Power Iteration Graph Clustering With functools Higher Order Methods

Georgios Drakopoulos
Department of Informatics
Ionian University
0000-0002-0975-1877

Phivos Mylonas
ICE Department
University of West Attica
0000-0002-6916-3129

Abstract

Graph mining operations take place on an unprecedented scale, dictating the need for scalability in both algorithms and implementation. In the context of graph partitioning, which tantamounts to community structure discovery, power iteration clustering (PIC) is an important and efficient method since it computes the primary eigenvector of a graph or of the Laplacian thereof. PIC converges for broad categories of matrices and can be applied to non-normal ones. Functional paradigm aspects like list transformations as well as capabilities like partially implemented objects and cached results are offered by the Python module functools. These can be combined with other recent major Python concepts such as decorators and lambda expressions to yield elegant and efficient code. PIC being a matrix free method can benefit from functools. As a concrete application a functional implementation of PIC has been applied to two Twitter graphs with different characteristics with encouraging results as evaluated by the Shilouette score and the Gini index.

1. Introduction

Twitter abounds with activity regarding a plethora of topics ranging from historical or cultural events to political campaigns. Major yet simple and efficient mechanisms like tweets, retweets, replies, and mentions promote online activity, frequently driven by incendiary comments, witty responses, or controversial hashtags. Out of this activity after a transient phase communities can be identified based on structural and functional criteria. These communities may well form the basis of a lower resolution view of the graph, allowing easier visualization or locating isolated segments.

978-8-3315-0450-2/24/$31.00 ©2024 IEEE

Table 1. Notation synopsis.

Symbol	Meaning	First in
$\triangleq$	Definition or equality by definition	Eq. (1)
$\{s_1, \ldots, s_n\}$	Set with elements $s_1, \ldots, s_n$	Eq. (16)
$\lvert S \rvert$	Set, tuple, or sequence cardinality	Eq. (15)
$\deg(v)$	Degree of vertex v	Eq. (18)
$\Gamma(u)$	Neighborhood of u	Eq. (17)
$\lVert \cdot \rVert$	Vector or matrix norm	Eq. (4)
$\dim(\mathcal{V})$	(Sub)space dimension	Eq. (11)
$\oplus$	Direct (sub)space sum	Eq. (10)
$\det(\mathbf{M})$	Matrix determinant	Eq. (9)
$\mathrm{prob}\{\Omega\}$	Probability of event Ω	Eq. (12)

One way to obtain graph community structure is the *power iteration clustering* (PIC), a linear algebraic scheme directly deriving from the *power method*. The latter is an iterative scheme for approximating the primary eigenvector of a matrix [1]. The critical observation is that PIC terminates early during the execution of the power method, specifically during the transition from phase of disarray between the elements of the candidate eigenvector to a phase of an initial order emerging among them. It is worth mentioning that PIC has been reported to have been used by Google to cluster documents and by Twitter for recommendations[1]. Also PIC has been applied to scientometrics, long supply chain management and logistics, maritime transportation networks, and human omics.

Because of the sheer size of contemporary social graphs as well as the complexity of the mining tasks, typically involving attribute aggregation over a neighborhood of depth more than one, an efficient implementation is essential. Moreover, the distributed nature of the information stored in graphs should be taken under consideration. One way to do this is to exploit the capabilities of *functools*[2], a Python module

1. https://en.wikipedia.org/wiki/Power_iteration
2. https://docs.python.org/3/library/functools.html

designed for functional programming and higher order functions. Observe that these capabilities are analogous to those of the MapReduce programming paradigm.

The primary research objective of this conference paper is a functional implementation over functools of PIC for deriving partitionings of Twitter graphs based on a multitude of attributes. As a secondary objective, certain remarks about PIC convergence and initialization are made. The proposed methodology is versatile enough to be applied to any field as long as the proper attributes have been identified.

The remaining of this work is structured as follows. In section 2 the scientific literature regarding PIC, graph mining, and functional programming is briefly overviewed. PIC is explained in section 3 and its functional implementation in section 4. The results obtained are given in section 5. Possible future research directions are the focus of section 6. Capital boldface letters denote matrices, lowercase boldface vectors, and normal lowercase scalars. Also, capital calligraphic letters symbolize linear spaces. Technical acronyms are explained the first time they are encountered in text. Finally, table 1 summarizes the notation of this text.

