

Efficient Graph Building from Text Data

Thibault Debatty

Royal Military Academy, Brussels, Belgium

THIBAUT.DEBATTY@RMA.AC.BE

Pietro Michiardi

EURECOM, Campus SophiaTech, France

PIETRO.MICHIARDI@EURECOM.FR

Olivier Thonnard

Symantec Research Labs, Sophia Antipolis, France

OLIVIER_THONNARD@SYMANTEC.COM

Wim Mees

Royal Military Academy, Brussels, Belgium

WIM.MEES@RMA.AC.BE

Abstract

This is the abstract for this article.

Keywords: List of keywords

1. Introduction

A graph is a mathematical structure used to represent relations between objects. A graph is made up of nodes (or vertices) connected with edges. In some cases, the edges have a weight, resulting in a weighted graph. Graph theory is a very ancient topic, dating back from 1736. It has received a strong highlight these last years with the explosion of social networks like Facebook and Twitter. Indeed, the data generated by these networks can easily be formatted as a graph, making graph algorithms a premium choice for processing the data.

As a consequence, a lot of algorithms have been developed, and implemented, to efficiently analyze this kind of data, either sequentially or in parallel. XXXXXXXX add examples of graph algorithms connected components, clustering, ...

In a lot of cases, the data to analyze is not formatted as a graph. Nevertheless, it can be interesting to convert the dataset into a graph, to be able to use these algorithms. In most cases (XXXXXXXXXX add reference), the trick used is to build a k-nearest neighbors graph (k-NN graph), a graph where each node is connected to (has an edge to) its k nearest neighbors, according to a given similarity metric.

2. Related work

2.1. Nearest neighbor search

k-NN graph building algorithms are of course closely related to nearest neighbor search algorithms, and also to the k-nearest neighbors method.

The nearest-neighbor search problem is formally defined as follows: given a set S of points in a space M and a so-called query point $q \in M$, find the closest point in S to q . The

k-NN search is a direct generalization of this problem, where we need to find the k closest points.

But k-NN search is also a method used for classification and regression. In k-NN classification, the algorithm assigns to an object the class which is most common amongst its k nearest neighbors in the training set. In the case of regression, the value computed by the algorithm is the average value of the k nearest neighbors of the object in the training set.

Different methods exist to find the nearest neighbor, or the k -nearest neighbors, of a point. The naive method, also called linear search, consists in computing the distance between the query point and every other point in the set, keeping track of the "best so far" (or k "best so far").

Some techniques rely on the branch and bound algorithm, with some kind of index to partition the space M . For example, a k -d tree, that recursively partition the space into equally sized sub-spaces can be used to speedup search.

XXXXXXX ref: Andrew Moore. "An introductory tutorial on KD trees" and Lee, D. T.; Wong, C. K. (1977). "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees". *Acta Informatica* 9 (1): 2329. doi:10.1007/BF00263763.

R-trees can also be used for euclidean spaces. In the case of metric spaces, vantage-point trees and BK-trees can be used.

XXXXXXX add explanation and references!

Finally, some techniques use locality-sensitive hashing (LSH). XXXXXX ref: A. Rajaraman and J. Ullman (2010). "Mining of Massive Datasets, Ch. 3.". LSH is originally a method used to perform probabilistic dimension reduction of high-dimensional data. The basic idea is to hash the input items so that similar items are mapped to the same buckets with high probability. At the opposite of conventional hash functions, such as those used in cryptography, in the case of LSH the goal is to maximize the probability of "collision" between similar items.

2.2. k-NN graph building algorithms

The most naive way to build a k-NN graph is of course using brute force to compute all pairwise similarities. This method has a computational cost of $O(n^2)$.

A more subtle way consists in iteratively using a nearest neighbor search algorithm, like the ones presented above, to find the neighbors of all nodes in the dataset.

Lately, different algorithms have been proposed specifically to efficiently build an approximate k-NN graph from a dataset. Most of them will naturally share similarities with nearest neighbor search algorithms.

XXXX add references to algorithms that uses indexes.

