Nearest neighbor search through function minimization

Chang Shu, Michael Greenspan, Guy Godin

Institute for Information Technology
National Research Council Canada
Montreal Road, Bldg. M50
Ottawa, Ontario, Canada K1A 0R6

Abstract

This paper describes a solution to the nearest neighbor problem. The proposed algorithm, which makes use of the triangle inequality property, is considered from a function minimization perspective. The distance function is regularized through the computation of distance to a reference point; an initial starting point is rapidly found, and used in an iterative refinement using search over a sorted list. The algorithm is described, and simulation results are provided that suggest better performance than that obtained with spatial partition techniques such as Elias and k-d tree, for moderate size point sets.

1 Introduction

Nearest neighbor search is a classical problem in computational geometry. Given a set P of n points in R^d and a query point q, the nearest neighbor problem is to find point p_c in P such that $\|p_c - q\| \leq \|p_i - q\|$ for any p_i in P. The solution to this problem is important in various application areas including pattern recognition, robotics, signal compression, etc. These applications repeatedly query nearest neighbors on large point sets and thus demand a highly efficient algorithm. Almost all solutions involve preprocessing the points in some way such that less than n distance computations are needed at query time. The dominant solution method so far is the k-d tree [1]. This method subdivides the whole space recursively with hyper-planes and arranges points into cells. The k-d tree structure can be constructed in $O(n \log n)$ time and takes $O(n)$ in space. Friedman et al. [5] reported $O(\log n)$ expected time for finding the nearest neighbor.

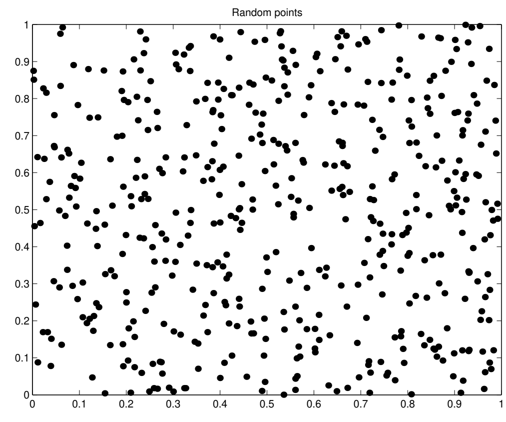
An alternative method, called *Elias algorithm* [8], embeds the points in a regular grid. The search starts from the grid element containing the query point and concentrically searches points in the surrounding grid elements until the distance to all remaining grid elements is guaranteed to be larger than the current shortest distance. This method is expensive in space but very efficient in time, especially for quasi-uniformly distributed point sets.

A few authors suggested the use of the triangle inequality property for points in metric spaces to prune the number of distance computations [9]. This approach leads to extremely simple implementations. Recently, Greenspan et al. [7] reported experimental evidence that triangle inequality based methods can be superior to the k-d tree method, especially for moderate sized point sets. This method was recently applied [6] to improve the performance of the ICP algorithm for registering 3D images [2].

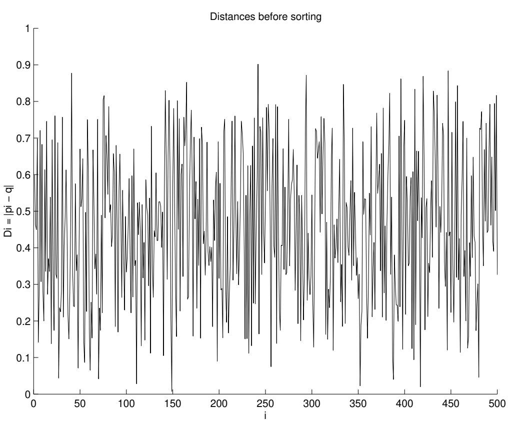
Most existing algorithms for the nearest neighbor problem use spatial partitioning data structures. They are suitable for problems in which the data sets are static and the number of queries is far greater than the size of the data set. However, many applications, such as imaging and graphics, involve changing data of moderate size, and they therefore require algorithms that have low overhead.

In this paper, we extend the results of [7] and present the algorithm from a new perspective. The main idea is to formulate the problem as a function minimization problem. This leads to an algorithm with minimal preprocessing and simple data structures. Although it has a poor worst case complexity ($O(n)$), its average time performance compares favorably with that of the k-d tree and Elias methods for small and moderately sized point sets. Other advantages include its capability of handling dynamic data and higher dimensional data.

Our method bears some similarities to Friedman et al. [4] which projects multidimensional points into one dimension. However, their method does not rely on



(a)



(b)

Figure 1: Distance function for 500 random points

triangle inequality.

2 Nearest neighbor search as function minimization

Consider the distance function $F(i) = \|p_i - q\|$ where q is the query point. The nearest neighbor problem is equivalent to finding index i_m such that $F(i_m)$ is the minimum of function F. For randomly distributed points, this function is highly irregular. Figure 1 shows a plot of F for 500 random 2D points. This function is obviously hard to minimize, and we naturally want to regularize function F so that it behaves smoother. We can do this through re-parameterization.