2. Previous Work

PIC has been proposed among others in [1] and expanded in [2]. PIC can be considered as a graph mining technique as data point affinity can be graphically expressed [3]. High dimensional considerations for clustering are explored in [4], while parallel versions are proposed in [5] and in [6]. The connection between graph neural networks (GNNs) and PIC is explored in [7]. The relationship between PIC and clustering is the focus of [8]. In [9] a semi-supervised version of PIC is derived. PIC variants are compared in [10] and a parallel pack for power method is described in [11]. Due to its simplicity the power method has been extended to numerous embeddings [12] and has been combined with k-means [13]. Adaptive versions have been proposed in [14]. Computing eigenvectors of large sparse matrices can be done in parallel [15], while a deflated PIC variant is described in [16]. PIC also has close ties to the field of graph mining [17] as it can be seen as a simulated walk on a weighted graph [18].

Discovering graph community structure is an open problem in social network analysis [19][20], mainly because of the task complexity [21] and of the multiple and equally valid community definitions [22][23]. Algorithmic approaches include graph convolutional networks (GCNs) [24], GNNs [25], non-negative matrix factorization [26], genetic algorithms [27], and non-linear optimization [28]. With the advent of graph signal processing [29] interest shifted to community discovery with signal processing methodologies like community aware modularity [30] and polar decomposition [31]. Applications include graph databases [32], knowledge graphs [33], and multiview [34], data point density [35], and biomedical document clustering [36].

Functional data structures [37] and programming languages [38] have come a long way towards becoming an alternative for large scale computing [39]. A prime exaple is Scala [40] which can be used to write applications over distributed processing environments such as Spark [41]. Although Python is not a purely functional language, it supports functional programming in a proper Pythonic way [42]. The way lambda expressions in Python can form the basis for a quantum computing language are explored in [43].

3. Algorithmic approach

3.1. Power iteration clustering

The similarity or affinity matrix $\mathbf{A}$ of a set of n data vectors $\mathbf{x}_k$ is elementwise constructed as shown in equation (1). Depending on the concentration of values of the normalized similarity metric $g(\cdot,\cdot)$, matrix $\mathbf{A}$ typically is diagonally dominant or close to it.

$$\mathbf{A}[i,j] \triangleq \begin{cases} g(\mathbf{x}_i,\mathbf{x}_j), & i \neq j \\ 1, & i = j \end{cases} \in [0,1]^{n\times n} \quad (1)$$

As a concrete example, consider a set consisting of four vectors numbered from $\mathbf{x}_1$ to $\mathbf{x}_4$. Then, the similarity matrix $\mathbf{A}$ has the form of equation (2).

$$\mathbf{A} = \begin{bmatrix} 1 & g(\mathbf{x}_1,\mathbf{x}_2) & g(\mathbf{x}_1,\mathbf{x}_3) & g(\mathbf{x}_1,\mathbf{x}_4) \\ g(\mathbf{x}_1,\mathbf{x}_2) & 1 & g(\mathbf{x}_2,\mathbf{x}_3) & g(\mathbf{x}_2,\mathbf{x}_4) \\ g(\mathbf{x}_1,\mathbf{x}_3) & g(\mathbf{x}_2,\mathbf{x}_3) & 1 & g(\mathbf{x}_3,\mathbf{x}_4) \\ g(\mathbf{x}_1,\mathbf{x}_4) & g(\mathbf{x}_2,\mathbf{x}_4) & g(\mathbf{x}_3,\mathbf{x}_4) & 1 \end{bmatrix} \quad (2)$$

The power method (PM) is an iterative linear algebra technique for computing the primary eigenvector $\mathbf{s}_1$ of a matrix $\mathbf{A}$, namely the eigenvector corresponding to the eigenvalue with the largest magnitude λ_1. In case its algebraic multiplicity is more than one, then PM converges to the eigenvector with the strongest component in the starting vector. This, along with

the requirement that the latter contains at least a component of the (or any) primary eigenvector, leads to a strategy of a small set of orthogonal starting vectors.

In view of the interpretation of the affinity matrix as a graph adjacency matrix, a PM or PIC iteration as shown in algorithm 1 simulates one step of a walk on the graph. In this sense PIC works akin to the way a Markov chain works and the normalized eigenvector approximation corresponds to an approximation to its stationary distribution under ergodicity assumptions. Said eigenvector is positive as a result of the Perron-Frobenius theorem [44] and can reveal the cluster structure. Also it has connections to graph Laplacian.

