N (2020) p1-pN Submitted MM/YY; Published MM/YY

Hierarchical Similarity via Quotient Graphs - DRAFT

Yizhar (Izzy) Toren YTOREN@GMAIL.COM *Berlin, Germany*

Editor: TBD

Abstract

We present a fast and scalable algorithm for calculating similarity across different levels of hierarchical data, using a specific class of aggregation functions. Our algorithm takes an existing similarity matrix between a set of low-level items and aggregates it values into the similarity matrix between a known set of higher-level entities. We frame the hierarchical similarity problem in terms of quotient graphs and matrix representations, describe our algorithm in terms of matrix operations and provide a proof of correctness. We also provide links to implementations in R and python, discuss possible extension to our framework and provide an illustrative numerical example.

Keywords: Similarty, Hierarchical Data, Quotient Graphs, Matrix Representation

Note: This is a *draft* written as part of a submission process.

1. Introduction

Many similarity problems involve data that contains hierarchical relationships. When these relationships are known, we are often interested in deducing the similarity between the set of higher level entities from similarity between the set of lower level items. Some types of hierarchical data relationships provide a natural way to "flatten" the data from the item level to the entity level, which allows for a direct calculation of similarity. For example: repeated measures, time series data, bipartite graphs (users selecting products), etc. However this does not hold in the general case, specifically when items are multi-dimensional (e.g. patients with image data) or when there is no natural order or align the items (e.g. products with product review texts). For such cases we would like to establish a meaningful of aggregating item-level similarities to represent the similarity between the linked entities of interest.

Previous work in this field deals primarily with cases where the hierarchical relationships are unknown or the set of higher-level entities is undetermined. Solutions to these cases range from graph partitioning algorithms (see a review by Buluç et al. [2016]) to block models or matrix-based clustering (see Doreian et al. [2005]).

In contrast, this paper focuses on cases where the hierarchy is already known in the form of a partition of the low-level items. We assume there is a well defined measure of similarity between each pair of low-level items, and that the calculation of this measure is reasonably fast and scalable. Based on the know similarity matrix, we offer a fast and scalable way to calculate the similarity between the higher level entities as defined by the groups in the partition. Our calculations are limited to a class of aggregation functions (see section 2.3), but can be easily extended to other cases (see discussion).

The reminder of the paper is divided as follows: We begin with a formal definition of the problem using similarity matrices and quotient graphs. Next we describe our proposed algorithm and prove that it correctly constructs the quotient graph. Finally we discuss possible extension to our algorithms, possible future research and provide a numeric example.

2. Formal definition

2.1 Similarity representations: matrix vs. graph

Let Φ be a features space of interest and let $X = \{x_1, \ldots, x_N\} \subset \Phi$ be a set of N items from the feature space. Examples of feature spaces can be $\mathbb{R}^m$ for m continuous features (or 0/1 encoded discrete data), all $m_1 \times m_2$ RGB images (equivalent to $[0, 1, \ldots, 255]^{m_1 \times m_2 \times 3}$), all texts of length m_3 or any product of such spaces (in the case of mixed data).

Let $d : \Phi \times \Phi \to [0, 1]$ be a distance measure that expresses the pairwise similarity between elements of X as:

$$\text{similarity}(x_i, x_j) = 1 - d(x_1, x_j) \quad \forall i, j \in [1, \ldots, N] \tag{1}$$

The values in (1) are typically represented as a similarity matrix $S \in [0, 1]^{N \times N}$ where each element $s_{i,j} = \text{similarity}(x_i, x_j)$ represents the similarity between all possible pairs of items in X. Note that by definition $d(x_i, x_j) = d(x_j, x_i)$ which makes S symmetric ($S_{i,j} = S_{j,i}$) and $s_{i,i} = 1 \, \forall i \in 1, \ldots N$.

An equivalent representation of the similarity structure is as a fully connected weighted graph $G = (V, E, W)$, where each element of X is a vertex ($V = X$) and:

$$\forall e_{i,j} \in E \;\; w_{i,j} = \text{weight}(e_{i,j}) = s_{i,j} \tag{2}$$

A more compact representation is typically achieved simply by removing all edges where $w_{i,j} = 0$, but for the purpose of this paper we assume a fully connected graphs with 0 valued weights. For more details about this construction see Cormen et al. [2001].

