

Exploiting Computation-Friendly Graph Compression Methods for Adjacency-Matrix Multiplication

Alexandre P Francisco*, Travis Gagie†, Susana Ladra‡, and Gonzalo Navarro§

*INESC-ID / IST
Universidade de Lisboa
Portugal
aplf@ist.utl.pt

†EIT, Diego Portales University
and CeBiB
Chile
travis.gagie@gmail.com

‡Facultade de Informática / CITIC
Universidade da Coruña
Spain
sladra@udc.es

§Department of Computer Science
University of Chile
Chile
gnavarro@dcc.uchile.cl

Abstract

Computing the product of the (binary) adjacency matrix of a large graph with a real-valued vector is an important operation that lies at the heart of various graph analysis tasks, such as computing PageRank. In this paper we show that some well-known Web and social graph compression formats are *computation-friendly*, in the sense that they allow boosting the computation. In particular, we show that the format of Boldi and Vigna allows computing the product in time proportional to the compressed graph size. Our experimental results show speedups of at least 2 on graphs that were compressed at least 5 times with respect to the original. We show that other successful graph compression formats enjoy this property as well.

Introduction

Let \mathbf{A} be an $n \times n$ binary matrix and $\mathbf{x} \in \mathbb{R}^n$ a vector. Matrix vector multiplication, either $\mathbf{x} \cdot \mathbf{A}$ or $\mathbf{A} \cdot \mathbf{x}^\top$, is not only a fundamental operation in mathematics, but also a key operation in various graph-analysis tasks, when \mathbf{A} is their adjacency matrix. A well-known example, which we use as a motivation, is the computation of PageRank on large Web graphs. PageRank is a particular case of many network centrality measures that can be approximated through the power method [1]. Most real networks, and in particular Web and social graphs, have very sparse adjacency matrices [2]. While it is straightforward to compute a matrix-vector product in time proportional to the nonzero entries of \mathbf{A} , the most successful Web and social graph compression methods exploit other properties that allow them to compress the graphs well beyond what is possible by their mere sparsity. It is therefore natural to ask whether those more powerful compression formats allow us, as sparsity does, to compute the product *in time proportional to the size of the compressed representation*. This is an instance of *computation-friendly compression*, which seeks for compression formats that not only reduce the size of the representation of objects, but also speeds up computations on them by directly operating on the compressed representations. Other examples

of computation-friendly compression are pattern matching in compressed strings [3], computation of edit distance between compressible strings [4], speedups for multiplying sequences of matrices and the Viterbi algorithm [5], building small and shallow circuits [6], among other tasks [7].

In this paper we exploit compressed representations of Web and social networks and show that matrix-vector products can be carried out much faster than just operating on all the nonzero entries of the matrix. Although our approach can be extended to other compressed representations of graphs and binary matrices, we mostly consider the representation proposed by Boldi and Vigna [8]. The relevant observation for us is that adjacency lists, i.e., rows in \mathbf{A} , are compressed differentially with respect to other similar lists, and thus one can reuse and “correct” the result of the multiplication of a previous similar row with \mathbf{x}^\top .

We describe previous work in the next section. The following sections describe PageRank and the compression format of Boldi and Vigna. We then describe how we exploit that compression format to speed up matrix multiplication. The following section contains experimental results, and we conclude with a discussion of other compression formats that favor matrix multiplications, and future work directions.

Previous Work

Matrix multiplication is a fundamental problem in computer science; see, e.g., [9] for a recent survey of results. Computation-friendly matrix compression has been already considered by others, even if indirectly. Karande et al. [10] addressed it by exploiting a structural compression scheme, namely by introducing virtual nodes. Although their results were similar to the ones presented in this paper, their approach was more complex and it could not be used directly, requiring the correction of computation results. On the other hand, contrary to their belief, we show in this paper that representational compression schemes do not always require the same amount of computation, providing a much simpler approach that can be used directly without requiring corrections.

Another interesting approach was proposed by Nishino et al. [11]. Although they did not exploit compression in the same way we do, they observed that intermediate computational results for the matrix multiplication of equivalent partial rows of a matrix are the same. They used then an adjacency forest where rows are represented by sharing common suffixes. We should note that the authors consider general real matrices, and not only Boolean matrices as we do. Nevertheless they presented results for computing the PageRank over adjacency matrices as we do, achieving similar results. Their approach implied preprocessing the graph, however, while we start from an already compressed graph. An interesting question is how their approach could be exploited on top of k^2 -trees [12].

