Scalable K-Nearest Neighbor Graph Construction Based on Greedy Filtering

Youngki Park, Sungchan Park, Sang-goo Lee
School of Computer Science and Engineering
Seoul National University
{ypark, baksalchan, sglee}@europa.snu.ac.kr

Woosung Jung
Department of Computer Engineering
Chungbuk National University
wsjung@cbnu.ac.kr

ABSTRACT

K-Nearest Neighbor Graph (K-NNG) construction is a primitive operation in the field of Information Retrieval and Recommender Systems. However, existing approaches to K-NNG construction do not perform well as the number of nodes or dimensions scales up. In this paper, we present *greedy filtering*, an efficient and scalable algorithm for selecting the candidates for nearest neighbors by matching only the dimensions of large values. The experimental results show that our K-NNG construction scheme, based on greedy filtering, guarantees a high recall while also being 5 to 6 times faster than state-of-the-art algorithms for large, high-dimensional data.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

Keywords

K-nearest neighbor graphs; greedy filtering; similarity join

1. INTRODUCTION

K-NNG construction is a primitive operation in the field of Information Retrieval and Recommender Systems. For example, assuming that we constructed a K-NNG graph whose nodes represent users, we can quickly recommend items to user u by looking at the purchase lists of u's nearest neighbors. As another example, if we implement an enterprise search system, we can easily provide an additional feature that finds K documents most similar to recently viewed documents.

Let us define the K-NNG construction formally. We assume that the nodes of the K-NNG are represented by normalized vectors, as shown in Table 1. Let the normalized vectors be V and the dimensions in V be D. Each vector $v \in V$ consists of $\langle d_i, r_j \rangle$ pairs where $d_i \in D$ and $0 \leq r_j \in \mathbb{R} \leq 1$. Given cosine similarity $sim(v_i \in V, v_j \in V) = (v_i \cdot v_j)/(\|v_i\| \, \|v_j\|)$ as a similarity measure, the K-NNG construction returns the top-k similar vectors for each vector.

As far as we know, NN-Descent [1] is the most efficient approach for constructing K-NN graphs. While brute-force algorithms require too many similarity calculations that cost $O(|V|^2|D|)$, NN-Descent dramatically reduces the number of comparisons. However, it does not perform well as the number of nodes or dimensions scales up. In this paper, we present a novel, scalable algorithm and compare its performance with those of existing approaches.

Table 1: An example of greedy filtering

v_1	d_1 0.5	d_3 0.37	d_8 0.33	d_4 0.31	d_9 0.23	...
v_2	d_1 0.73	d_2 0.55	d_5 0.37	d_3 0.1	d_8 0.05	...
v_3	d_2 0.4	d_7 0.29	d_6 0.27	d_1 0.25	d_{10} 0.1	...
v_4	d_5 0.8	d_4 0.35	d_3 0.3	d_2 0.27	d_1 0.25	...
v_5	d_3 0.48	d_5 0.37	d_7 0.34	d_4 0.32	d_{10} 0.2	...

2. CONSTRUCTING K-NN GRAPHS

Assuming that we sorted the elements of each vector into a descending order according to their values and that the prefix of each vector is determined beforehand (See Table 1, for example, where the prefixes are colored), according to length filtering [2], we can guarantee that $sim(\text{suffix}(v_i), \text{suffix}(v_j)) \leq \tau_{v_i}, \forall v_j$ based only on the values of prefix(v_i). This raises the question of whether we can guarantee that $sim(v_i, v_j) \leq \tau_{v_i}$ if the dimensions of prefix(v_i) and prefix(v_j) do not match at all. If $sim(v_i, v_j) \leq \tau_{v_i}$, then there's a high probability that the vectors $\langle v_i, v_j \rangle$ are not similar, because we sorted the elements so that τ_{v_i} is a small value. In Table 1, there is a counterexample in which $sim(v_2, v_4) > \tau_{v_2} = 0.58$ despite the fact that the dimensions of the prefixes do not match. However, we observe that the counterexamples are few and far between.

If we generalize this observation, we can assert that in most cases, two vectors are not similar if their prefixes do not overlap at all. If two vectors are not similar, there would be no edge in K-NNG between them. That is to say, we would obtain an approximate K-NNG by calculating similarities between sorted vectors that have at least one common dimension in their prefixes. In Table 1 as an example, we calculate the similarities of $\langle v_1, v_3 \rangle$, $\langle v_1, v_4 \rangle$ and we filter out $\langle v_3, v_5 \rangle$. Because this approach initially checks whether the dimensions of large values match, we will call it *greedy filtering.*