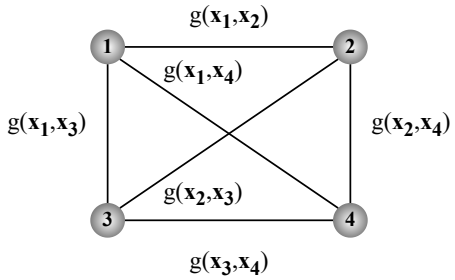

Figure 1. Graph from affinity matrix.

PIC, shown in algorithm 1, is based on PM. The critical difference between these two methods is that PIC terminates earlier and more controlled than PM based on a set of criteria derived from mechanics as shown in equations (5), (6), and (7) below.

Algorithm 1 PIC outline

Require: Matrix $\mathbf{A}$, termination criterion τ
Ensure: Obtain a primary eigenvector approximation

1: **if** random initialization is required **then**
2: initialize $\mathbf{x}^{[0]}$ as in (9)
3: **else**
4: read starting point $\mathbf{x}^{[0]}$
5: **end if**
6: normalize $\mathbf{x}^{[0]}$ as in (4)
7: **repeat**
8: compute $\mathbf{u}^{[k]}$ as in (3)
9: normalize $\mathbf{u}^{[k]}$ as in (4)
10: replace $\mathbf{x}^{[k]}$ with $\mathbf{u}^{[k]}$
11: **until** τ is **true**
12: **return** $\mathbf{u}^{[k]}$ as $\mathbf{s}_1$

PIC relies on repeated applications of $\mathbf{A}$ on $\mathbf{x}$ and normalization of the result as shown in equations (3) and (4). The result of the latter is the candidate primary eigenvector of the current iteration. Since the matrix-vector multiplication of (3) is the most computationally expensive operation of PIC, it should be optimized. This is indeed the focus of a large body of research.

$$\mathbf{u}^{[k]} = \mathbf{A}\mathbf{x}^{[k]} \tag{3}$$

The unit length normalization of equation (4) ensures numerical stability and alleviates the effect of successive matrix multiplications keeping only the eigenvector approximation direction.

$$\mathbf{u}^{[k]} = \mathbf{u}^{[k]} / \left\| \mathbf{u}^{[k]} \right\|_2 \tag{4}$$

PM undergoes four phases as listed below. PIC terminates once the second one is reached.

- Initially, the values of vector $\mathbf{u}^{[k]}$ are in disarray and convergence is relatively slow.
- For a few iterations the values of $\mathbf{u}^{[k]}$ are clustered and convergence slows down.
- After that $\mathbf{u}^{[k]}$ converges rapidly to the primary eigenvector of the input matrix $\mathbf{A}$.
- Once $\mathbf{u}^{[k]}$ has converged to the primary eigenvector, which is a stationary point.

Observe that algorithm 1 is matrix-free, like the Lanczos and the conjugate gradient algorithms, in the sense that $\mathbf{A}$ need not be explicitly known. Instead, it suffices that there is way to directly or indirectly compute (3). This is useful when $\mathbf{A}$ has a special structure, when for instance is a filtering matrix, it can be expressed as a product of Givens rotations [45] or Householder reflections [46], or is (approximately) banded as in the discretization of differential equations. Given that most social media accounts tend to frequently interact with a relatively small number of other accounts, then the affinity matrix will be sparse and approximately banded. The reverse symmetric Cuthill-McKee algorithm can reveal this structure. Common mechanisms for obtaining the result of (3) are linear functionals, probabilistic approximations, low rank approximations, incomplete factorizations, and preconditioned versions of $\mathbf{A}$. Such mechanisms reduce the memory or complexity requirements. This may be particularly attractive in distributed or parallel environments where communication between computational nodes is the primary bottleneck factor.

PIC convergence relies on the spectral properties of $\mathbf{A}$, and specifically on gap between successive eigenvalues. When the latter are clustered, then the

power method is significantly accelerated. However, this information is typically inaccessible or too expensive to obtain. In this case, the condition number or an estimation thereof, such as the Hager method, can serve as a crude indicator of the convergence rate. The eigenvalues can be computed in a number of ways such as the Rayleigh quotient and the characteristic polynomial of the companion matrix.

