Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Vlad Sebastian Velici
December 7, 2014

# Similar nodes in large graphs

Project Supervisor: Dr. Adam Prügel-Bennett
Second Examiner: Dr. Sasan Mahmoodi

A progress report submitted for the award of
MEng Computer Science with Artificial Intelligence

# Contents

# INTRODUCTION

Plenty of datasets are or can be represented as graphs where vertices represent entities and edges represent relationships between entities. A problem of interest is to find entities that are similarly connected. Example instances of this problem are finding *people you may know* in a social network, people with common interests from research publications repositories or identifying possible duplicates in a dataset.

It is easy to find exact similarities between vertices in small graphs by performing pairwise comparisons. Such an algorithm is too slow for large datasets of millions of vertices. This project investigates a method to compute an approximation of similarities between nodes and attempt to evaluate its performance on different datasets.

# BRUTE-FORCE APPROACH

The similarity algorithm is described in regards to the undirected graph $G(v, \epsilon)$ with adjacency matrix $M$. The number of nodes is $N = |v|$ and the number of edges is $|\epsilon|$. In order to derive an algorithm to find similarities between vertices in a graph, a measure of similarity between two vertices must be defined. $M$ is the adjacency matrix of the graph and $M_{ij} = [(i, j) \in \epsilon]$, that is $M_{ij} = 1$ if there is an edge between vertices $i$ and $j$, otherwise $M_{ij} = 0$. Let $D$ be a normalised adjacency matrix such that each column sums up to 1:

$$D_{ij} = \frac{1}{\sum_{j=1}^{N} M_{ij}} \tag{1}$$

Let $\delta_\mathbf{i}$ be the $i^{th}$ column of the $N \times N$ identity matrix. Let $\mathbf{C_i}$ be a vector that represents how vertex $i$ is connected to the graph. In a brute-force approach, it can be computed as:

$$\mathbf{C_i} = \sum_{t=0}^{\infty} \mu^t D^t \delta_\mathbf{i} \tag{2}$$

Where $\mu \in (0, 1)$ is a penalising factor. $\mu$ controls how much the far connections are considered into the similarities. Therefore a large $\mu$ enforces consideration of very far connections whereas a small $\mu$ considers far connections less. The similarity between two vertices $i$ and $j$

is the norm of the difference between $\mathbf{C_i}$ and $\mathbf{C_j}$:

$$s(i, j) = \|\mathbf{C_i} - \mathbf{C_j}\| \tag{3}$$

The brute-force approach was run on small randomly generated connected graphs (200 vertices) with 50 iterations and it gave meaningful results but it does not scale for large datasets. In the next section an algorithm that scales to large datasets is described.

## APPROXIMATIVE METHOD

Geometrically, if vectors $\mathbf{C_i}$ and $\mathbf{C_j}$ start from the origin, then $\mathbf{C_i}$, $\mathbf{C_j}$ and $\mathbf{a} = \mathbf{C_i} - \mathbf{C_j}$ describe a triangle. Let $\theta$ be the angle between $\mathbf{C_i}$ and $\mathbf{C_j}$. From the Law of Cosines:

$$\|\mathbf{a}\|^2 = \|\mathbf{C_i}\|^2 + \|\mathbf{C_j}\|^2 - 2\|\mathbf{C_i}\|\|\mathbf{C_j}\|\cos\theta \tag{4}$$

Take the dot product $\mathbf{C_i^T C_j} = \|\mathbf{C_i}\|\|\mathbf{C_j}\|\cos\theta$. Observe that the angle between $\mathbf{C_i}$ and itself is 0 and $\cos 0 = 1$, and obtain $\|\mathbf{C_i}\|^2 = \mathbf{C_i^T C_i}$. Now *Equation 4* becomes:

$$\|\mathbf{a}\|^2 = \mathbf{C_i^T C_i} + \mathbf{C_j^T C_j} - 2\mathbf{C_i^T C_j} \tag{5}$$

Therefore it is sufficient to be able to compute all dot products $\mathbf{C_i^T C_j}$ (for all $i, j \in v$) to compute the similarities between all vertices of the graph. Computing and storing all $\mathbf{C_i^T C_j}; i, j \in v$ takes too much time and memory for a very large dataset if we use the brute-force approach.