2.2 Hierarchy via index partitions

To represent the fact that some items in X are related to one another in some arbitrary way (in our case - belong to the same higher-level entity), we introduce $p = \{p_1, \ldots, p_L\}$, a partition on the row-indices of X, which has the following properties:

$$\begin{aligned} &\text{(i)} && p_i \subset [1, \ldots, N] \\ &\text{(ii)} && p_i \neq \emptyset \\ &\text{(iii)} && p_i \bigcap p_j = \emptyset \\ &\text{(iv)} && \bigcup_{i=1}^{L} p_i = [1, \ldots, N] \end{aligned} \tag{3}$$

for all $i, j \in [1, \ldots, L]$, where $\emptyset$ is the empty set. Note that since we require all elements of p to be non-empty, it follows that $L \leq N$, which fits nicely with our idea of hierarchy: p "maps" single elements of X to a smaller set of L higher level entities.

2.3 The aggregation function

In order to aggregate elements of S we must first define an aggregation function σ. Our choice of σ depends on the problem we are trying to solve: In some cases we may aim to keep the similarity matrix structure (for example functions that will return real values in $[0, 1]$ such as average weight or maximal weight); other cases may require different aggregations, such as total weights, number of weights above a threshold etc. In this paper we restrict our discussion functions that fulfil the following conditions:

$$\begin{aligned} &\text{(i)} \quad \sigma(w) \geq 0 \\ &\text{(ii)} \quad \sigma(w) = 0 \Leftrightarrow \forall w_k \in w.\, w_k = 0 \end{aligned} \tag{4}$$

2.4 Aggregating similarity via quotient graphs

Following the definition in Buluç et al. [2016] (section 2) we cconstruct the quotient graph

$$G^{(p,\sigma)} = \left(V^{(p)}, E^{(p,\sigma)}, W^{(p,\sigma)}\right) \tag{5}$$

by applying the following steps:

1. Using the partition p we group together elements of the original vertex set V to create a "set of sets" (or "blocks") $V^{(p)}$, where each vertex represents of a set of items in X that are connected to the same entity, as defined by p:

$$V^{(p)} = \{\{X_k | k \in p_1\}, \ldots, \{X_k | k \in p_L\}\} \tag{6}$$

2. We aggregate the weight set W to match by applying our aggregation function σ. The quotient weights are calculated by taking all possible pairs of vertex sets $(\{X_k | k \in p_i\}, \{X_l | l \in p_j\})$ and use σ to aggregate the weights on edges that link the two sets in the original graph G:

$$w_{i,j}^{(p,\sigma)} = \sigma\left(\{w_{k,l} | k \in p_i, l \in p_j\}\right) \tag{7}$$

   Note that when $i = j$ the weights $w_{i,i}^{(p.\sigma)}$ represent an aggregation of the intra-set (or intra-block) weights, for example the total sum of weights on edges between the nodes $\{X_k | k \in p_i\}$ in the original graph G. Depending on the problem at hand, we may choose to ignore these values, arbitrarily set them to 1 (if we are to maintain the similarity graph/matrix conventions), or aggregated as:

$$w_{i,i}^{(p,\sigma)} = \sigma\left(\{w_{k,l} |\, (k, l) \in p_i \times p_i\}\right) \tag{8}$$

3. Since we limit our discussion fully connected graphs, the quotient graph can also defined as fully connected, simply by the fact that there is at least one edge connecting any pair of vertex sets $\{X_k | k \in p_i\}, \{X_l | k \in p_j\}$. As before we can opt for a compact representations by setting

$$E^{(p,\sigma)} = \{(i, j) |\, w_{i,j}^{(p,\sigma)} > 0\} \tag{9}$$

Similar to (2), these definitions allow us to define the "quotient similarity matrix" $S^{(p,\sigma)} \in \mathbb{R}^{L \times L}$, which represents the aggregated similarity between the L higher level entities, where

$$S^{(p,\sigma)}_{i,j} = w^{(p,\sigma)}_{i,j} \tag{10}$$

3. The algorithm

3.1 Design considerations

We aim to provide a fast and scalable algorithm to construct the quotient similarity matrix $S^{(p,\sigma)}$ given a similarity matrix S, an index partition p and an aggregation function σ. While it is possible to describe our algorithm in pseudo-code (for example as a series of loops over the nodes / edges of the graph G), we decided instead to focus on matrix operations, which have some beneficial properties:

- *Portability:* Most modern languages support data structures and operations over real valued matrices as part of the base language or by using extension packages. Once we formulate our algorithm in such terms it becomes easy to implement it in a concise and short form across many languages and systems.
- *High level syntax with low level speed:* Many high level programming languages such as R (R Core Team [2020]) and Python's NumPy (Harris et al. [2020]) implement matrix operations using APIs to pre-complied code in C or other low-level languages. This means what these calculations are typically much faster when naive implementations in the original language using for/while loops.
- *Scalability and parallelisation:* Several frameworks for distributed large-scale computations support matrix representations / operation when either N or L are large. Both TensorFlow and Apache Spark support large basic matrix operations (Zaharia et al. [2016], Abadi et al. [2016]).