The criterion τ is critical for PIC to stop at the second phase as stated earlier. To this end, the elementwise speed and acceleration during the k-th iteration as shown respectively in equations (5) and (6).

$$v^{[k,j]} \triangleq \left| \mathbf{u}^{[k]}\left[j\right] - \mathbf{u}^{[k-1]}\left[j\right] \right| \tag{5}$$

Similarly the acceleration is the elementwise absolute second order difference of $\mathbf{u}^{[k]}$.

$$a^{[k,j]} \triangleq \frac{\left| \mathbf{u}^{[k]}\left[j\right] - 2\mathbf{u}^{[k-1]}\left[j\right] - \mathbf{u}^{[k-2]}\left[j\right] \right|}{2} \tag{6}$$

The overall acceleration during a given iteration shown in equation (7) is computed as the harmonic mean of the elementwise accelerations since it can handle very small or zero values and also it is less prone to outliers compared to the arithmetic mean.

$$a^{[k]} \triangleq \frac{n}{\sum_{j=1}^{n} \frac{1}{a^{[k,j]}}} \tag{7}$$

PIC can also be applied to non-normal operators, including matrices which do not commute with its transpose. In this case convergence is achieved in the sense that for a matrix $\mathbf{N}$ the primary eigenvector $\mathbf{y}$ and eigenvalue μ there exists a scalar β such that:

$$\left\| \frac{\mathbf{N}^k \mathbf{y}}{\mu^k} - \beta \mathbf{y} \right\| \to 0 \tag{8}$$

Given algorithm 1, a reasonable question is what constitutes a good starting vector. One solution is to select a random vector as this will not have an effect on the convergence itself as long as this starting point is not perpendicular to the space spanned by the primary eigenvector of the distance matrix. The bigger this component is, the quicker the convergence. One way to do this use a multivariate Gaussian distribution with mean vector $\mathbf{c}$ and covariance matrix $\mathbf{C}$ as shown in equation (9). The Gaussian distribution has the maximum differential entropy among all distributions with the same covariance matrix, thus it can explain the largest possible number of probabilistic scenarios.

$$\mathbf{x}^{[0]} \sim \frac{1}{\left((2\pi)\det\left(\mathbf{C}\right)\right)^{n/2}} \times \exp\left(-\left(\mathbf{x}-\mathbf{c}\right)^T \mathbf{C}^{-1} \left(\mathbf{x}-\mathbf{c}\right) \right) \tag{9}$$

A scheme to minimize the probability of selecting an inappropriate random starting point is as follows. Let $\mathcal{W}$ be the subspace generated by the primary eigenvector of $\mathbf{A}$ of algorithm 1. Then $\mathbb{R}^n$ is the direct sum of $\mathcal{W}$ and its perpendicular subspace $\mathcal{W}^{\perp}$ generated by the remaining eigenvectors as in (10).

$$\mathbb{R}^n = \mathcal{W} \oplus \mathcal{W}^{\perp} \tag{10}$$

Given (10) it follows immediately that as in (11):

$$n = \dim\left(\mathcal{W}\right) + \dim\left(\mathcal{W}^{\perp}\right) = 1 + (n-1) \tag{11}$$

Let p_0 be the probability of selecting an initial random vector perpendicular to subspace $\mathcal{W}$ as in (12).

$$p_0 \triangleq \text{prob}\left\{ \mathbf{s}^{[0]} \perp \mathcal{W} \right\} \tag{12}$$

In order to achieve a lower probability than that of equation (12), d random starting points can be selected and stacked in a $n \times d$ matrix $\mathbf{S}^{[0]}$ as in (13).

$$\mathbf{S}^{[0]} \triangleq \begin{bmatrix} \mathbf{s}_1^{[0]} & \mathbf{s}_2^{[0]} & \dots & \mathbf{s}_d^{[0]} \end{bmatrix}, \quad d \ll n \tag{13}$$

The probability q_0 that all d starting points belong to $\mathcal{W}^{\perp}$ is computed as in equation (14). When d is relatively large and p_0 is sufficiently small, then the following approximation is satisfactory:

$$q_0 = \prod_{k=1}^{d} p_0 \approx e^{-dp_0} \tag{14}$$

In this work a single starting vector has been used in the PIC to approximate the primary eigenvector.