An approximative algorithm to efficiently compute the dot products $\mathbf{C_i^T C_j}$ will be described. Let $W$ be a diagonal matrix:

$$W_{ij} = \frac{[i = j]}{\sum_{j=1}^{N} M_{ij}} \tag{6}$$

Now *Equation 1* can be written as $D = WM$, thus

$$D^t = (WM)^t \tag{7}$$

Define a new matrix $A$:

$$A = W^{1/2} M W^{1/2} \tag{8}$$

From *Equation 7* and *Equation 8*:

$$D^t = W^{1/2} A^t W^{-1/2} \tag{9}$$

The matrices $W^{1/2}$ and $W^{-1/2}$ are easy to compute because $W$ is a diagonal matrix. Apply the matrix operation only on the diagonal elements and obtain $(W^{1/2})_{ii} = (W_{ii})^{1/2}$ and $(W^{-1/2})_{ii} = (W_{ii})^{-1/2}$.

Let $\lambda_i$ be the $i^{th}$ eigenvalue and $\bigvee^{(i)}$ be the $i^{th}$ eigenvector ($\forall i \in \{1, 2, ..., N\}$). $\bigvee$ is a matrix of all computed eigenvectors (such that $\bigvee^{(i)}$ is the $i^{th}$ column of $\bigvee$), and $\bigwedge$ is a diagonal matrix of all computed eigenvalues:

$$\bigvee = \begin{bmatrix} | & | & & | \\ \bigvee^{(1)} & \bigvee^{(2)} & \cdots & \bigvee^{(N)} \\ | & | & & | \end{bmatrix} \tag{10}$$

$$\bigwedge = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \lambda_N \end{bmatrix} \tag{11}$$

The eigendecomposition of the matrix A is $A = \bigvee \bigwedge \bigvee^{-1}$. The graph $G$ is an undirected graph which makes $A$ a real symmetric matrix, thus $\bigvee^{-1} = \bigvee^T$. The eigendecomposition of the matrix $A$ is therefore:

$$A = \bigvee \bigwedge \bigvee^T \tag{12}$$

$$\Rightarrow A^t = (\bigvee \bigwedge \bigvee^T)^t = \bigvee \bigwedge^t \bigvee^T \tag{13}$$

$$\Leftrightarrow A^t = \sum_{a=1}^{N} \lambda_a^t \bigvee^{(a)} \bigvee^{(a)^T} \tag{14}$$

From *Equation 9* and *Equation 14*:

$$D^t = W^{1/2} \sum_{a=1}^{N} \lambda_a^t \bigvee^{(a)} \bigvee^{(a)^T} W^{-1/2} \tag{15}$$

Plug $D^t$ from *Equation 15* into *Equation 2*:

$$\mathbf{C_i} = \sum_{t=0}^{\infty} \mu^t W^{1/2} \sum_{a=1}^{N} \lambda_a^t \bigvee^{(a)} \bigvee^{(a)^T} W^{-1/2} \delta_{\mathbf{i}} \tag{16}$$

Rearrange the equation and obtain:

$$\mathbf{C_i} = W^{1/2} \sum_{a=1}^{N} \sum_{t=0}^{\infty} \mu^t \lambda_a^t \bigvee^{(a)} \bigvee^{(a)^T} W^{-1/2} \delta_{\mathbf{i}} \tag{17}$$