3.2 Matrix row aggregation operator

Let $\bigoplus$ be an operation over $\mathbb{R}$ with the following attributes $\forall r_i, r_j, r_k \in \mathbb{R}$:

$$\begin{aligned}
&\text{(i) closed:} && r_i \bigoplus r_j \in \mathbb{R} \\
&\text{(ii) commutative:} && r_i \bigoplus r_j = r_j \bigoplus r_i \\
&\text{(iii) distributive:} && r_i \bigoplus \left(r_j \bigoplus r_k \right) = \left(r_i \bigoplus r_j \right) \bigoplus r_k
\end{aligned} \tag{11}$$

Depending on the context of the problem, this will often be the sum operation $(r_i + r_j)$, $\min(r_i, r_j)$, $\max(r_i, r_j)$ etc. See the section 5.1 in the discussion for possible extensions of this definition.

For a given index partition p of length $L = |p|$ and an aggregation operation $\bigoplus$ we define the row aggregation operator over real-valued $m_1 \times m_2$ matrices:

$$A^{(m_1,m_2)}_{p,\bigoplus} : \mathbb{R}^{m_1 \times m_2} \to \mathbb{R}^{L \times m_2} \tag{12}$$

which take $M \in \mathbb{R}^{m_1 \times m_2}$ and returns $M' \in \mathbb{R}^{L \times m_2}$ with the following property:

$$M'_{i,j} = \left[A^{(m_1,m_2)}_{p,\bigoplus}(M)\right]_{i,j} = \bigoplus_{\{k|k\in p_i\}} M_{k,j} \tag{13}$$

In other words, for each subset of indices $p_i \in p$ the operator selects the corresponding subset of rows of the matrix M and performs a columns-wise aggregation of the values using the operation $\bigoplus$. The result is a set of L single rows $\{M'_{i,\cdot}\}_{i=1}^{L}$ with aggregated elements, that can be "stacked" into the matrix M', which has one row per $p_i \in p$ and the same number of columns as the original matrix M.

WLOG we can simplify the notation by using the fact that (1) the algorithms is agnostic to the number of columns in M and that (2) the number of rows and columns is already "encoded" in well defined row-partition p. We denote $L = |p|$, $N = \left|\bigcup_{i=1}^{L} p_i\right|$ and define:

$$A_{p,\bigoplus} : \mathbb{R}^{N\times\bullet} \to \mathbb{R}^{L\times\bullet} \tag{14}$$

which is well defined for real valued matrices with N rows and and any finite of columns.

3.3 Quotient matrix operator

For a given row partition p and an operation $\bigoplus$ we define the quotient matrix operator $Q_{p,\bigoplus} : \mathbb{R}^{N\times N} \to \mathbb{R}^{L\times L}$ as a compostion of matrix operations:

$$Q_{p,\bigoplus} = T \circ A_{p,\bigoplus} \circ T \circ A_{p,\bigoplus} \tag{15}$$

where T is the matrix transpose operator. Alternatively this can be written as an operator on a square matrix $S \in \mathbb{R}^{N\times N}$:

$$Q_{p,\bigoplus}(S) = A_{p,\bigoplus}\left(A_{p,\bigoplus}(S)^T\right)^T \tag{16}$$

Theorem 1 *Let $\sigma_{\bigoplus}$ be an aggregation function that can be written in terms of an an operation $\bigoplus$:*

$$\sigma_{\bigoplus}(w) = \bigoplus_{w_k\in w} w_k \tag{17}$$

And let $S^{(p,\sigma_{\bigoplus})}$ be the equivalent matrix representation of the quotient graph $G^{(p,\sigma_{\bigoplus})}$, that was aggregated using the partition p and the aggregation function $\sigma_{\bigoplus}$