The question addressed here can also be of interest for the problem of Online Matrix-Vector (OMV) multiplication. Given a stream of binary vectors, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots$, the results of matrix-vector multiplications $\mathbf{x}_i \cdot \mathbf{A}$ can be computed faster than computing them independently, with most approaches making use of previous computations $\mathbf{x}_j \cdot \mathbf{A}$, for $j < i$, to speed up the computation of each new product $\mathbf{x}_i \cdot \mathbf{A}$ [13, 14].

Nevertheless, none of those approaches preprocess matrix \mathbf{A} to exploit its redundancies. Hence, by exploiting a suitable succinct representation of \mathbf{A} as we do here, an improvement for OMV can be easily obtained, with computational time depending on the length of the succinct representation of \mathbf{A} instead.

PageRank

Given $G = (V, E)$ a graph with $n = |V|$ vertices and $m = |E|$ edges, let \mathbf{A} be its adjacency matrix; $A_{uv} = 1$ if $(u, v) \in E$, and $A_{uv} = 0$ otherwise. The normalized adjacency matrix of G is the matrix $\mathbf{M} = \mathbf{D}^{-1} \cdot \mathbf{A}$, where \mathbf{D} is an $n \times n$ diagonal matrix with D_{uu} the degree d_u of $u \in V$, i.e., $D_{uu} = d_u = \sum_v A_{uv}$. Note that \mathbf{M} is the standard random walk matrix, where a random walker at vertex u jumps to a neighbor v of u with probability $1/d_u$. Moreover the k -power of \mathbf{M} , \mathbf{M}^k , is the random walk matrix after k steps, i.e., M_{uv}^k is the probability of the random walker being at vertex v after k jumps, having started at vertex u . PageRank is a typical random walk on G with transition matrix \mathbf{M} . Given a constant $0 < \alpha < 1$ and a probability vector \mathbf{p}_0 , the PageRank vector \mathbf{p}_α is given by the following recurrence [15]:

$$\mathbf{p}_\alpha = \alpha \mathbf{p}_0 + (1 - \alpha) \mathbf{p}_\alpha \cdot \mathbf{M}.$$

The parameter α is called the teleport probability or jumping factor, and \mathbf{p}_0 is the starting vector. In the original PageRank [16], the starting vector \mathbf{p}_0 is the uniform distribution over the vertices of G , i.e., $\mathbf{p}_0 = \mathbf{1}/n$. When \mathbf{p}_0 is not the stationary distribution, \mathbf{p}_α is called a personalized PageRank. Intuitively, \mathbf{p}_α is the probability of a lazy Web visitor to be at each page assuming that he/she surfs the Web by either randomly starting at a new page or jumping through a link from the current page. The parameter α ensures that such a surfer does not get stuck at a dead end. PageRank can be approximated iteratively through the power iteration method by iterating, for $t \geq 1$:

$$\mathbf{p}_t = \alpha \mathbf{p}_0 + (1 - \alpha) \mathbf{p}_{t-1} \cdot \mathbf{M}. \quad (1)$$

We show how to speed up these matrix-vector multiplications when the adjacency matrix \mathbf{A} is compressible.

Our Approach

Our main idea is to exploit the copy-property of adjacency lists observed in some graphs, such as Web graphs [8]. The adjacency lists of neighbor vertices tend to be very similar and, hence, the rows in the adjacency matrix are also very similar. Moreover these networks reveal also strong clustering effects, with local groups of vertices being strongly connected and/or sharing many neighbors. The copy-property effect can then be further amplified through clustering and suitable vertex reordering, an important step for achieving better graph compression ratios [17]. Most compressed representations for sparse graphs rely on these properties [18–20]. In this paper, we consider the WebGraph framework, a suite of codes, algorithms and tools that aims at making it easy to manipulate large Web graphs [8]. Among several compression techniques used in Webgraph, our approach makes use of list referencing.