Note that $\sum_{t=0}^{\infty} \mu^t \lambda_a^t$ is a sum of a geometric series with the first term 1 and ratio $\mu\lambda_a$, therefore:

$$\sum_{t=0}^{\infty} \mu^t \lambda_a^t = \frac{1}{1 - \mu\lambda_a} \tag{18}$$

Observe $\bigvee^{(a)^T} W^{-1/2} \delta_{\mathbf{i}}$ is a scalar:

$$\bigvee^{(a)^T} W^{-1/2} \delta_{\mathbf{i}} = \bigvee_i^{(a)} W_{ii}^{-1/2} \tag{19}$$

Substitute *Equation 19* and *Equation 18* back into *Equation 17* and obtain:

$$\mathbf{C_i} = W^{1/2} \sum_{a=1}^{N} \frac{1}{1 - \mu\lambda_a} \bigvee^{(a)} \bigvee_i^{(a)} W_{ii}^{-1/2} \tag{20}$$

An approximation of the vectors $\mathbf{C_i}(\forall i \in \nu)$ can be obtained by only using the $m$ leading eigenvectors and eigenvalues of $A$ instead of all of them. Compute the largest $m$ eigenvalues and eigenvectors of the matrix $A$ using the Lanczos Method for real symmetric matrices[1]. For directed graphs the Arnoldi Iteration[2] will be used instead. These algorithms perform well on large sparse matrices. In real datasets it is very often the case that the number of edges is orders of magnitude smaller than the number of vertices squared, resulting in sparse adjacency matrices and a sparse $A$. Define $\hat{\mathbf{C}}_{\mathbf{i}}$ to be an approximation of $\mathbf{C_i}$:

$$\hat{\mathbf{C}}_{\mathbf{i}} = W^{1/2} \sum_{a=1}^{m} \frac{1}{1 - \mu\lambda_a} \bigvee^{(a)} \bigvee_i^{(a)} W_{ii}^{-1/2} \tag{21}$$

To simplify the equation, let $Z$ be a $N \times m$ matrix such that:

$$Z_{ij} = \frac{\bigvee_i^{(j)} W_{ii}^{-1/2}}{1 - \mu\lambda_j} \tag{22}$$

Substitute in *Equation 21* and get:

$$\hat{\mathbf{C}}_{\mathbf{i}} = W^{1/2} \sum_{a=1}^{N} Z_{ia} \bigvee^{(a)} \tag{23}$$

The dot product between $\hat{\mathbf{C}}_{\mathbf{i}}$ and $\hat{\mathbf{C}}_{\mathbf{j}}$ becomes:

$$\hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}}\hat{\mathbf{C}}_{\mathbf{j}} = \left(W^{1/2}\sum_{a=1}^{m}Z_{ia}\bigvee^{(a)}\right)^{T}\left(W^{1/2}\sum_{a'=1}^{m}Z_{ia'}\bigvee^{(a')}\right) \tag{24}$$

$$\Leftrightarrow \hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}}\hat{\mathbf{C}}_{\mathbf{j}} = \sum_{a=1}^{m}Z_{ia}\bigvee^{(a)^{T}}W\sum_{a'=1}^{m}Z_{ia'}\bigvee^{(a')} \tag{25}$$

$$\Leftrightarrow \hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}}\hat{\mathbf{C}}_{\mathbf{j}} = \sum_{a=1}^{m}\sum_{a'=1}^{m}Z_{ia}Z_{ja'}\bigvee^{(a)^{T}}W\bigvee^{(a')} \tag{26}$$

Define a new $m \times m$ matrix $Q$ such that:

$$Q_{ij} = \bigvee^{(i)^{T}}W\bigvee^{(j)} \tag{27}$$

Substitute $Q$ into *Equation 26* and obtain:

$$\hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}}\hat{\mathbf{C}}_{\mathbf{j}} = \sum_{a=1}^{m}\sum_{a'=1}^{m}Z_{ia}Z_{ja'}Q_{aa'} \tag{28}$$

$$\Leftrightarrow \hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}}\hat{\mathbf{C}}_{\mathbf{j}} = \mathbf{Z}_{\mathbf{i}}^{\mathbf{T}}Q\mathbf{Z}_{\mathbf{j}} \tag{29}$$

From *Equation 29* it is deduced that it is only required to store the matrixes $Q$ and $Z$, which only takes $O(m^2 + Nm)$ memory, instead of $O(N^2)$ to store all $\hat{\mathbf{C}}_{\mathbf{i}}$ vectors.

