The Sparse + Low-Rank Trick for Matrix Factorization-Based Graph Algorithms

Nathan de Lara
ndelara@enst.fr
Télécom Paris
Paris, France

ABSTRACT

Matrix factorization is a central block in many graph algorithms for embedding, clustering and a number of other tasks. This block is usually the computational bottleneck of these algorithms and a naive implementation can prevent them from scaling to large datasets. However, the matrices to factorize often have a closed-form "sparse + low-rank" structure. In this paper, we show how to adapt state-of-the-art matrix factorization techniques to this class of matrices. We demonstrate that the method is highly competitive with respect to a naive implementation and that it comes at a very small extra cost as compared to the decomposition of the sparse component alone.

CCS CONCEPTS

• **Computing methodologies → Machine Learning**;

KEYWORDS

algorithms, matrix factorization, graph mining

ACM Reference format:
Nathan de Lara. 2019. The Sparse + Low-Rank Trick for Matrix Factorization-Based Graph Algorithms. In *Proceedings of KDD '19: 25th ACM SIGKDD conference on knowledge discovery and data mining, Anchorage, Alaska, August 04–08, 2019 (KDD '19),* 4 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Also known as "spectral methods", matrix factorization-based algorithms are widely used in graph mining: spectral clustering [21], spectral modularity optimization [17], manifold learning [1], graph classification [4], node ranking [19], natural language processing [14], connected-components search [5], anomaly detection [9]... As the decomposition of the matrix is usually the computational bottleneck, it has to be able to scale to large graphs in order for these algorithms to compete with iterative methods for analogous tasks [2, 6, 8, 11, 15]. Making such graph algorithms scale is mandatory to obtain more comprehensive benchmarks.

In the course of this paper, we use the following notations:

- $A \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix of directed or undirected graph,
- $B \in \mathbb{R}^{n_1 \times n_2}$ denotes the biadjacency matrix of a bipartite graph,
- m is the number of non-zero coefficients in a matrix (i.e. the number of edges in the graph),
- $d = A1$ or $B1$ denotes the out-degree vector, and $D = \text{diag}(d)$,
- $f = A^T 1$ or $B^T 1$ denotes the in-degree vector,
- $w = 1^T A1$ or $1^T B1$ is the total weight of the graph.

There are two distinct issues to tackle in order for spectral methods to scale: the matrix factorization itself, but also the storage of the input. Indeed, even though the adjacency matrix of a graph and some derived matrices are usually very sparse (random walk transition matrix, Laplacian matrix,...), other matrices of interest are dense, and in the case of large graphs, they do not fit in the RAM. Such dense matrices include

- random surfer's transition matrix [18]: $R_{ij} = \alpha \dfrac{A_{ij}}{d_i} + \dfrac{1-\alpha}{n}$,
- modularity [16]: $Q_{ij} = \dfrac{1}{w} A_{ij} - \dfrac{\gamma}{w^2} d_i d_j$,
- shifted pointwise mutual information[1] [3]: $SPMI_{ij} = \log\big(\dfrac{wA_{ij}}{d_i d_j}\big)_{A_{ij}>0} - \log(k)$,
- regularized graph Laplacian [22]: $L^\tau_{ij} = I - \dfrac{A^\tau_{ij}}{\sqrt{d^\tau_i d^\tau_j}}$, where $A^\tau_{ij} = A_{ij} + \tau/n$,
- principal components analysis matrix [20]: $\bar{A}_{ij} = A_{ij} - f_j/w$.

However, note that these matrices all have a closed-form "sparse + low-rank" decomposition (even sparse + rank-1 in this case) i.e. one can write them as $S + xy^T$ where S is a sparse matrix and x and y are vectors of appropriate dimensions. For the sake of simplicity, we only consider "sparse + rank-1" matrices. However, the extension to higher ranks is trivial.

Formally, these matrices are represented by tuples of the form $(S, (x, y))$. Storing such tuples only requires $O(m + 2n)$ memory space ($O(m + 2rn)$ if there are r rank-1 tuples) instead of $O(n^2)$ or $O(n_1 n_2)$ for $S + xy^T$. Besides, this specific structure allows to leverage powerful factorization methods as discussed in Section 2.

The identification of the low-rank factors is almost direct for all the aforementioned matrices as shown in Table 1.