3.2.Attributes and metrics

Matrix $\mathbf{A}$ is elementwise constructed by the composite vertex distance metric $g\left(\cdot,\cdot\right)$ shown in equation (15). Said metric is based on a set of individual metrics h, over which the sum of (15) ranges over. The transformed metric $\tilde{h}$ is dervived from the corresponding h through (21). The members of h are built on easy to compute Twitter attributes collected for each vertex.

Observe that the mathematical formulation of equation (15) can be directly cast in functional programming code similarly to the examples of section 4. Moreover, the proposed approach can be extended in a straightforward manner to an arbitrary number of vertex distance metrics, capturing thus critical structural graph aspects.

$$g\left(u_1, u_2\right) \triangleq \frac{1}{|H|} \sum_{h} \tilde{h}\left(u_1, u_2\right), h \in H \tag{15}$$

The metric set H consists of the metrics explained in equations (17) to (20). Each metric has been normalized as shown in equation (21).

$$h \triangleq \{h_a, h_c, h_r, h_t\} \tag{16}$$

The Adamic-Adar similarity metric h_a for u and v is based on information theory principles. In particular, it is the (17) is the average inverse logarithm of the degree of the vertices in their common neighborhood. Observe that all metrics therein are symmetric.

$$h_a\left(u_1, u_2\right) \triangleq \sum_{s \in \{\Gamma(u) \cap \Gamma(v)\}} \frac{1}{\ln |\Gamma(s)|} \tag{17}$$

The attachment metric h_c of (18) is a measure of the connection potential between a pair of vertices. It can be computed functionally operations as only the length of an adjacency list needs to be computed.

$$h_c\left(u_1, u_2\right) \triangleq \deg\left(u_1\right) \deg\left(u_2\right) \tag{18}$$

The harmonic mean of the followers-to-following ratios r_1 and r_2 of any two accounts u_1 and u_2 respectively is an affinity metric evaluating the influence of both accounts. The harmonic mean tends to smooth outliers and as stated earlier is less prone to numerical errors. Alternatively, the logarithms of said ratios can be used in order to quantify the affinity in the order of magnitudes of the ratios and balance large differences.

$$h_r\left(u_1, u_2\right) \triangleq \frac{2}{\frac{1}{r_1} + \frac{1}{r_2}} = 2\frac{r_1 r_2}{r_1 + r_2} \tag{19}$$

The last metric h_t is the Tanimoto similarity coefficient between the hashtag sets T_1 and T_2 of vertices u_1 and u_2 respectively. This is a functional metric indicating the functional coherency between them.

$$h_t\left(u_1, u_2\right) \triangleq \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{|T_1 \cap T_2|}{|T_1| + |T_2| - |T_1 \cap T_2|} \tag{20}$$

The second form of the Tanimoto coefficient comes from Venn diagrams and relies only on set intersection and cardinality. Both can be efficiently computed or estimated with set cardinality estimators [47].

$$\tilde{h}\left(u_1, u_2\right) \triangleq \begin{cases} \dfrac{\exp\left(h\left(u_1, u_2\right)\right)}{\sum\limits_{(u_1, u_2)} \exp\left(h\left(u_1, u_2\right)\right)}, & \text{similarity} \\ 1 - \dfrac{\exp\left(h\left(u_1, u_2\right)\right)}{\sum\limits_{(u_1, u_2)} \exp\left(h\left(u_1, u_2\right)\right)}, & \text{distance} \end{cases} \tag{21}$$

The PIC matrix relies heavily on the affinity between vertex pairs. To achieve that, the transformed metric $\tilde{h}$ is computed as in (21) depending whether metric h computes distance or affinity.

4. Functional Implementation

Graph operations can be naturally expressed in functional programming since they contain recursive higher order patterns and frequently entail list transformations. This leads to elegant and efficient code. Moreover, functional programming techniques are the basis of the MapReduce paradigm which lies at the core of Hadoop. On the contrary, the paradigm of Apache Spark is that of directed acyclic graphs (DAG).