A versatile algorithm to efficiently compute an approximate k-NN graph is described in [Dong et al. \(2011\)](#). The algorithm, called NN-Descent, starts by creating edges between random nodes. Then, for each node, it computes the similarity between all neighbors of the current neighbors, to find better edges. The algorithm iterates until it cannot find better edges anymore. The main advantage of this algorithm is that it works with any similarity measure. Dong et al. experimentally found the computational cost is around $O(n^{1.14})$.

They also propose a MapReduce version of the algorithm. Internally, instead of working with edges, each node has a neighbors list, a list of candidate neighbors. The algorithm first creates a random neighbors list for each node. Then each iteration is realized with two MapReduce jobs. First, the mapper emits the initial neighbors list, and reverses the neighbors list to produce new candidate neighbors. The reducer merges all neighbors for each node, to produce a new extended neighbors list. In the second job, the mappers compute and emit the pairwise similarity between all elements of each neighbors list. Finally, the reducer merges the neighbors of each key node, keeping only the k neighbors with the highest similarity.

Various authors propose algorithms relying on locality-sensitive hashing.

In [Hsieh and Wu \(2012\)](#), the authors propose a MapReduce algorithm that first bins the scale-invariant feature transform (SIFT) description of images into overlapping pools. The algorithm then computes the pairwise similarity between images in the same bucket to build the k-nn graph of images. For binning, the algorithm uses MinHash, a variant of LSH that uses Jaccard coefficient as similarity measure.

Similarly, in [Zhang et al. \(2013\)](#), the authors use LSH to divide the dataset into small groups. Then, inside these small groups, they use the algorithm proposed by Dong et al. to build the k-nn graph. As groups are not overlapping, the constructed graph is a union of multiple isolated small graphs. To bind the final graph, and improve the approximation quality, the division is repeated several times to generate multiple approximate graphs, which are finally combined to produce the final graph.

Furthermore, the authors propose a method to produce equally sized groups, thus alleviating the computational cost of skewed data. They first project the item’s hash code on a random direction. Then they sort items by their projection values. Finally, they divide this sequence of items into equally sized buckets. By doing so, the items with same hash code will fall in the same bucket with a high probability.

Finally, Zhang et al. show experimentally that their algorithm is much faster than existing algorithms, for similar quality of the built graph. But the main drawback of the algorithm is that LSH functions are defined only for some similarity measures: l_p , Mahalanobis distance, kernel similarity, and χ^2 distance.

3. NNCTPH

3.1. Context Triggered Piecewise Hashing

4. Experimental evaluation

4.1. Performance measure

Like in [Dong et al. \(2011\)](#), we use recall to measure the accuracy of algorithms. The ground truth is the true k-NN graph obtained using the naive, brute-force, algorithm. The recall of a single node is the number of its correct edges divided by k . The recall of an approximate k-NN graph is the average recall of all nodes.

To compare computational cost, as the number of similarities to compute depends on the size of the dataset, we use the scan rate, defined as follows:

$$\text{scan rate} = \frac{\# \text{ similarity evaluations}}{n(n-1)/2}$$

where $n(n-1)/2$ is the number of similarities computed by the naive algorithm.

4.2. Results

Table 1: Running time and scan rate of NNCTPH algorithm

	spam200k	spam400k	spam600k	spam800k
# spams	200.000	400.000	600.000	800.000
Running time (sec)	97	316	675	1093
# similarities	88.262.345	359.577.959	773.200.303	1.346.569.275
Scan rate	0.44%	0.45%	0.43%	0.42%

5. Conclusions

Acknowledgments

This work has been partially supported by the EU project BigFoot (FP7-ICT-317858).

References

- Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th international conference on World wide web - WWW '11*, page 577, 2011. doi: 10.1145/1963405.1963487. URL <http://portal.acm.org/citation.cfm?doid=1963405.1963487>.
- LC Hsieh and GL Wu. Two-stage sparse graph construction using MinHash on MapReduce. In *ICASSP*, pages 1013–1016, 2012. ISBN 9781467300469. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288057.
- Yan-ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-lin Liu. Fast k NN Graph Construction with Locality Sensitive Hashing. In *Machine Learning and Knowledge Discovery in Databases*, pages 660–674. 2013. ISBN 978-3-642-40990-5.