An Improved Hierarchical Datastructure for Nearest Neighbor Search

Mengdie Nie,¹, ² Zhi-Jie Wang,¹, ², ³ Chunjing Gan,¹, ² Zhe Quan,² Bin Yao,³ Jian Yin¹, ²
¹Sun Yat-Sen University, ²Hunan University, ³Shanghai Jiao Tong University.
⁴Guangdong Key Laboratory of Big Data Analysis and Processing
{niemd, ganchj3} @mail2.sysu.edu.cn, quanzhe@hun.edu.cn, yaobin@cs.sjtu.edu.cn, {wangzhij5, issjyin} @mail.sysu.edu.cn

Abstract

Nearest neighbor search is a fundamental computational tool and has wide applications. In past decades, many datastructures have been developed to speed up this operation. In this paper, we propose a novel hierarchical datastructure for nearest neighbor search in moderately high dimension. Our proposed method maintains good run time guarantees, and it outperforms several state-of-the-art methods in practice.

Introduction

Nearest neighbor (NN) search is a basic computational tool that can be applied to many domains. The datastructure utilized plays a key role in NN search, and it can be used to speed up the kNN classification algorithm and many other tasks such as reinforcement learning (Izbicki and Shelton 2015). The basic NN problem is as follows: Given a set S of n points in some metric space (X, d), the problem is to preprocess S so that given a query point q ∈ X, one can find efficiently a point p ∈ S which minimizes d(q, p).

The naive method involves a linear scan of all the data points and takes time O(n). So far, many datastructures such as kd-tree have been developed to speed up this process. There are also some more complicated and powerful datastructures like the metric skip list (Karger and Ruhl 2002) and the navigating net (Krauthgamer and Lee 2004). Later, Beygelzimer, Kakade, and Langford (2006) proposed the cover tree (CT) — a leveled tree where each level is a “cover” for the level beneath it. It is a hierarchical datastructure that simplifies navigating nets while it maintains good run time guarantees (consuming linear space and logarithmic time). Recently, (Izbicki and Shelton 2015) developed the simplified nearest ancestor cover tree (SNACT), which provides a simpler definition, reducing the number of nodes from O(n) to exactly n. Moreover, it introduces an “additional” invariant, i.e., nearest ancestor invariant, that makes queries faster in practice. This paper revisits cover tree structures and develops a new method that achieves the same run time guarantees while it significantly outperforms them in practice.

The Proposed Method

Following (Izbicki and Shelton 2015), we use level(p), children(p) and descendants(p) to denote the level, children and descendants of node p, respectively. Generally, similar to SNACT, our method also maintains several invariants (i.e., leveling, covering, separating, and nearest ancestor invariants) when constructing our structure. Instead, we introduce several other important concepts to enhance the pruning ability, and so improve the search efficiency.

One of our ideas is to use a concept called range list, denoted by rl. The jth (j ≥ 0) element in rl is computed as

$$r_l[j] = \arg\max_{q \in \text{descendants}(p)} d(p, q)$$

The equation above implies that rl[j] stores the maximum distance from p to its descendants whose levels are in the corresponding range. For any point p′, let rlfinal denote the final element in rl, and assume p is the current NN (found so far) of q. Our range list has the following advantages: (i) when d(p′, q) − rlfinal ≥ d(p, q), it can help us prune the whole subtree rooted at p′; and (ii) otherwise, we may find an appropriate j such that d(p′, q) − rl[j] ≥ d(p, q) > d(p′, q) − rl[j + 1], thereby we can prune nodes (in the subtree) whose levels are in [level(p′) − 1 − j, level(p′)].

Another main idea is to introduce the concept of opposite quadrant. One can easily understand that the horizontal and vertical axes of a d-dimensional Cartesian coordinate system divides the data space into 2d parts, numbered as 0 to 2d − 1. For a node p, denote by parent(p) its parent node. We can view parent(p) as the original point (of the coordinate system), and then define p’s quadrant information below.

$$p_1 = \sum_{i=1}^{d} 2^i, \text{ s.t. } p[i] < 0$$

The equation above essentially accumulates all 2i that can satisfy p[i] < 0, where p[i] refers to the ith dimension value of point p. Given any other point p′, we say p and p′ are in the opposite quadrant if and only if p1 + p′1 = 2n − 1. This concept is helpful for us to prune unqualified nodes, since
one can replace $p'$ with the query point $q$, and then all points satisfying the above condition can be pruned safely.

Besides, we introduce the vectorial angle cosine to further enhance the pruning ability. Assume $p$ is the current NN of query point $q$, one can imagine that there exists a hyperplane $P$ that is vertical to segment $pq$ and passes through $p$. This implies that $P$ divides the data space into two parts. Clearly, for any point $p'$ located in the part that does not contain $q$, it is never to be the NN of $q$. This can be determined by computing the vectorial angle cosine defined as $\cos \theta = \frac{\vec{pq} \cdot \vec{p'q'}}{|\vec{pq}| \cdot |\vec{p'q'}|}$. It is obvious (by analytic geometry) that, if $\cos \theta < 0$, then $p'$ can be pruned safely.

The above ideas can be easily integrated into the SNACT. In brief, one can calculate the range list and quadrant information when the tree is constructed, and then attach them as additional attributes to corresponding nodes. Later, we can leverage the new hierarchical datastructure containing these additional attributes to assist us to perform NN search. Note that, the vectorial angle cosine and the quadrant information $q_i$ of query point $q$ are calculated during the query. The paradigm of performing NN search is basically similar to that of SNACT. That is, we start from the root and keep track of the current NN from a subset $S_i$ that may contain the NN of $q$, and then this process iteratively constructs $S_{i-1}$ by expanding $S_i$ to its children. In the process of expanding $S_i$, some unqualified children are pruned based on heuristics designed above. The rest of steps are the same to that in SNACT. Note that, in SNACT the major heuristic is the $\maxdist(p)$, whose function is equal to $\text{ advantage}(i)$ of our range list.

One can easily understand that our modifications to SNACT do not change the breadth of the non-leaf node and the depth of the tree. Namely, it is same to the SNACT: every node can have at most $O(c^4)$ children and the depth of the tree is bounded by $O(c^2 \log n)$. Thus, the runtime bound of our solution is also $O(c^6 \log n)$, where $n$ is the number of data points, and $c$ is the expansion constant.

### Experiments and Results

To evaluate our solution, we use three benchmark datasets obtained from http://archive.ics.uci.edu/ml/index.php. They are yearpredict (515,345 points with 90 dimensions), corel (68,040 points with 32 dimensions), artificial40 (10,000 random generated points with 40 dimensions), respectively.

We compare the proposed method (PM) with two competitors: CT and SNACT. Following (Izbicki and Shelton 2015), we use “all nearest neighbour search” to study the query performance and also normalize the query cost by the baseline. Remark that “all nearest neighbor search” refers to searching the nearest neighbor for each point in the dataset. Furthermore, to study the effectiveness of the proposed techniques, we implement several other algorithms, which are the variants of SNACT: (i) QI, which employs the quadrant information; (ii) RI, which employs the range list information; (iii) DI, which employs the direction information (including quadrant and vectorial angle). All methods are implemented in C++, and are executed on a machine with an Intel(R) Core(TM) CPU @2.40GHZ and 64GB RAM.

### Conclusion

In this paper we suggested a new method for nearest neighbor search in moderately high dimension. We conducted empirical study on three benchmark datasets. The experimental results consistently demonstrate that our method is efficient and competitive, compared against the competitors.

### References