In Python functional methods and higher order functionality are provided by the *functools* module. Higher order functions accept at least one function as an argument or return a function. The main functional method is reduce which performs transforms lists like adjacency lists. For instance the entropy of a vector excluding zero entries is computed as follows.

```
import functools as f
import numpy as np

vals = d.values()
H = f.reduce(lambda H,p: \
    H-p*np.log2(p) if p else H,\
    vals, 0)
```

Linked data may come from any iterable such as a **yield** statement or from the values of a dictionary within a function. For instance, the following segment relies on a generator to count how many vertices of an undirected graph have a degree equal to one.

```
import functools as f

def row_gen(adj):
    for row in range(adj.shape[0]):
        yield adj[row]

y = f.reduce(lambda x,r: \
    x+1 if (2 == sum(r)) \
    else x, row_gen)
```

Other Python concepts can be combined with the functional tools. A **lambda** expression can act as a lightweight constructor. Decorators can alter computation and caching results speed up computations.

5. Results

PIC has been applied to two Twitter graphs with the functional and structural attributes of table 2 taken from [31]. The two graphs are the following:

- The US2020 graph was taken from US political Twitter during a polarized and heated Presidential Election. It has mostly inflammatory comments and intense arguments.
- The 1821 graph was extracted from Greek political Twitter. It consists mainly of celebratory and positive tweets, relaxed conversations, and occasional but short arguments.

The results are shown in table 3. Therein clustering quality is evaluated in terms of the Gini index and the entropy of the hashtags in each cluster as well as the Shilouette score average and maximum intercluster distances. The entries are normalized by dividing with the maximum respective value.

The Gini index for the hashtag distribution in each category is shown in (22). Therein $f_{k,i}$ is the frequency of the i-hashtag in the k-th cluster and n_k the total number of unique hashtags in the k-th cluster.

$$g_k \triangleq \frac{\sum_{i=1}^{n_k}\sum_{j=1}^{n_k}|f_{k,i}-f_{k,j}|}{\sum_{i=1}^{n_k}\sum_{j=1}^{n_k}f_{k,i}} = \frac{\sum_{i=1}^{n_k}\sum_{j=1}^{n_k}|f_{k,i}-f_{k,j}|}{2n_k\sum_{i=1}^{n_k}f_{k,i}} \tag{22}$$

From table 3 several conclusions can be drawn. First, it is evident that PIC outperforms the two benchmarks. Therefore, PIC helps discovering latent structure in the dataset. Moreover, the Shilouette score is systematically stricter than intercluster distance, which is consistent with previously reported results. Additionally, the fractured and polarized nature of the US2020 graph resulted in less compact clusters.

6. Conclusions and future work

This conference paper focuses on finding Twitter communities with power iteration clustering (PIC), an iterative clustering method with close ties to graph mining operating on the affinity matrix of the dataset. Among its advantages are scalability, parallelism, flexibility, and matching the semantics of the underlying field through the appropriate selection of similarity metrics. The proposed methodology has been applied to two Twitter graphs with different activity characteristics with very encouraging results with respect to two well-known clustering algorithms.

This work can be extended in a number of ways. First and foremost, more affinity Twitter metrics can be used. This will be of interest to policymakers and stakeholders. Additionally, PIC can be applied to larger datasets in order for its scalability to be experimentally evaluated. Finally, PIC variants for sparse affinity matrices should be developed to accelerate computations.

Acknowledgment

This work is part of Project 451, a research initiative with a primary objective of developing novel, scalable, and interpretable higher order analytics.

References

[1] F. Lin and W. W. Cohen, "Power iteration clustering," in *ICML*. PMLR, 2010.

[2] M. Ouerfelli, M. Tamaazousti, and V. Rivasseau, "Selective multiple power iteration: From tensor PCA to gradient-based exploration of landscapes," *The European Physical Journal Special Topics*, pp. 1–16, 2023.

[3] Y. Zhang, H. Zhu, Z. Song, P. Koniusz, and I. King, "Spectral feature augmentation for graph contrastive learning and beyond," in *AAAI*, vol. 37, no. 9, 2023, pp. 11 289–11 297.

[4] A. Hernández-Cano, Y. Kim, and M. Imani, "A framework for efficient and binary clustering in high-dimensional space," in *DATE*. IEEE, 2021, pp. 1859–1864.

Table 2. Benchmark Twitter graphs.

Graph	Vertices	Edges	Triangles	Squares	Tweets	Hashtags	Unique #	Favorites
1821	133.17	2.251.77	446.513	215.387	17.465.844	21.113.091	121.322	36.994.815
US2020	147.881	2.447.224	489.773	218.633	22.773.674	25.091.112	182.094	42.114.509

Table 3. Clustering quality metrics.