Let r be an arbitrary reference point in P. Let function $R(i) = \|p_i - r\|$ be defined as the function of distance from each point p_i to the reference point r. Let π be a permutation of $\{1, \ldots, n\}$, such that $R(\pi(k)) \leq R(\pi(l))$ if $k < l$. From π, we obtain a new function $G(i) = F(\pi(i))$. Plotted in Figure 2(a), we can see that function G is much better behaved than function F, though it is still not smooth.

Let us now consider another function $\delta(i) = |R(\pi(i)) - R_q|$ where $R_q = \|q - r\|$ is the distance between the query point and the reference point. The following property is well known.

Property 1 (Triangle Inequality) *For any points p, q, and r in R^d*

$$\Big| \|p - r\| - \|q - r\| \Big| \leq \|p - q\| \leq \|p - r\| + \|q - r\|$$

The following propositions follow directly from the triangle inequality.

Proposition 1 *For any i in $\{1 \ldots n\}$*

$$\delta(i) \leq G(i)$$

Proposition 2 (Triangle Constraint)

$$\delta(i) > G(j) \Longrightarrow G(i) > G(j)$$

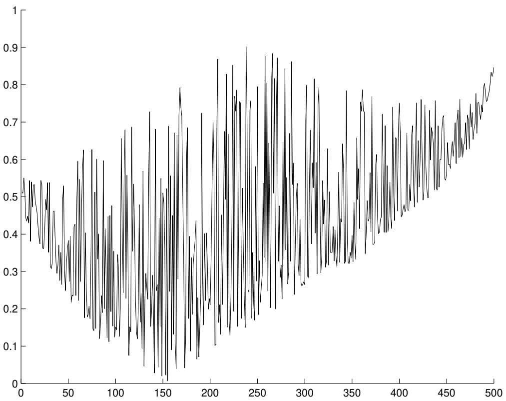
Figure 2(b) shows G and δ plotted together in the same graph. δ is a much smoother function than G, and it bounds G from below. The following Ordering Theorem shows that δ consists of two monotonic parts: monotonically decreasing on the left, and monotonically increasing on the right, separated by the minimum point $min\{\delta(i)\}$.

Theorem 1 (Ordering Theorem) *For any c and l in $\{1, \ldots, n\}$, if $\delta(l) > G(c)$, then*

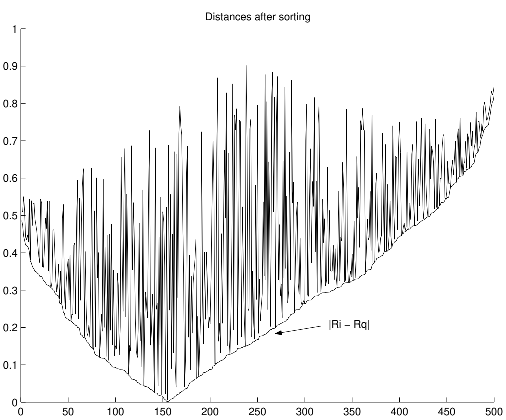
1. *if $l > c$, then $G(k) > G(c)$ for all $k > l$;*
2. *if $l < c$, then $G(k) > G(c)$ for all $k < l$.*

3 Triangle inequality nearest neighbor algorithm

We observe that the minimum of the function G, which is the nearest neighbor we are looking for, is very close to the minimum of the function δ. This suggests that we can use δ to guide the search for the true minimal point on G. Based on this observation, we can use the minimal point of $\delta(i)$ as a starting point for searching the minimal point of G.



(a)



(b)

Figure 2: Distance function after sorting

A novel solution to the nearest neighbor problem based upon the Ordering Theorem, which we call the Triangle Inequality Nearest Neighbor (TINN) method is as follows.

1. The point set is first preprocessed by calculating $R_i = \|p_i - r\|$ for all $p_i \in P$. The results are organized into a list $\mathbf{L}$ such that each list element L_i references a point and its distance to the reference point, i.e. $L_i = \{p_i, R_i\}$. The list is then ordered by increasing R_i. We will assume in the subsequent description that the subscript enumerates the ordering of $\mathbf{L}$, i.e. $i < j \iff R_i < R_j$.

2. At run time, the value R_q is determined for the query point q. The list element whose distance to the reference point is closest to R_q is then identified and denoted as the *pivot* element L_k. This can be accomplished efficiently as a binary search over $\mathbf{L}$. The distance D_k between the query point and the pivot point is calculated, and the closest point is temporarily assigned as the pivot point, i.e. $p_c \leftarrow p_k$, $D_c \leftarrow D_k$.

3. Starting at the next lowest neighbor p_{k-1} of p_k, we then compare the value of $|R_{k-1} - R_q|$ with D_k. If $|R_q - R_{k-1}| \le D_c$, we then calculate D_{k-1}. If $D_{k-1} < D_c$, then we reset the temporary closest point to p_{k-1}, i.e. $p_c \leftarrow p_{k-1}$, $D_c \leftarrow D_{k-1}$. Otherwise, if $|R_q - R_{k-1}| > D_c$, then there is no need to calculate D_{k-1} as Proposition 2 guarantees that its value will exceed D_c and the p_{k-1} cannot be the closest point. Furthermore, it is unnecessary to consider any other list elements in the same direction, as the ordering of $\mathbf{L}$ ensures that $|R_i - R_q|$ will increase for all monotonic sequences of i on the same side of L_k.