Then:

$$S^{(p,\sigma_{\bigoplus})} = Q_{p,\bigoplus}(S) \tag{18}$$

Proof Let $w^{(p,\sigma_{\bigoplus})}_{i,j} = S^{(p,\sigma_{\bigoplus})}_{i,j}$ represent the similarity between the i'th and j'th vertices of the quotient graph $G^{(p,\sigma_{\bigoplus})}$. By (7) the quotient similarity between the two vertices is calculated by applying the aggregation $\sigma_{\bigoplus}$ over the set of all weights $\{w_{l,k}|k \in p_i, l \in p_j\}$. Since our aggregation operation $\bigoplus$ is commutative and distributive we can write it as:

$$S^{(p,\sigma_{\bigoplus})}_{i,j} = w^{(p,\sigma_{\bigoplus})}_{i,j} = \sigma_{\bigoplus}(\{w_{l,k}|k \in p_i, l \in p_j\}) = \bigoplus_{\{(k,l)|k\in p_i, l\in p_j\}} w_{k,l} = \bigoplus_{\{(k,l)|k\in p_i, l\in p_j\}} S_{k,l} \tag{19}$$

From the RHS we have:

$$\left[A_{p,\bigoplus}(S)\right]_{i,j} = \bigoplus_{\{k|k\in p_i\}} S_{k,j} \tag{20}$$

Applying the transpose operator means we are now summing over the original columns instead of rows:

$$\left[T \circ A_{p,\bigoplus}(S)\right]_{i,j} = \left[A_{p,\bigoplus}(S)^T\right]_{i,j} = \left[A_{p,\bigoplus}(S)\right]_{j,i} = \bigoplus_{\{k|k\in p_i\}} S_{j,k} \tag{21}$$

Applying our column-agnostic operator again (this time on the resulting $N \times L$ matrix we get:

$$\left[A_{p,\bigoplus} \circ T \circ A_{p,\bigoplus}(S)\right]_{i,j} = \left[A_{p,\bigoplus}\left(A_{p,\bigoplus}(S)^T\right)\right]_{i,j} = \bigoplus_{\{l|l\in p_j\}} \left(\bigoplus_{\{k|k\in p_i\}} S_{l,k} \right) \tag{22}$$

which results in a square, $L \times L$ matrix. By the properties of $\bigoplus$ we can re-write this expression as:

$$\left[A_{p,\bigoplus} \circ T \circ A_{p,\bigoplus}(S)\right]_{i,j} = \bigoplus_{\{(l,k)|l\in p_j, k\in p_i\}} S_{l,k} \tag{23}$$

Applying the transpose operation again and re-arranging our indices we get:

$$\left[Q_{p,\bigoplus}(S)\right]_{i,j} = \left[A_{p,\bigoplus}\left(A_{p,\bigoplus}(S)^T\right)^T\right]_{i,j} = \bigoplus_{\{(k,l)|k\in p_i, l\in p_j\}} S_{k,l} = S_{i,j}^{(p,\sigma_\bigoplus)} \tag{24}$$

■

Since our original similarity matrix S is symmetric (1) we can simplify the calculation by omitting the last transpose operation.

Corollary 2 *For a non-directed similarity graph with a symmetric similarity matrix S':*

$$Q'_{p,\bigoplus} = A_{p,\bigoplus} \circ T \circ A_{p,\bigoplus} \tag{25}$$

And it still holds that:

$$S'^{(p,\sigma_\bigoplus)} = Q'_{p,\bigoplus}(S') \tag{26}$$

Proof The proof follows directly the symmetry of S:

$$\left[A_{p,\bigoplus} \circ T \circ A_p(S')\right]_{i,j} = \bigoplus_{\{(l,k)|k\in p_i, l\in p_j\}} S'_{l,k} = \bigoplus_{\{(k,l)|k\in p_i, l\in p_j\}} S'_{l,k} = S'^{(p,\sigma_\bigoplus)}_{i,j} \tag{27}$$

■

4. Implementation

We provide two implementations of the algorithm: an R package (link) and a Python 3 package (link). Both implementations demonstrate the efficiency achieved by using lower-level matrix APIs in a high-level language: in python we utilise the SciPy sparse API (Virtanen et al. [2020]) and in R we utilise the Matrix package (Bates and Maechler [2019]), both of which are written in high performance compiled languages. Our python package documentation also contains some benchmarks against existing implementations of quotient graph calculation using the NetworkX package (Hagberg et al. [2008]) that illustrate the efficiency of our method compared with existing implementations.

