A Survey of LSH for Similarity Search

Zeyan Li 2018310816 **Chencheng Xu 2018310851**

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Introduction

In past decades, locality sensitive hashing (LSH) has been widely proposed for similarity search in high-dimensional spaces, especially for content-based search of feature-rich data such as audio recordings, digital photos, digital videos, and other sensor data (2). Generally speaking, there are two ways to apply hashing to similarity search (**?**). The first one is indexing data terms using hash tables, which is formed by storing the terms with the same code in hash buckets. In detail, algorithms in this way regard the items lying the buckets corresponding to the codes of the query as the similarity candidates and then determine nearest neighbors by checking the distance between the query and candidates. Most researches in this direction aim to design hash functions satisfying the locality sensitive property and to design efficient search schemes, such as Multi-probe LSH (2), Entropy-based LSH (**?**) and query adaptative LSH (**?**). The other way is to approximate the distance between instances by the one computed with hash code, which ranks the instances according to the distances computed using the short codes. Such algorithms include LSH forest (**?**) and dynamic collision counting based LSH (1).

An ideal LSH scheme for similarity search should satisfy the following properties (**?** 2):

- Accurate: $\frac{\#\ of\ True\ Near\ Neightbors}{\#\ of\ Retrieved\ Candidates}$ should be as large as possible.
- Efficient Queries: The number of retrieved candidates should be as small as possible to reduce I/O and computation costs.
- Efficient Maintenance: The index should be built in a single scan of the dataset, and subsequent inserts and deletes of objects should be efficient.
- Domain Independence: The indexing scheme should adapt to any data domain and require no special tuning of parameters for each specific dataset.
- Minimum Storage: Storage consumption should be as little as possible, ideally linear in the data size.

2 Methods of LSH

2.1 Entropy-based LSH

(**?**) proposed a method to hash several randomly chosen points in the neighborhood of the query point and showed that at least one of them will hash to the bucket containing its nearest neighbor. The number of randomly chosen points in the neighborhood of the query point q required depends on the entropy of the hash value $h(p)$ of a random point p at distance r from q, given q and the locality preserving hash function h chosen randomly from the hash family.

Preprint. Work in progress.

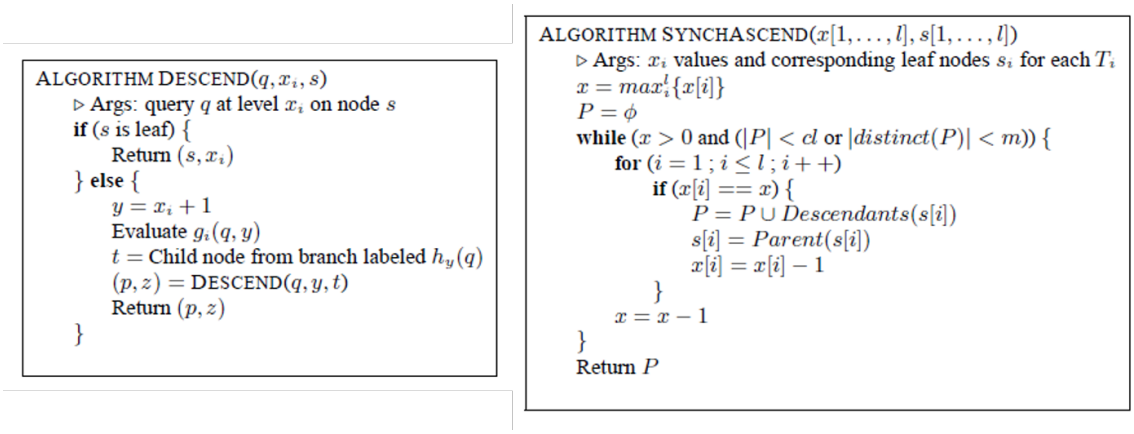

Figure 1: Top-down and bottom-up phase of LSH Forest query.

