*, Vol. 8, No 2, 2010, 1- 4.

[5] Dhanabal, S., and Chandramathi, S. A Review of various k-Nearest Neighbor Query Processing Techniques, *Computer Applications*, Vol. 31, No 7, 2011, 14-22.

[6] Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., and Zhang, H.Fast approximate nearest-neighbor search with k-nearest neighbor graph, *In Proceedings of 22 international joint conference on Artificial Intelligence*, Vol. 2 (IJCAI'11), Toby Walsh (Ed.), 2011, 1312-1317.

[7] Fukunaga, K. Narendra, P. M. A Branch and Bound Algorithm for Computing k-Nearest Neighbors, *IEEE Transactions on Computer*,Vol. 24, No 7, 1975, 750-753.

[8] Song, Z., Roussopoulos, N. K-Nearest Neighbor Search for Moving Query Point, *In Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases* (Redondo Beach, California, USA), Springer-Verlag, 2001, 79-96.

[9] Güting, R., Behr, T., and Xu, J. Efficient k-Nearest Neighbor Search on moving object trajectories, *The VLDB Journal* 19, 5, 2010, 687-714.

[10] Frentzos, E., Gratsias, K., Pelekis, N., and Theodoridis, Y.Algorithms for Nearest Neighbor Search on Moving Object Trajectories, *Geoinformatica* 11, 2, 2007,159-193.

[11] Berg, M. , Cheong, O. , Kreveld, M., and Overmars, M.*Computational Geometry: Algorithms and Applications*, Third Edition, Springer-Verlag, 2008.

[12] Lee, D. T. On k-Nearest Neighbor Voronoi Diagrams in the Plane, *Computers, IEEE Transactions on Volume:C-31*, Issue:6, 1982, 478–487.

[13] Fortune, S. A sweep line algorithm for Voronoi diagrams, *Proceedings of the second annual symposium on Computational geometry*, Yorktown Heights, New York, United States, 1986, 313–322.

[14] Meyerhenke, H. Constructing Higher-Order Voronoi Diagrams in Parallel, *Proceedings of the 21st European Workshop on Computational Geometry*, Eindhoven, The Netherlands, 2005, 123-126.

[15] Gao, Y., Zheng, B., Chen, G., and Li, Q. Algorithms for constrained k-nearest neighbor queries over moving object trajectories, *Geoinformatica* 14, 2 (April 2010), 241-276.

This paper may be cited as:

Abbasifard, M. R., Naderi, H. and Mirjalili, M., 2014. HOV-kNN: A New Algorithm to Nearest Neighbor Search in Dynamic Space. *International Journal of Computer Science and Business Informatics, Vol. 9, No. 1, pp. 12-22*.