5. Discussion

5.1 Beyond simple aggregation

Our constraints on the aggregation operation $\bigoplus$ (11) and the resulting aggregation function $g_{\bigoplus}$ (17) are quite strict. It would in fact be difficult to think of operations other than summation, minimum / maximum values and different types of counts as good candidates for the aggregation operation in that context. While many types of additional modes of aggregations can be achieved by slightly modifications to the algorithm, some measure are strictly out of scope for the current implementation. Below are a few interesting cases.

5.1.1 Average and moment-based similarity

Simple arithmetic mean can be achieved indirectly by running two calculations: total similarity and counting edges and dividing the two quotient matrices element-wise. Depending on the problem we are solving, we may or may not count edges where $w_{i,j} = 0$. Additional iterations to calculate sums of squares or polynomials of higher degrees will allow the calculation of standard deviation, variance or higher moments. We can also sum up $1/S_{i,j}$ to calculate the harmonic mean, or aggregate by multiplying values and use our counts to calculate the geometric mean.

5.1.2 Order driven aggregation

In cases when there is some way to order items relative to one another (for example by recency, magnitude or some arbitrary external order) we can extend the aggregation operator to use such ordinals as part of the calculation. This will require a slightly more complex implementation of the row aggregation function, but will allow us to make position based exclusions (e.g. similarity between the two most "recent" items) and some notions of weighted sums.

5.1.3 Percentile and position based aggregation

Strictly speaking, percentile based aggregations seem to be out of scope for the current implementation. It's easy see how a median of row-medians will potentially not reflect the actual median of all values.

5.1.4 "Matrix within matrix"

A possible solution to many limitation is to abandon the concept of aggregation completely, and instead "collect" all similarity values into small matrix structure embedded in a larger matrix. We can maintain the simple row-wise aggregation: instead returning a number, we return $|p_i|$ long vector in the first step, and then combine the vectors again into a matrix. The composite operator will yield a matrix where the value of each cell is itself a $|p_i| \times |p_j|$ matrix:

$$S_{i,j}^{(p)} \in \mathbb{R}^{|p_i| \times |p_j|} \tag{28}$$

Note that the matrix "structure" $S^{(p)}$ is no longer symmetric, even if the original matrix S is symmetric.

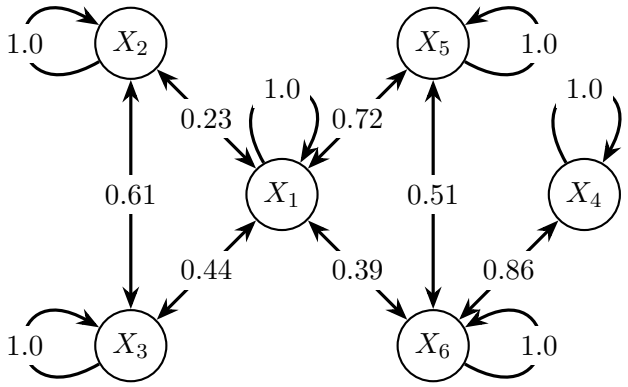
Since $S^{(p)}$ maintains the original order of the rows and columns within each matrix cell, we can use much more complex aggregation operators: percentiles (e.g. median similarity) row/columns order based (e.g. maximal similarity, but only when the row index is larger than the column index) etc. One potential downside of such structures is the fact that that not all frameworks support a "matrix of matrices" data structure as efficiently as they support real valued matrices, which can dramatically impact the speed of calculations and the efficiency of sparse memory representations.

5.2 Overlapping partitions

Another possible extension if the algorithm is to drop the requirement (i) in (3) and allow for cases where partitions overlap ($p_i \bigcap p_j \neq \emptyset$). This will allow us to handle cases where an item X_k belongs to more than one partition, but requires our algorithm to aggregate the same similarity multiple times (or equivalently, aggregate some rows/columns of S more than once). In principle, relaxing this requirement should entail significant changes to the algorithm or to pur implementations, but we leave the proof and possible extensions of this notion (for to bi-partite graphs) to future work.