Let \mathbf{A} be an $n \times n$ binary sparse matrix,

$$\mathbf{A} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}$$

where $\mathbf{v}_i \in \{0,1\}^n$ is the i -th row, for $i = 1, \dots, n$. Let $\mathbf{r} \in \{0, 1, \dots, n\}^n$ be a referencing vector such that, for $i \in \{1, \dots, n\}$, $r_i < i$ and \mathbf{v}_{r_i} is some previous row used for representing \mathbf{v}_i . Let also $\mathbf{v}_0 = \mathbf{0}$ and $r_1 = 0$. The reference r_i is found in the WebGraph framework within a given window W , i.e., $r_i \in \{\max(1, i - W), \dots, i\}$, and it is optimized to reduce the length of the representation of \mathbf{v}_i . The line \mathbf{v}_i is then represented by adding missing entries and marking spurious ones, with respect to \mathbf{v}_{r_i} , and encoded using several techniques, such as differential compression and codes for natural numbers [8, 21].

Proposition 1. *Given an $n \times n$ matrix \mathbf{A} , $\mathbf{x} \in \mathbb{R}^n$, and a referencing vector \mathbf{r} for \mathbf{A} , let \mathbf{A}' and \mathbf{w} be defined as follows:*

$$\mathbf{A}' = \begin{bmatrix} \mathbf{v}_1 - \mathbf{v}_{r_1} \\ \vdots \\ \mathbf{v}_n - \mathbf{v}_{r_n} \end{bmatrix}$$

$$w_i = \mathbf{v}_{r_i} \cdot \mathbf{x}^\top$$

Then we have that:

$$\mathbf{A} \cdot \mathbf{x}^\top = \mathbf{A}' \cdot \mathbf{x}^\top + \mathbf{w}^\top$$

Proof. By definition,

$$\mathbf{A}' \cdot \mathbf{x}^\top + \mathbf{w}^\top = \begin{bmatrix} \mathbf{v}_1 \cdot \mathbf{x}^\top - \mathbf{v}_{r_1} \cdot \mathbf{x}^\top \\ \vdots \\ \mathbf{v}_n \cdot \mathbf{x}^\top - \mathbf{v}_{r_n} \cdot \mathbf{x}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{v}_{r_1} \cdot \mathbf{x}^\top \\ \vdots \\ \mathbf{v}_{r_n} \cdot \mathbf{x}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \cdot \mathbf{x}^\top \\ \vdots \\ \mathbf{v}_n \cdot \mathbf{x}^\top \end{bmatrix} = \mathbf{A} \cdot \mathbf{x}^\top$$

■

Let us compute $\mathbf{y}^\top = \mathbf{A} \cdot \mathbf{x}^\top$ by iterating over $i = 1, \dots, n$. Then \mathbf{w} can be incrementally computed because $r_i < i$ and $w_i = y_{r_i}$, ensuring that w_i is already computed when required to compute y_i . Given inputs \mathbf{A}' , \mathbf{r} and \mathbf{x} , the algorithm to compute \mathbf{y} is as follows:

1. Set $\mathbf{y} = \mathbf{0}$ and $y_0 = 0$.
2. For $i = 1, \dots, n$, set $y_i = y_{r_i} + \sum_j A'_{ij} x_j$.
3. Return \mathbf{y} .

Note that the number of operations to obtain $\mathbf{y}^\top = \mathbf{A} \cdot \mathbf{x}^\top$ is proportional to the number of nonzeros in \mathbf{A}' , that is, to the compressed representation size. Depending on the properties of \mathbf{A} discussed before, this number may be much smaller than the number of nonzeros in \mathbf{A} . We present in the next section experimental results for Web graphs, where we indeed obtain considerable speedups the computation of PageRank.

Table 1: Datasets used in the experimental evaluation, where n is the number of vertices, m is the number of edges (i.e., nonzeros in \mathbf{A}), m' is the number of nonzeros in \mathbf{A}' , t is the average time in seconds to compute a matrix-vector product with \mathbf{A} , t' is the average time in seconds to compute a matrix-vector product with \mathbf{A}' , and S is the speedup observed in the computation of PageRank. The first five datasets are Web crawls and the remaining ones are social networks. All datasets are available at <http://law.di.unimi.it/datasets.php>. Times and speedups were only computed for web graphs.

