An Improved Hierarchical Datastructure for Nearest Neighbor Search*

Mengdie Nie[†,⊤], Zhi-Jie Wang[†,⊤], Chunjing Gan[†,⊤], Zhe Quan[‡], Bin Yao[§], Jian Yin[†,⊤]
[†] Sun Yat-Sen University, [‡] Hunan University, [§] Shanghai Jiao Tong University
[⊤] Guangdong Key Laboratory of Big Data Analysis and Processing.
{niemd, ganchj3}@mail2.sysu.edu.cn, quanzhe@hnu.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 (Beygelzimer, Kakade, and Langford 2006). 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 dimensionality reduction and reinforcement learning (Izbicki and Shelton 2015). The basic NN problem is as follows: Given a set $\mathcal{S}$ of n points in some metric space $(\boldsymbol{X}, d)$, the problem is to preprocess $\mathcal{S}$ so that given a query point $q \in \boldsymbol{X}$, one can find efficiently a point $p \in \mathcal{S}$ 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 and ball 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 work was partially supported by the National Key R&D Program of China (2018YFB1004400), and the NSFC (61472453, U1401256, U1501252, 61602166, U1611264, U1636210, U1711261, 61729202, U1711262, 61872235, 61832017).

In this paper, we revisit cover tree structures and develop 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 $\texttt{level}(p)$, $\texttt{children}(p)$ and $\texttt{descendants}(p)$ 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 introduce a concept called the range list, denoted by rl. The jth ($j \geq 0$) element in rl is computed as

$$rl[j] = \underset{\substack{q \in descendants(p) \\ level(p) - 1 - j \leq level(q) \leq level(p) - 1}}{\operatorname{argmax}} d(p, q)$$

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 rl_{final} 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 $d(p', q) - rl_{final} \geq d(p, q)$, it can help us prune the whole subtree rooted at p'; and (ii) otherwise, we may find an appropriate j such that $d(p', q) - rl[j] \geq d(p, q) > d(p', q) - rl[j+1]$, 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 2^d parts, numbered as 0 to $2^d - 1$. For a node p, denote by $\texttt{parent}(p)$ its parent node. We can view $\texttt{parent}(p)$ as the original point (of the coordinate system), and then define p's quadrant information below.

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

The equation above essentially accumulates all 2^i 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 $p_\dagger + p'_\dagger = 2^n - 1$. 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 a concept, named vectorial angle cosine, to further enhance the pruning ability. Assume that p is the current NN (found so far) of query point q, one can imagine that there exists a hyperplane $\mathbb{P}$ that is vertical to segment $\overline{pq}$ and passes through p. This implies that $\mathbb{P}$ divides the data space into two parts. Clearly, for any point p' located in the part that does not contain point q, it is never to be the NN of q. This can be determined by computing the vectorial angle cosine defined as $vac = cos\theta = \frac{\vec{pq} \cdot \vec{pp'}}{|\vec{pq}| \times |\vec{pp'}|}$. It is obvious (by analytic geometry) that, if $vac < 0$, 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 the 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 $q_\dagger$ 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 the 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 *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 investigate 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.

Table 1: The comparison results on three benchmark datasets. The query cost is normalized by CT.

Dataset \ Method	CT	SNACT	PM
artificial40	1.00	0.81	**0.62**
yearpredict	1.00	0.56	**0.38**
corel	1.00	0.59	**0.33**

Table 1 reports the comparison results of three methods. From this table, it can be seen that PM has less query cost compared against two competitors on all these three datasets. More specifically, compared to CT, our method can reduce the query cost by $38\% \sim 67\%$. Even for the stronger competitor SNACT, our method can reduce the query cost by $24\% \sim 46\%$. These evidences essentially demonstrate the competitiveness of our proposed method.

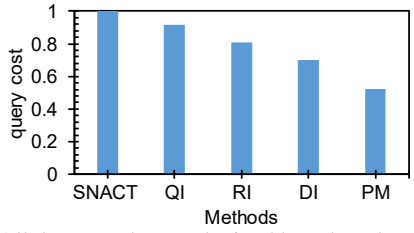

Figure 1: All these results are obtained based on the corel dataset. The query cost is normalized by SNACT.

Figure 1 shows the comparison results of SNACT, QI, RI, DI, and PM on the corel dataset. It can be seen that QI, RI, and DI achieve about 8%, 20%, 30% speed ups, respectively. This essentially demonstrates the effectiveness of these proposed strategies. On the other hand, one can see that PM, which integrates all these proposed strategies, achieves the best performance result (about 46% speed up, compared against SNACT). This further reflects the efficiency and effectiveness of our proposed method.

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

Beygelzimer, A.; Kakade, S.; and Langford, J. 2006. Cover trees for nearest neighbor. In *ICML*, 97–104.

Izbicki, M., and Shelton, C. R. 2015. Faster cover trees. In *ICML*, 1162–1170.

Karger, D. R., and Ruhl, M. 2002. Finding nearest neighbors in growth-restricted metrics. In *STOC*, 741–750.

Krauthgamer, R., and Lee, J. R. 2004. Navigating nets: simple algorithms for proximity search. In *SODA*, 798–807.