More detailly, let $I(X)$ donate the information-entropy of a discrete random variable X. For example, if X takes N possible values with probabilities $w_1, w_2, ..., w_N$, then $I(X) = I(w_1, w_2, ..., w_N) = \sum -w_i \log w_i$. Define $B(p, r)$ as the sphere of radius r centered at p. The proposed method tries to retrieve c-approximate nerest neighbors of a query point q when giving a distance r. Donate $M = I(h(p)|q, h)$, where p is a random point in $B(q, r)$, and g as an upper bound on the probability that two points that are at least distance cr apart will hash to the same bucket, then the algorithm is:

- **Constraction of hash table**: pick $k = \frac{\log n}{\log(1/g)}$ random hash functions $h_1, h_2, ..., h_k$. For each point p in the database compute $H(p) = (h_1(p), h_2(p), ..., h_k(p))$ and store p in a table at location $H(p)$. *polylogn* is used to construct hash tables.
- **Search**: Given q and r, pick $O(n^\rho)$ random points v from $B(q, r)$, where $\rho = \frac{M}{\log(1/g)}$, and search in the buckets $H(v)$.

The author illustrated that with such an algorithm, the time complexity to find the approximate nearest neighbor is $O(d + n^\rho)$ and the space complexity is near linear.

2.2 LSH forest

Since the B+ tree is always accurate, returning exactly the query results, the author tried to combine LSH and B+ tree together to improve accuracy and efficiency on a dynamic set of objects (**?**). The paper aims to design an index structure that enables efficient ϵ-approximate nearest-neighbor queries, the efficient building of the index, efficient insertion and deletion of points, and complete domain independence, all while ensuring minimal use of storage. The proposed methods work for any choice of distance function D for which there is a corresponding LSH family.

The main idea is to encode each data point with variable length hash label $g(p) = (h_1(p), ..., h_k(p))$, where k is different for different p and $k \leq k_m$. Then l prefix trees are constructed on the set of all labels, with each leaf corresponding to a point, and different trees with different hash function sets. In the query procedure, a query for the m nearest neighbors of a point q is answered by traversing the LSH Trees in two phases. In the top-down phase, the leaf x having the largest prefix match with q's label is found by descending each LSH Tree; in the second bottom-up phase, M points from the LSH Forest are collect by moving up from level x towards the root synchronously across all LSH Trees. These two phases are shown in Figure 1. The insertion can be done by simply top-down searching and the deletion top-down and removing the appropriate leaf node.

The theoretical analysis reveals that the LSH Forest is able to return ϵ-approximate neighbors of a query q with a non-zero probability greater than a constant C, as long as the distance from q to its neighbor is in the range (a, b), where the range (a, b) is the definition domain of hash family H.

The author also introduced ways to implement this algorithm in main-memory and disk. In the main-memory implementation, a long chain of logical internal nodes is compressed into just one

node that stores the path to it from the previous node to improve storage space. In Disk-based implementation, in order to minimize the number of disk accesses required to navigate down the tree for a query, the Prefix B-Trees is considered as the data structure to construct the LSH forest. The results on peer-to-peer task showing that LSH Forest exhibits higher efficiency than other methods, with higher average similarity and lower mean relative error in a dynamic setting.

2.3 Adaptative LSH

This work (**?**) using E_8 lattices as hash functions instead of random projections to balance the trade-off between retrieval accuracy and complexity.

A lattice is a discrete subset of R^d defined by a set of vectors of the form $\{x = u_1a_1 + ... + u_da_d | u_1, ..., u_d \in Z\}$. The E_8 lattice can be defined based on another 8-dimensional lattice, the D_8 lattice, which is the set of point of Z^8 whose sum is even, e.g. $(1, 1, 1, 1, 1, 1, 1, 1) \in D_8$. Then E_8 lattice is defined as:

$$E_8 = D_8 \cup (D_8 + \frac{1}{2})$$

The author argues that the relevance of the hash function used in LSH is the lower the better. Based on the properties of E_8 lattice, the hash functions are defined as

$$h_i(x) = E_8(\frac{x_{i,8} - b_i}{w})$$

The detailed algorithm of QA-LSH is exactly the same as the original LSH algorithm, except the hash function family is defined based on E_8 lattice, instead of randomly projection. The results indicate that the QA-LSH achieve a higher proportion of nearest neighbors correctly found compared with traditional LSH methods.

3 Evaluation

3.1 Datasets

3.2 Evaluation Metrics

3.3 Effectiveness

3.4 Time Efficiency

3.5 Space Efficiency

4 Conclusion

References

[1] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 541–552. ACM, 2012.

[2] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment, 2007.