Appendix A. Examples

Let $X = \{x_1, \ldots, x_6\}$ be a set of 6 product reviews, and let S be the 6x6 similarity matrix between the 6 texts, who's values $S_{i,j}$ were calculated using an appropriate similarity measure.



$$S = \begin{pmatrix} 1 & 0.23 & 0.44 & & 0.72 & 0.39 \\ 0.23 & 1 & 0.61 & & & \\ 0.44 & 0.61 & 1 & & & \\ & & & 1 & & 0.86 \\ 0.72 & & & & 1 & 0.51 \\ 0.39 & & & 0.86 & 0.51 & 1 \end{pmatrix}$$

Let $U = \{u_1, u_2, u_3\}$ be a set of 3 users that generated the reviews, where u_1 generated $\{t_1, t_2, t_3\}$, u_2 generated $\{t_4\}$ and u_3 generated $\{t_5, t_6\}$. We can describe the corresponding index partition

$$p = \{\{1, 2, 3\}, \{4\}, \{5, 6\}\}$$

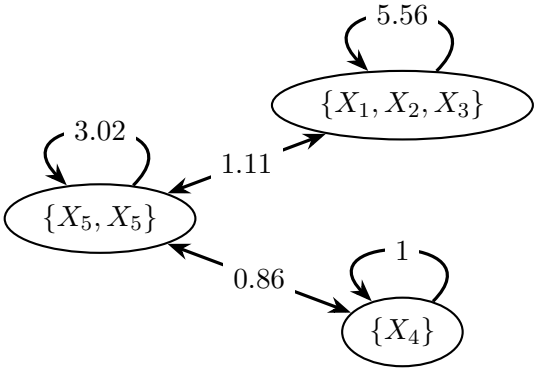
Below are some example of different aggregation functions and the resulting quotient similarity matrices.

A.1 Total similarity

We define:

$$\sigma(w) = \sum_{\{w_k \in w\}} w_k$$

and after aggregation we get:



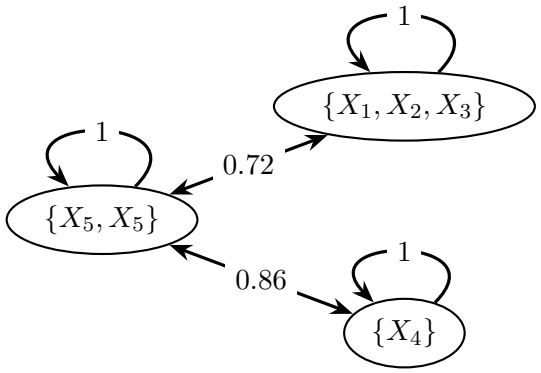
$$S^{(p,sum)} = \begin{pmatrix} 5.56 & 0 & 1.11 \\ 0 & 1 & 0.86 \\ 1.11 & 0.86 & 3.02 \end{pmatrix}$$

A.2 Maximal similarity

We define:

$$\sigma(w) = \max_{\{w_k \in w\}} w_k$$

and after aggregation we get:



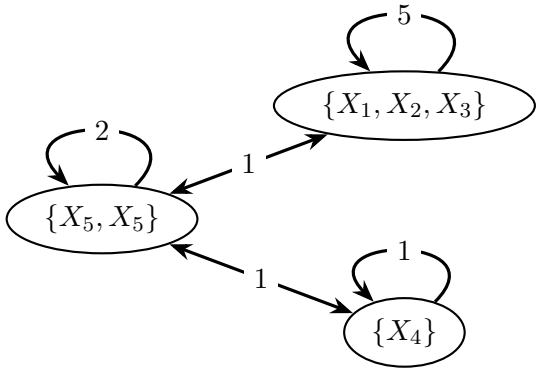
$$S^{(p,max)} = \begin{pmatrix} 1 & 0 & 0.72 \\ 0 & 1 & 0.86 \\ 0.72 & 0.86 & 1 \end{pmatrix}$$

A.3 Conditional counts

In this example we count how many item-to-item similarities crossed an arbitrary threshold of 0.6. We define:

$$\sigma(w) = |\{w_k \in w\} | w_k \geq 0.6|$$

and after aggregation we get:



$$S^{(p,\#0.6)} = \begin{pmatrix} 5 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2019. URL `https://CRAN.R-project.org/package=Matrix`. R package version 1.2-18.

Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. 2016. ISBN 978-3-319-49486-9. URL `http://dblp.uni-trier.de/db/series/lncs/lncs9220.html#BulucMSS016`.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 22.1, page 527. MIT Press, second edition, 2001. ISBN 0-262-03293-7.

P. Doreian, V. Batagelj, and A. Ferligoj. *Generalized Blockmodeling*. Structural Analysis in the Social Sciences. Cambridge University Press, 2005. ISBN 9780521840859. URL `https://books.google.de/books?id=-q59yzR6LGAC`.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org`.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0:

Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016. URL `http://dblp.uni-trier.de/db/journals/cacm/cacm59.html#ZahariaXWDADMRV16`.