Graph	n	m	m'	m/m'	t	t'	S
eu-2015-hc	1.07×10^9	9.17×10^{10}	1.11×10^{10}	8.26	3244.0	1099.0	2.95
eu-2015-host-hc	1.13×10^7	3.87×10^8	1.10×10^8	3.52	11.55	8.15	1.42
gsh-2015-hc	9.88×10^8	3.39×10^{10}	7.08×10^9	4.78	1803.6	953.4	1.89
it-2004-hc	4.13×10^7	1.15×10^9	2.27×10^8	5.08	24.65	12.0	2.05
uk-2014-hc	7.88×10^8	4.76×10^{10}	6.26×10^9	7.58	2034.0	665.8	3.05
twitter-2010-hc	4.17×10^7	1.47×10^9	1.44×10^9	1.02	–	–	–
amazon-2008-hc	7.35×10^5	$5,16 \times 10^6$	4.48×10^6	1.15	–	–	–
enwiki-2013	4.21×10^6	1.01×10^8	9.62×10^7	1.05	–	–	–
wordassoc.-2011	1.06×10^4	7.22×10^4	7.15×10^4	1.01	–	–	–

Experimental Evaluation

We computed the number of nonzeros m' in \mathbf{A}' for the adjacency matrix \mathbf{A} of several graphs available at <http://law.di.unimi.it/datasets.php> [8, 17, 22]. Whenever $|\mathbf{v}_i - \mathbf{v}_{r_i}| \geq |\mathbf{v}_i|$, we kept \mathbf{v}_i as the row in \mathbf{A}' , since it resulted in fewer nonzeros. Results are presented in Table 1, including the number of vertices n and the number of edges m , for each graph. Both \mathbf{A}' and \mathbf{r} were obtained directly from the Web-graph representation using high compression, which uses stronger referencing among adjacencies and thus favors our approach.

As expected, our approach works extremely well for Web graphs, with the number of nonzeros in \mathbf{A}' being less than 20% for page graphs and less than 30% for host graphs. Note that Web graphs are known to verify the copy-property among adjacencies. Other networks we tested, instead, seem not to verify this property in the same degree, and therefore our approach is not beneficial. This was expected, as social networks are not as compressible as Web graphs [23]. There may exist, however, other representations for these networks that may benefit from other compression approaches (see the next section).

We implemented PageRank using the algorithm above to compute matrix vector products. Since Eq. (1) uses left products and our representation is row-oriented, we use the transposed adjacency matrix and right products. The implementation is in Java and based on the Webgraph representation, where \mathbf{A}' is represented as two graphs: a positive one for edges with weight 1, and a negative one for edges with weight -1 . All tests were conducted on a machine running Linux, with an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (8 cores, cache 32KB/4096KB) and with 32GB of RAM. Java code was compiled and executed with OpenJDK 1.8.0_131.

We ran 10 iterations for the Web graphs in Table 1, starting with the uniform distribution. Let us consider the graphs eu-2015-host-hc and it-2004-hc. Our

implementation took 81.5 and 120.0 seconds for `eu-2015-host-hc` and `it-2004-hc`, respectively. An equivalent implementation of PageRank, using the adjacency matrix \mathbf{A} instead of \mathbf{A}' , represented with WebGraph, took 115.5 seconds and 246.5 for `eu-2015-host-hc` and `it-2004-hc`, respectively. Hence, we achieved speedups of 1.42 and 2.05, respectively, as presented in Table 1. Observed speedups are lower than what we would expect given that \mathbf{A}' has 3.52 times fewer nonzeros than \mathbf{A} for `eu-2015-host-hc`, and 5.08 times fewer for `it-2004-hc`. After profiling we could observe that, although \mathbf{A}' had much fewer nonzeros than \mathbf{A} , the nonzeros in \mathbf{A}' are more dispersed than those in \mathbf{A} , with \mathbf{A} benefiting from contiguous memory accesses. The speedups are nevertheless significant, namely when we are dealing with larger graphs like `eu-2015-hc`. Our implementation took 1h30m for this graph, about 3 times less than the equivalent implementation using matrix \mathbf{A} instead of matrix \mathbf{A}' .

We replicated the experiments with code written in C using a plain representation for sparse matrices, for both \mathbf{A} and \mathbf{A}' . The operations became 10 times faster, but the difference between operating with both \mathbf{A} and \mathbf{A}' remained similar.

Final Remarks

We have shown that the adjacency matrix compression scheme of Boldi and Vigna [8] allows for computing matrix-vector products in time proportional to the *compressed* matrix size. Therefore, compression not only saves space but also speeds up an operation that is key for graph analysis tasks.