2 FACTORIZATION METHOD

Computing partial spectral decomposition, i.e. the k first singular vectors or eigenvectors, is well researched problem. Standard solvers usually rely on power-iteration methods such as [13]. Faster

[1]The pointwise mutual information is set to 0 if $A_{ij} = 0$.

matrix	S (sparse)	(x, y)
random surfer	$\alpha D^{-1}A$	$(\frac{1-\alpha}{n}\mathbf{1}, \mathbf{1})$
modularity	$\frac{1}{w}A$	$(-\gamma\frac{d}{w}, \frac{d}{w})$
SPMI	$\log(wD^{-1}AD^{-1})_{A_{ij}>0}$	$(-\log(k)\mathbf{1}, \mathbf{1})$
Laplacian	$I - D_\tau^{-1/2}AD_\tau^{-1/2}$	$(-\frac{\tau}{n}\sqrt{d_i+\tau}, \sqrt{d_i+\tau})$
PCA	A	$(\frac{f}{w}, \mathbf{1})$

Table 1: Identification of sparse and low-rank factors.

methods use randomized projections to reduce dimensionality before factorizing [7]. The key point of these methods is that their computational bottlenecks are two simple primitives, namely matrix transposition and matrix-vector product.

Indeed, recall algorithm 4.4 in [7]:

Algorithm 1 Randomized Subspace Iteration

```
Require: input matrix $M$
1: Generate an $n \times 2k$ Gaussian test matrix $\Omega$.
2: Form $Y_0 = M\Omega$ and compute its QR factorization $Y_0 = Q_0R_0$.
3: for $i = 1, \dots, q$ do
4:     Form $\tilde{Y}_i = M^TQ_{i-1}$ and compute its QR factorization $\tilde{Y}_i = \tilde{Q}_i\tilde{R}_i$.
5:     Form $Y_i = M\tilde{Q}_i$ and compute its QR factorization $Y_i = Q_iR_i$.
6: $Q = Q_q$.
```

The parameter q is usually a small integer (between 1 and 3). Once Q is computed, form the small matrices $B_{eig} = Q^TMQ$ and $B_{svd} = Q^TMM^TQ \in \mathbb{R}^{2k\times 2k}$. Compute their respective k first eigenvectors using any method: $B_{eig} \approx U_{eig}\Lambda U_{eig}^T$ and $B_{svd} \approx U_{svd}\Sigma^2U_{svd}^T$. QU_{eig} and $QU_{svd} \in \mathbb{R}^{n\times k}$ are respective approximations of the k first eigenvectors and left singular vectors of M.

Now, consider the two primitives, transposition and matrix-vector product for "sparse + low-rank" matrices. The dense transposed matrix is simply $S^T + yx^T$. Hence, the transposed tuple is $(S^T, (y, x))$. In the standard compressed sparse row format[2], transposing a matrix comes at the cost of sorting its non-zero entries, hence $O(m \log m)$, whereas it comes at a constant cost for dense matrices. However, this drawback is well compensated by the efficiency of the second primitive. Indeed, consider the following matrix vector products:

$$a = (S + xy^T)z,$$
$$b = Sz + (y^Tz)x.$$

Even though these two vectors are formally equal i.e. $a = b$, computing b does not require to store a dense matrix while computing a does. Besides, computing b requires $O(m + 2n)$ floating point operations while $O(n^2)$ or $O(n_1n_2)$ operations are necessary to compute a. As a result, decomposing a sparse + low-rank matrix comes at the cost of the overhead of computing the matrix-vector product of its low-rank part with respect to the usual sparse case. Algorithms 2 and 3 summarize the two primitives.

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

Algorithm 2 Transposition

```
Require: $(S, (x, y))$.
1: return $(S^T, (y, x))$.
```

Algorithm 3 Matrix-vector product

```
Require: $(S, (x, y))$.
1: return $Sz + (y^Tz)x$.
```

In the end, the trick to decompose large "sparse + low-rank" matrices works as follows:

(1) Identify the sparse and low-rank components (See Table 1 for examples.).
(2) Overwrite transposition and matrix-vector with Algorithms 2 and 3 respectively.
(3) Plug-and-play with Halko's algorithm.

Note that, for very large graphs, the random projection matrix, which has between nk and $2nk$ floating-number coefficients, can be a memory bottleneck if $k \geq \frac{m}{n}$ (especially if the graph has simple binary edges). Therefore, the average number of neighbors of the nodes in the graph is a natural limit to the number of factors in the case of simple implementations. One way to overcome this issue is to drop the QR factorizations as suggested in [7] and form B_{eig} and B_{svd}, accessing the data chunk by chunk. However, this engineering lies beyond the scope of this paper.

3 EXPERIMENTS

We compute the partial eigenvalue decomposition and the partial singular value decomposition of the modularity matrix of 7 undirected and 7 bipartite graphs respectively. In the bipartite case, the modularity matrix is defined as $\frac{1}{w}B - \frac{\gamma}{w^2}df^T$. The graphs are collected from the Konect [12] database such that they have a varying number of nodes and edges. See Table 2 for some metadata. We extract 16 components in each case and record the computation times for our modified Python implementation of Halko's method. We compare our results to the computation times of the direct Lanczos method using Scipy [10].