To compute the similarity between two vertices, $i$ and $j$, use the norm formula from *Equation 29* and obtain:

$$\|\hat{\mathbf{C}}_{\mathbf{i}}^{\mathbf{T}} - \hat{\mathbf{C}}_{\mathbf{j}}\|^2 = \mathbf{Z}_{\mathbf{i}}^{\mathbf{T}}Q\mathbf{Z}_{\mathbf{i}} + \mathbf{Z}_{\mathbf{j}}^{\mathbf{T}}Q\mathbf{Z}_{\mathbf{j}} - 2\mathbf{Z}_{\mathbf{i}}^{\mathbf{T}}Q\mathbf{Z}_{\mathbf{j}} \tag{30}$$

After computing the matrices $Z$ and $Q$, the time complexity of computing the similarity between two nodes is therefore the same as multiplying three small matrices (of sizes $1 \times m$, $m \times m$ and $m \times 1$).

## Vectorisation

It is usually more efficient to have vectorised implementations (to use matrix operations in favour of iterative solutions) in programming languages like Matlab, Octave or R. In this subsection is presented a way of computing the matrices $Z$ and $Q$ using matrix operations. For this purpose, in this subsection, $\bigvee$ only contains the computed eigenvectors (is of size $N \times m$), and $\bigwedge$ only contains the computed eigenvalues (is of size $m \times m$).

To vectorise the computation of $Z$ from *Equation 22*, define a new diagonal matrix $R$:

$$R = \begin{bmatrix} \frac{1}{1-\mu\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{1-\mu\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \cdots & \frac{1}{1-\mu\lambda_m} \end{bmatrix} \tag{31}$$

And then define $Z$ as:

$$Z = W^{-1/2} \bigvee R \tag{32}$$

A vectorised version of *Equation 27* (to compute $Q$):

$$Q = \bigvee^T W \bigvee \tag{33}$$

## LIMITATIONS OF CURRENT IMPLEMENTATION

The current implementation of this algorithm has various limitations which are discussed along with possible improvements.

**Only undirected graphs**    Directed graphs are not currently supported. In practice, datasets have meaningful unidirectional relationships (e.g. in a social network person A follows person B, but B does not follow A), and often datasets are represented as directed graphs rather than undirected graphs. The algorithm can be adapted to support both directed and undirected graphs but it will have the disadvantage of requiring to compute $V^{-1}$ (for undirected graphs, $V^{-1} = V^T$).

**Ignored relationship types**    In the real world datasets might have different types of relationships between entities. Some relationship types might be more relevant than others in finding a specific result. For instance, in a social network two people being friends might be more relevant for recommending new friends than two people following the same topic. The current implementation of the algorithm can compute similarities if it is given a weighted (symmetric) adjacency matrix but it does not have a way to automatically obtain these weights from the dataset.

**Not distributed**    What if the dataset is too large to fit into main memory? The algorithm is currently only designed to run on one machine. Methods of distributing the algorithm on more than one machine over a network will be investigated in the future.

**Lacks systematic evaluation**    Evaluation of the performance of the algorithm is crucial in the development process. Currently there is no automatic way of evaluating the performance. Methods of automatically and objectively evaluating the performance will be analysed and implemented in the future.

# REFERENCES

[1]: Calvetti, Daniela, L. Reichel, and Danny Chris Sorensen. "An implicitly restarted Lanczos method for large symmetric eigenvalue problems." Electronic Transactions on Numerical Analysis 2.1 (1994): 21.

[2]: Lehoucq, Richard B., and Danny C. Sorensen. "Deflation techniques for an implicitly restarted Arnoldi iteration." SIAM Journal on Matrix Analysis and Applications 17.4 (1996): 789-821.