Graph	Method	Gini	Entropy	Shilouette	Avergage	Maximum
1821	k-means	0.3813	0.6314	0.5933	0.6541	0.6883
	agglomerative	0.3201	0.5811	0.5728	0.6333	0.7166
	PIC	0.4366	0.6633	0.6109	0.7191	0.7683
US2020	k-means	0.3631	0.6112	0.5772	0.6106	0.6673
	agglomerative	0.3146	0.5666	0.5618	0.5512	0.6946
	PIC	0.4174	0.6575	0.5989	0.6942	0.7985

[5] W. Yan, U. Brahmakshatriya, Y. Xue, M. Gilder, and B. Wise, "*p*-pic: Parallel power iteration clustering for big data," *Journal of Parallel and Distributed computing*, vol. 73, no. 3, pp. 352–359, 2013.

[6] A. Eamaz, F. Yeganegi, and M. Soltanalian, "CyPMLI: WISL-minimized unimodular sequence design via power method-like iterations," in *ICASSP*. IEEE, 2023, pp. 1–5.

[7] X. Li and Y. Cheng, "Understanding the message passing in graph neural networks via power iteration clustering," *Neural Networks*, vol. 140, pp. 130–135, 2021.

[8] S. Khedairia and M. T. Khadir, "A multiple clustering combination approach based on iterative voting process," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 1, pp. 1370–1380, 2022.

[9] Y. Yang, R. Bie, H. Wu, S. Xu, and L. Li, "Semi-supervised power iteration clustering," *Procedia computer science*, vol. 147, pp. 588–595, 2019.

[10] M. Augier and J. G. March, *Models of a man: Essays in memory of Herbert A. Simon*. MIT Press, 2022.

[11] Y. Li, Z. Wang, and H. Xie, "GCGE: A package for solving large scale eigenvalue problems by parallel block damping inverse power method," *CCF Transactions on High Performance Computing*, pp. 1–20, 2023.

[12] H. Huang, S. Yoo, D. Yu, and H. Qin, "Diverse power iteration embeddings and its applications," in *ICDM*. IEEE, 2014, pp. 200–209.

[13] J. Xu and K. Lange, "Power k-means clustering," in *ICML*. PMLR, 2019, pp. 6921–6931.

[14] Z. Liu and M. Barahona, "Graph-based data clustering via multiscale community detection," *Applied Network Science*, vol. 5, no. 1, pp. 1–20, 2020.

[15] G. Ballard, T. Kolda, and T. Plantenga, "Efficiently computing tensor eigenvalues on a GPU," in *International Symposium on Parallel and Distributed Processing Workshops*. IEEE, 2011, pp. 1340–1348.

[16] N. D. Thang *et al.*, "Deflation-based power iteration clustering," *Applied intelligence*, vol. 39, no. 2, pp. 367–385, 2013.

[17] H. Tang, L. Guo, X. Fu, Y. Wang, S. Mackin, O. Ajilore, A. D. Leow, P. M. Thompson, H. Huang, and L. Zhan, "Signed graph representation learning for functional-to-structural brain network mapping," *Medical image analysis*, vol. 83, 2023.

[18] P. Talukdar, J. Reisinger, M. Pasca, D. Ravichandran, R. Bhagat, and F. Pereira, "Weakly-supervised acquisition of labeled class instances using graph random walks," in *Conference on Empirical Methods in Natural Language Processing*, 2008, pp. 582–590.

[19] G. Drakopoulos, I. Giannoukou, S. Sioutas, and P. Mylonas, "Self organizing maps for cultural content delivery," *NCAA*, vol. 31, no. 7, 2022.

[20] Y. Liu, X. Yang, S. Zhou, X. Liu, S. Wang, K. Liang, W. Tu, and L. Li, "Simple contrastive graph clustering," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[21] E. Müller, "Graph clustering with graph neural networks," *Journal of Machine Learning Research*, vol. 24, pp. 1–21, 2023.

[22] J. Sun, W. Zheng, Q. Zhang, and Z. Xu, "Graph neural network encoding for community detection in attribute networks," *IEEE Transactions on Cybernetics*, 2021.

[23] G. Drakopoulos, A. Kanavos, and A. Tsakalidis, "Evaluating Twitter influence ranking with system theory," in *WEBIST*. SCITEPRESS, 2016.