We also record the running times for the *Reactome* (undirected) and *MovieLens user–tag* (bipartite) graphs for $k \in \{8, 16, 32, 64, 128, 256\}$.

All experiments are performed on a laptop equipped with an Intel i7-7820HQ CPU (2.90 GHz) and 16 GB of RAM. The code to reproduce the experiments is available online[3]. Results are displayed in Figure 1 and Figure 2 respectively.

In this setup, memory errors arise around n^2 or $n_1n_2 = 10^9$ for the dense implementation. At this point, the Lanczos method is already between one and two orders of magnitude slower than Halko's. On the other hand, the "sparse + low-rank trick" enables to handle graphs with more than 3 million nodes and 234 million edges in approximately five minutes. Furthermore, note that the running time for the implicit dense matrix is quite close to the running time for its sparse component. This holds for a fixed k and varying graphs as well as for fixed graphs and varying k.

In the end, we have demonstrated that, without any specific hardware, spectral methods can easily scale to handle graphs at

[3]https://github.com/nathandelara/Matrix-Factorization

undirected			
code	name	n	m
MK	Kangaroo	2.10^1	2.10^2
JZ	Jazz musicians	2.10^2	5.10^3
Shf	Hamsterster	2.10^3	2.10^4
RC	Reactome	6.10^3	3.10^5
GW	Gowalla	2.10^5	2.10^6
SK	Skitter	2.10^6	2.10^7
OR	Orkut	3.10^6	2.10^8
bipartite			
code	name	n_1, n_2	m
SW	Southern women 1	2.10^1, 2.10^1	9.10^1
UL	Unicode languages	2.10^2, 6.10^1	1.10^3
SX	Sexual escorts	1.10^4, 6.10^3	4.10^4
Mut	MovieLens user–tag	4.10^3, 2.10^4	4.10^4
R2	Reuters-21578	2.10^4, 4.10^4	1.10^6
M3	MovieLens 10M	7.10^4, 1.10^4	1.10^7
RE	Reuters	8.10^5, 3.10^5	6.10^7

Table 2: Metadata of the datasets.

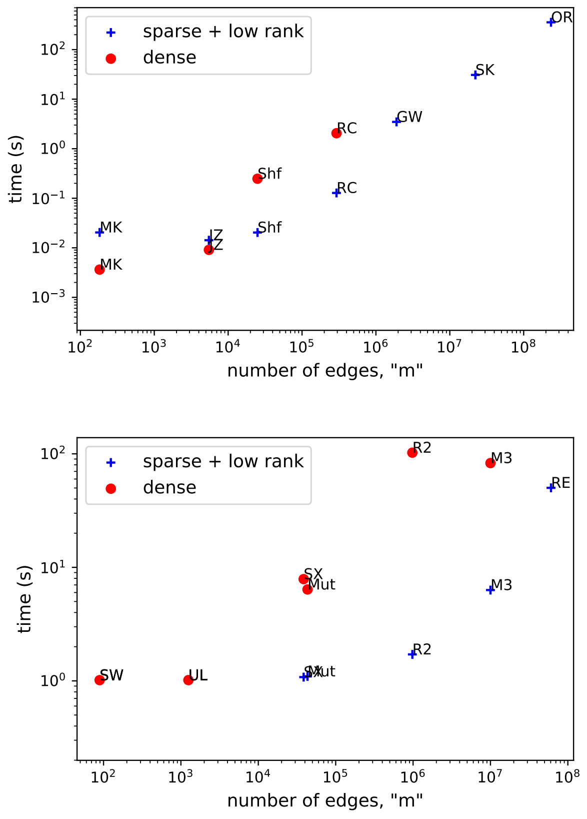

Figure 1: Computation times in seconds for partial decomposition of matrices. Top: undirected graphs. Bottom: bipartite graphs.