Algorithm 1 K-NNG Construction Based on GF

```
Data: sorted vectors $V$, parameters $\rho$, $K$
Result: K-NN queues $Q$
begin
    $L[d_i] \longleftarrow \phi, \forall d_i \in D$
    for $v_i \in V$ do
        $L[dim(e^1_{v_i})] = L[dim(e^1_{v_i})] \cup v_i$
        $P[v_i] \longleftarrow 2, \forall v_i \in D$
    end
    repeat
        for $d_i \in D$ do
            for $v_j \in V$ do
                $v_j \longleftarrow$ the $j^{th}$ vector in $L[d_i]$
                $SC[v_j] \longleftarrow SC[v_j] + |L[d_i]|$
            end
        end
        for $v_i \in V$ do
            if $SC[v_i] < \rho K$ and $|v_i| \geq P[v_i]$ then
                $L[dim(e^{P[v_i]}_{v_i})] \longleftarrow L[dim(e^{P[v_i]}_{v_i})] \cup v_i$
                $P[v_i] \longleftarrow P[v_i] + 1$
            end
        end
    until $L$ is not changed in this iteration
    $Q[v_i] \longleftarrow \phi, \forall v_i \in V$
    for $d_i \in D$ do
        compare all pairs $\langle v_x, v_y \rangle$ in $L[d_i]$
        update the priority queues, $Q[v_x]$ and $Q[v_y]$
    end
    return $Q$
end
```

Algorithm 1 describes how a K-NNG is constructed based on greedy filtering. $e^j_{v_i}$ denotes the j^{th} element of the vector v_i, $dim(e)$ denotes the dimension number of the element e, and $SC[v_i]$ denotes the number of similarity calculations for the vector v_i. For vectors whose $SC < \rho K$, we incrementally increase their prefix sizes in a round-robin fashion. Table 1 shows the prefixes determined by $\rho = 2$. By default, we use $\rho = 2$, which means that we expect there are K-nearest neighbors for each vector among approximate ρK-nearest neighbors.

3. EXPERIMENTS

We considered four types of algorithms for comparison: Inverted Index Join as a baseline algorithm, which calculates all similarities with inverted indices [2]; NN-Descent [1], which was developed for the purpose of constructing K-NN graphs; similarity join algorithms [2][3], which return every pair $\langle v_i, v_j \rangle$ whose similarity is above ϵ; and the top-k similarity join algorithm [4], which returns k pairs with the highest similarity values. The third and fourth types of algorithms should be repetitively executed while varying ϵ or k until K-NN graphs are found. Unfortunately, because they did not perform better than Inverted Index Join, we only report the results of Inverted Index and NN-Descent for comparison in this paper. We implemented all algorithms in Java and executed them on a single workstation with 14GB RAM and a 3GHz CPU.

Figure 1 shows the execution time for the DBLP dataset as the data scales ($\rho = 2$ and $K = 10$). Not shown is the data preprocessing time, which takes only a minor portion.

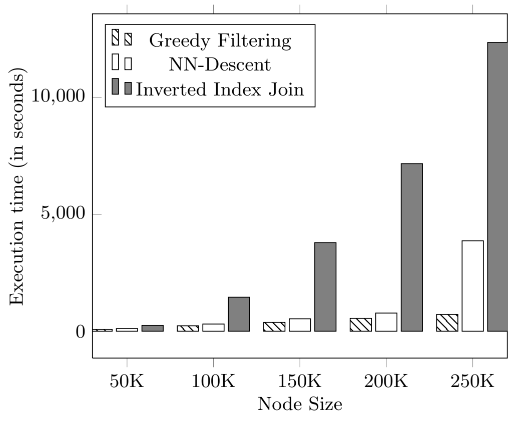

Figure 1: Execution time

Table 2: Recall and scan rate

Nodes	Dim.	NN-Descent		Ours	
		recall	scan.	recall	scan.
50,000	55,010	0.469	0.016	0.898	0.061
100,000	87,759	0.339	0.008	0.916	0.045
150,000	115,703	0.282	0.006	0.914	0.032
200,000	140,913	0.247	0.004	0.901	0.024
250,000	163,841	0.211	0.004	0.906	0.019

Our approach performs best among all algorithms; comparatively much better when the data scales. Table 2 shows the recall and scan rate with the same parameters. Recall is the number of correct k-nearest neighbors in K-NNG divided by the number of edges in K-NNG; the scan rate is the total number of similarity calculations of the algorithm divided by the total number of similarity calculations of the brute-force search. The results show that our approach maintains a stable recall value above 0.9 as the data scales, whereas the recall of NN-Descent decreases gradually. An interesting finding is that the scan rate of NN-Descent is lower than that of our approach. However, because our approach does not require significant overhead, such as overhead for maintaining hash tables, it is faster than NN-Descent.

4. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20120005695). This work was supported by Seoul R&BD Program(WR080951C0209725).

5. REFERENCES

[1] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *WWW '11*, pp. 577–586, 2011.

[2] D. Lee, J. Park, J. Shim, and S.-g. Lee, "An efficient similarity join algorithm with cosine similarity predicate," in *DEXA '10*, pp. 422–436, 2010.

[3] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *WWW '07*, pp. 131–140, 2007.

[4] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k set similarity joins," in *ICDE '09*, pp. 916–927, 2009.