Fast Neighborhood Graph Converge based on Orientation Sensitive Hashing

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Converge on neighborhood graph tends to encounter a lot of disk page accesses. We present a technique called OSH (Orientation Sensitive Hashing) to prevent from wasting unnecessary disk page accesses. Inspired by LSH, we find that during the expansion, neighbors that are closer to the query are more like to provide valid converge path. However, most methods do not pay attention on filter out further neighbors which we consider as false positives (or unnecessary points) during the expansion.

OSH can register the orientation of all neighbors of a node. During the ANN search, the orientation of the query w.r.t the current node can also be figured out quickly. We then sort all the orientation bit string and firstly choose the one with the most same orientation to expand.

1 Motivation

NNG has now become a powerful tools to address ANN search, especially in high dimensions.

However, there is a common drawback in most search algorithms on NNG, they encounter large amount of random I/Os and consume large amount of original distance evaluations (which is very expensive in high dimensions). There are mainly three reasons.

- Due to the large volume of the dataset, it is reasonable to store the topology of the NNG apart from the original dataset. The topology of the NNG is a kind of inverted file, where each node maintains all the IDs of its neighbors. When converging on the NNG, the original data of the neighbors are loaded following the needs. This is a space-efficient storage strategy. Instead, one has to store multiple copies of the datasets. However, the disadvantage is obvious, the neighbors can not be stored adjacently on the disk. Therefore, all the load of the neighbor are of random I/O consumptions.
- Existing NNG search algorithms do not provide further to save random I/Os.

The reason is that there is no reference that can be used to fastly discriminate the quality of the neighbors of the current node. A basic operation of NNG convergence is that when locating at a prior node, one needs to load and check the neighbors of the prior node to provide further converge route to the true NNs.

Existing algorithms can be divided into two categories, the greedy ones and the backtrack ones.

to construct further route to converge or

We see how many places we can apply our OSH techniques.

2 Introduction

3 Background

3.1 Search algorithms on NNG

There has been a development over the search algorithms on NNG (or k-NNG).

3.1.1 Naive algorithm

A naive algorithm is the downhill search algorithm, as shown in Fig. 1.

Algorithm 1: Naive downhill search

Input: graph vertices P, directed graph edges E,
query point Q, search start index v

Output: nearest neighbour index v

for *each edge* E_i *with start vertex* P_v **do**
  $u \leftarrow$ index of end vertex of E_i
  if *distance*$(Q, P_u) <$ *distance*(Q, P_v) **then**
    $v \leftarrow u$

return v

Figure 1: Downhill search algorithm

In [1], the authors uses a "best-first" strategy to search NNs: start from a random point Y_0, expand to the best neighbor among its E neighbors, until it reaches a point who is better than any of its neighbor. Collect all encountered E neighbors, find best K ones as the K-NN search result. **The efficiency is very plain, the speedup around 3-16 on a 17k SIFT datasets with** $K = 30$.

FANNG [2] proposes a backtrack search algorithm, which is some kind of confused, I cannot understand yet. It seems like a greedy strategy, because it will move to a nearer node once found a nearer one. What confuses me is the priority query stores edges instead of nodes. Now I understand, all the edges are sorted according to the distance between the start point and the query.

$>$ Our technique can help to provide a sorted edge list in the ascending order of the distance to the query, in order to speedup the convergence, as shown in Fig. 3.

NNGPQ [**?**] uses a combination of kNNG and a bridge NNG. During the search, it uses a priority queue Q of data points. Each time pop out a best one, expand all its neighbors, push them into the Q and also update the ANN result.

$>$ It is concerned that there is a case which we do not take into account. If we only expand neighbors that are closer than the current point O, when O is quite near to the query, a neighbor of O which is even further than O can be a true NN. For this case, we say that for all neighbors that are closer to O we consider,

4 Our methods

5 Sketch of orientation sensitive hashing

Orientation sensitive, as the term suggests, means that we want to seize the orientation similarity between data points.

5.1 Definition of orientation

Suppose there is a reference point $o \in R^d$, given two data points p and q we define the orientation of p w.r.t q as whether p and q located at the same side of the hyperplane $H_{qo}^{\perp}$, i.e.,

$$orient_q(p) = sign(op \cdot oq) \tag{1}$$

Input: a k-NN graph $\mathcal{G} = (\mathcal{D}, \mathcal{E})$, a query point Q, the number of required nearest neighbors K, the number of random restarts R, the number of greedy steps T, and the number of expansions E.
ρ is a distance function. $N(Y, E, \mathcal{G})$ returns the first E neighbors of node Y in $\mathcal{G}$.
$\mathcal{S} = \{\}$.
$\mathcal{U} = \{\}$.
$Z = X_1$.
for $r = 1, \ldots, R$ **do**
  Y_0: a point drawn randomly from a uniform distribution over $\mathcal{D}$.
  for $t = 1, \ldots, T$ **do**
    $Y_t = \mathrm{argmin}_{Y \in N(Y_{t-1}, E, \mathcal{G})}\, \rho(Y, Q)$.
    $\mathcal{S} = \mathcal{S} \bigcup N(Y_{t-1}, E, \mathcal{G})$.
    $\mathcal{U} = \mathcal{U} \bigcup \{\rho(Y, Q) : Y \in N(Y_{t-1}, E, \mathcal{G})\}$.
  end for
end for
Sort $\mathcal{U}$, pick the first K elements, and return the corresponding elements in $\mathcal{S}$.

Figure 2: NN Search in FkNNG

Algorithm 3: Backtrack search

Input: graph vertices P, directed graph edges E, query point Q, search start index v, maximum distance calculations M
Output: nearest neighbour index n

```
X ← empty priority queue  // closest to Q first
add edge e_0 with start vertex P_v to X
m ← 1                     // count distance computed to Q
n ← v
while m < M do
    e_i ← remove top of X
    u ← = index of end vertex of e_i
    if P_u has not been visited yet then
        add edge e_0 with start vertex P_u to X
        m ← m + 1         // add 1 to compute count
        if distance(Q, P_u) < distance(Q, P_n) then
            n ← u
    v ← = index of start vertex of e_i
    if i < number of edges with start vertex P_v then
        add edge e_{i+1} with start vertex P_v to X
return n
```

根据start vertex到query的距离

每个has not been visited 的节点在初次处理时，先根据OSH进行排序，此后按照OSH的顺序加入

Figure 3: NN Search in FkNNG

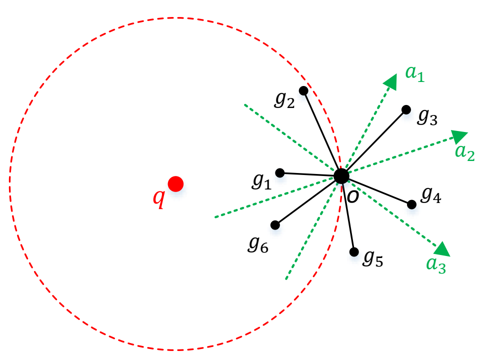

Figure 4: Sample of orientation hashing

We first define a function *OSH_order* that calculates OSH keys for all edges of a node

6 Related Works

7 Evaluation

8 Conclusion

Acknowledgments

References

[1] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2011, pp. 1312–1317. [Online]. Available: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-222

[2] B. Harwood and T. Drummond, "FANNG: fast approximate nearest neighbour graphs," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 5713–5722.