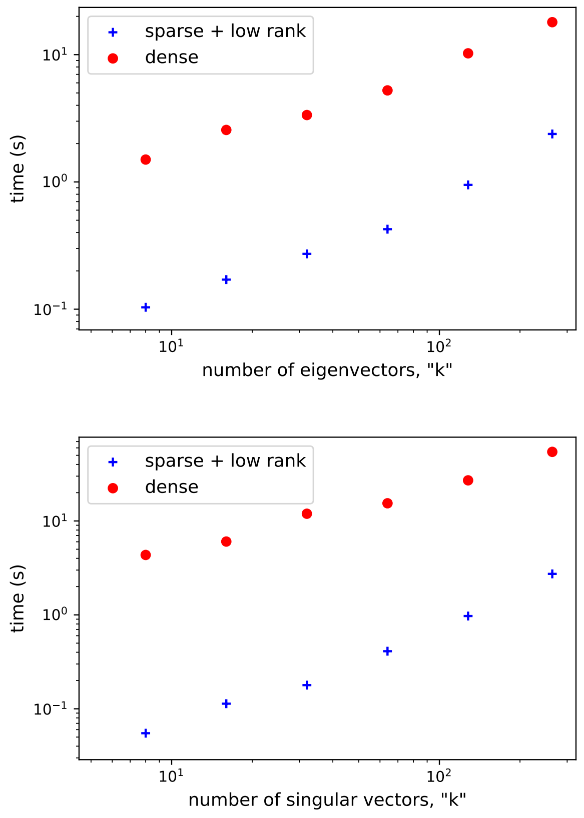

Figure 2: Computation times in seconds for different values of k. Top: Reactome. Bottom MovieLens user–tag.

least as big as the ones used to benchmark most iterative methods. A Python module to perform such factorizations is implemented in Scikit-network[4]. We hope that this will help produce richer benchmarks for graph algorithms.

REFERENCES

[1] Mikhail Belkin and Partha Niyogi. 2002. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15 (2002), 1373–1396. https://doi.org/10.1162/089976603321780317

[2] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (Oct. 2008), P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008

[3] Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL* (2009), 31–40.

[4] Nathan de Lara and Edouard Pineau. 2018. A Simple Baseline Algorithm for Graph Classification. *arXiv:1810.09155 [cs, stat]* (Oct. 2018). http://arxiv.org/abs/1810.09155 arXiv: 1810.09155.

[5] Chris H. Q. Ding, Xiaofeng He, and Hongyuan Zha. 2001. A Spectral Method to Separate Disconnected and Nearly-disconnected Web Graph Components. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. ACM, New York, NY, USA, 275–280. https://doi.org/10.1145/502512.502551 event-place: San Francisco, California.

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[4]https://github.com/sknetwork-team/scikit-network

[7] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2009. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *arXiv:0909.4061 [math]* (Sept. 2009). http://arxiv.org/abs/0909.4061 arXiv: 0909.4061.

[8] T. Haveliwala. 1999. *Efficient Computation of PageRank.* Technical Report 1999-31. Stanford InfoLab. http://ilpubs.stanford.edu:8090/386/

[9] Tsuyoshi Idé and Hisashi Kashima. 2004. Eigenspace-based Anomaly Detection in Computer Systems. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*. ACM, New York, NY, USA, 440–449. https://doi.org/10.1145/1014052.1014102 event-place: Seattle, WA, USA.

[10] Eric Jones, Travis Oliphant, and Pearu Peterson. 2001. {SciPy}: Open source scientific tools for {Python}. (2001). http://www.scipy.org

[11] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[12] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13 Companion)*. ACM, New York, NY, USA, 1343–1350. https://doi.org/10.1145/2487788.2488173 event-place: Rio de Janeiro, Brazil.

[13] Cornelius Lanczos. 1950. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators.* United States Governm. Press Office Los Angeles, CA.

[14] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[16] M. E. J. Newman. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103, 23 (June 2006), 8577–8582. https://doi.org/10.1073/pnas.0601602103

[17] M. E. J. Newman. 2013. Spectral methods for community detection and graph partitioning. *Physical Review E* 88, 4 (Oct. 2013), 042822. https://doi.org/10.1103/PhysRevE.88.042822

[18] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab. http://ilpubs.stanford.edu:8090/422/ Previous number = SIDL-WP-1999-0120.

[19] Nicola Perra and Santo Fortunato. 2008. Spectral centrality measures in complex networks. *Physical Review E* 78, 3 (Sept. 2008), 036107. https://doi.org/10.1103/PhysRevE.78.036107

[20] Marco Saerens, Francois Fouss, Luh Yen, and Pierre Dupont. 2004. The Principal Components Analysis of a Graph, and Its Relationships to Spectral Clustering. In *Machine Learning: ECML 2004 (Lecture Notes in Computer Science)*, Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi (Eds.). Springer Berlin Heidelberg, 371–383.

[21] Ulrike von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17, 4 (Dec. 2007), 395–416. https://doi.org/10.1007/s11222-007-9033-z

[22] Yilin Zhang and Karl Rohe. 2018. Understanding Regularized Spectral Clustering via Graph Conductance. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 10631–10640. http://papers.nips.cc/paper/8262-understanding-regularized-spectral-clustering-via-graph-conductance.pdf