[24] N. S. Sattar and S. Arifuzzaman, "Community detection using semi-supervised learning with graph convolutional network on GPUs," in *Big Data*. IEEE, 2020, pp. 5237–5246.

[25] G. Drakopoulos, I. Giannoukou, P. Mylonas, and S. Sioutas, "A graph neural network for assessing the affective coherence of Twitter graphs," in *IEEE Big Data*. IEEE, 2020, pp. 3618–3627.

[26] D. Li, Q. Lin, and X. Ma, "Identification of dynamic community in temporal network via joint learning graph representation and nonnegative matrix factorization," *Neurocomputing*, vol. 435, pp. 77–90, 2021.

[27] G. Drakopoulos and P. Mylonas, "A genetic algorithm for Boolean semiring matrix factorization with applications to graph mining," in *Big Data*. IEEE, 2022.

[28] J. Zeng and H. Yu, "Effectively unified optimization for large-scale graph community detection," in *Big Data*. IEEE, 2019, pp. 475–482.

[29] G. Drakopoulos, E. Kafeza, P. Mylonas, and L. Iliadis, "Transform-based graph topology similarity metrics," *NCAA*, vol. 33, no. 23, pp. 16 363–16 375, 2021.

[30] M. Petrovic, R. Liegeois, T. A. Bolton, and D. Van De Ville, "Community-aware graph signal processing: Modularity defines new ways of processing graph signals," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 150–159, 2020.

[31] G. Drakopoulos, E. Kafeza, P. Mylonas, and S. Sioutas, "Approximate high dimensional graph mining with matrix polar factorization: A Twitter application," in *IEEE Big Data*. IEEE, 2021, pp. 4441–4449.

[32] M. Marountas, G. Drakopoulos, P. Mylonas, and S. Sioutas, "Recommending database architectures for social queries: A Twitter case study," in *AIAI*. Springer, 2021.

[33] G. Tamašauskaitė and P. Groth, "Defining a knowledge graph development process through a systematic review," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, pp. 1–40, 2023.

[34] J. Wen, Z. Zhang, L. Fei, B. Zhang, Y. Xu, Z. Zhang, and J. Li, "A survey on incomplete multiview clustering," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 2, pp. 1136–1149, 2022.

[35] S. K. Anand and S. Kumar, "Experimental comparisons of clustering approaches for data representation," *CSUR*, vol. 55, no. 3, pp. 1–33, 2022.

[36] G. Drakopoulos and A. Kanavos, "Tensor-based document retrieval over Neo4j with an application to PubMed mining," in *IISA*. IEEE, 2016.

[37] C. Okasaki, *Purely functional data structures*. Cambridge University Press, 1999.

[38] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM CSUR*, vol. 21, no. 3, pp. 359–411, 1989.

[39] D. Pollak, V. Layka, and A. Sacco, "Functional programming," in *Beginning Scala 3*. Springer, 2022, pp. 79–109.

[40] M. Hartmann and R. Shevchenko, *Professional Scala: Combine object-oriented and functional programming to build high-performance applications*. Packt Publishing Ltd, 2018.

[41] S. Chellappan and D. Ganesan, "Scala: Functional programming aspects," in *Practical Apache Spark*. Springer, 2018, pp. 1–37.

[42] S. F. Lott, *Functional Python programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*. Packt Publishing Ltd, 2018.

[43] S. Garhwal, M. Ghorani, and A. Ahmad, "Quantum programming language: A systematic review of research topic and top cited languages," *Archives of Computational Methods in Engineering*, vol. 28, no. 2, pp. 289–310, 2021.

[44] Y. Hashimoto, M. Ikeda, and H. Kadri, "Deep learning with kernels through RKHM and the Perron-Frobenius operator," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[45] T. Frerix and J. Bruna, "Approximating orthogonal matrices with effective Givens factorization," in *ICML*. PMLR, 2019, pp. 1993–2001.

[46] V. W. Neo and P. A. Naylor, "Second order sequential best rotation algorithm with Householder reduction for polynomial matrix eigenvalue decomposition," in *ICASSP*. IEEE, 2019, pp. 8043–8047.

[47] G. Drakopoulos, S. Kontopoulos, and C. Makris, "Eventually consistent cardinality estimation with applications in biodata mining," in *SAC*. ACM, 2016.