4. Step 3 is repeated for all neighbors in the same direction (i.e. $P_{k-h}, h > 1$), until either the stopping condition is met ($|R_{k-h} - R_q| > D_c$), or the top of $\mathbf{L}$ is reached.

5. Steps 3 to 4 are repeated for the L_i on the other side of the orginal pivot element (i.e. $p_{k+h}, h \ge 1$).

Generating $\mathbf{L}$ takes $O(n \log n)$ time, requiring only a single pass through P to determine the R_i, followed by a sort. The storage cost is extremely modest, comprising an extra float value per point, plus a single reference point r. Using binary search, the pivot point can be found in $O(\log n)$ time. This can be improved by using parabolic interpolation on the function δ. The method also trivially extends to a solution of the k nearest neighbors problem.

4 Performance analysis

The worst case complexity of the TINN algorithm is $O(n)$. This is the case when all points lie on a sphere and the reference point lies at the center of the sphere. In practice, this case rarely happens since we pick the reference point randomly.

As a step toward the goal of estimating the average case performance of the algorithm, we are interested in estimating the number of distance computations once we have found the minimal point $min\{\delta(i)\}$. Our observation that $min\{\delta(i)\}$ is not too far from $min\{G(i)\}$ is supported by the following experiment.

Here we provide a Monte Carlo simulation for $n \le 500$. For each number n in $\{1, \ldots, 500\}$, we generate

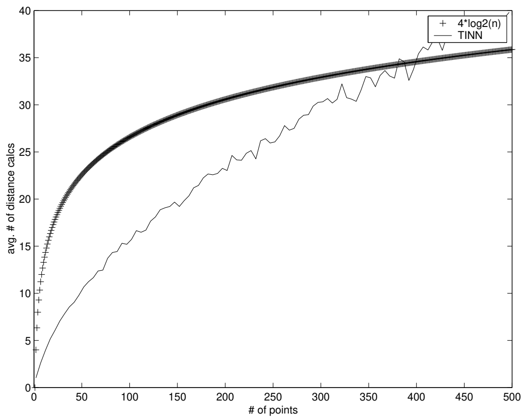


Figure 3: Monte Carlo simulation results

10^3 configurations of P, each containing n points randomly uniformly distributed on a unit square. For each configuration, a query point q was randomly generated with the same distribution, and the nearest neighbor of q was found. The average number of times that the actual distance was computed was measured for each configuration.

Figure 3 plots the results of this simulation. For comparison, we also plot the function $\log_2(n)$ with a constant which reflects the analysis of the expected number of distance computations for k-d tree and Elias methods (2^k, where k is the dimension of the problem under consideration [3]). TINN is observed as outperforming these logarithmic algorithms for $n < 400$, a useful range of sizes in practice.

5 Conclusions

We have presented a solution to the nearest neighbor problem based on the triangle inequality property, and we have formulated the problem as a function minimization problem. Our method differs from traditional solutions which focus on spatial partitioning. Instead, we apply a regularization process to smooth the distance function so that a good initial point can be found as a starting point for an iteration process. A Monte Carlo simulation suggests that the TINN algorithm is superior to k-d tree and Elias methods for moderate sized point sets. Compared to the projection approach proposed by Friedman et al. [4], our method offers restricted search space. As the only data structure used in the TINN method is a sorted linear list, dynamic data sets can be handled easily. Point insertion and deletion can be performed in $O(\log n)$ time. Also, the method can be easily extended to higher dimensions.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] Paul J. Besl and Neil D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[3] John Gerald Cleary. Analysis of an algorithm for finding nearest neighbors in euclidean space. *ACM Transcations on Mathematical Software*, 5(2):183–192, 1979.

[4] Jerome H. Friedman, Forest Baskett, and Leonard J. Shustek. An algorithm for finding nearest neighbors. *IEEE Trans. Compters*, C-24:1000–1006, 1975.

[5] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transcations on Mathematical Software*, 3(3):209–226, 1977.

[6] Michael Greenspan and Guy Godin. A nearest neighbor method for efficient ICP. In *Proceedings of the 3rd International Conference on 3-D Digital Imaging and Modeling, Quebec City, Canada*, pages 161–168, May 2001.

[7] Michael Greenspan, Guy Godin, and Jimmy Talbot. Acceleration of binning nearest neighbor methods. In *Proceedings of Vision Interface 2000, Montreal, Canada*, pages 337–344, May 2000.

[8] Ronald L. Rivest. On the optimality of Elias's algorithm for performing best-match searches. *Information Processing*, 74:678–681, 1974.

[9] Enrique Vidal Ruiz. An algorithm for finding nearest neighbors in (approximately) constant time. *Pattern Recognition Letters*, 4:145–157, 1986.