This is not a property unique to that compression format. Another suitable format is the biclique extraction method of Hernández and Navarro [24]. They decompose the edges of G into a number of bicliques (S_r, C_r) , so that every node from S_r points to every node from C_r , plus a residual set of edges. The $|S_r| \cdot |C_r|$ edges of each biclique are represented in $|S_r| + |C_r|$ words, by just listing both sets. This format is shown to be competitive to compress both Web and social graphs. In order to compute $\mathbf{A} \cdot \mathbf{x}^\top$, we compute for each biclique r the value $c_r = \sum_{j \in C_r} x_j$. We then initialize n counters $y_j = 0$ and, for each biclique r and each $i \in S_r$, we add c_r to y_i . Finally, for each residual edge $A_{ij} = 1$, we add x_j to y_i . The final answer is the vector \mathbf{y}^\top , which is obtained in time proportional to the size of the compressed matrix.

We plan to study the practical speedup obtained with this compression format. We also plan to improve the results on Boldi and Vigna’s algorithm by varying the size of the window and splitting the input matrix into submatrices of consecutive columns so matches are more flexible and need not span entire rows. We will also consider other formats where it is less clear how to translate the reduction in space into a reduction in computation time [18–20, 24], and study which other relevant matrix operations can be boosted by which compression formats.

Acknowledgments

This research has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie [grant agreement No 690941], namely while the first author was visiting the University of Chile, and while

the second author was affiliated with the University of Helsinki and visiting the University of A Coruña. The first author was funded by Fundação para a Ciência e a Tecnologia (FCT) [grant number UID/CEC/50021/2013]; the second author was funded by Academy of Finland [grant number 268324] and Fondecyt [grant number 1171058]; the third author was funded by Ministerio de Economía y Competitividad (PGE and FEDER) [grant number TIN2016-77158-C4-3-R] and Xunta de Galicia (co-founded with FEDER) [grant numbers ED431C 2017/58; ED431G/01]; and the fourth author was funded by Millennium Nucleus Information and Coordination in Networks [grant number ICM/FIC RC130003].

- [1] Newman, M.: Networks: An introduction. Oxford University Press (2010).
- [2] Chung, F.R. and Lu, L.: Complex graphs and networks. Number 107 in CBMS – Regional Conference Series in Mathematics. American Mathematical Society (2006).
- [3] Travis Gagie, Pawel Gawrychowski, and Simon J. Puglisi. Approximate pattern matching in LZ77-compressed texts. *Journal of Discrete Algorithms*, 32:64–68, 2015.
- [4] Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013.
- [5] Yury Lifshits, Shay Mozes, Oren Weimann, Michal Ziv-Ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. *Algorithmica*, 54(3):379–399, 2009.
- [6] Moses Ganardi, Danny Hucke, Artur Jez, Markus Lohrey, Eric Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86: 136–158, 2017.
- [7] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2): 241–299, 2012.
- [8] Boldi, P. and Vigna, S.: The WebGraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, 595–602, ACM (2004).
- [9] Josh Alman and Virginia Vassilevska Williams. Further limitations of the known approaches for matrix multiplication. *Proceedings of the 2018 Conference on Innovations in Theoretical Computer Science*, 25:1–25:15 (2018).
- [10] Karande, C., Chellapilla, K. and Andersen, R.: Speeding Up Algorithms on Compressed Web Graphs. *Internet Mathematics*, 6(3):373–398 (2009).
- [11] Nishino, M., Yasuda, N., Minato, S.-i. and Nagata, M.: Accelerating Graph Adjacency Matrix Multiplications with Adjacency Forest. *Proceedings of the 2014 SIAM International Conference on Data Mining*, 1073–1081 (2014).
- [12] Brisaboa, N., Ladra, S. and Navarro, G.: Compact Representation of Web Graphs with Extended Functionality. *Information Systems*, 39:152–174 (2014).
- [13] Henzinger, M., Krinninger, S., Nanongkai, D. and Saranurak, T.: Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. *Proceedings of the forty-seventh annual ACM Symposium on Theory of Computing*, 21–30 (2015).
- [14] Larsen, K.-G. and Williams, R.: Faster Online Matrix-Vector Multiplication. *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2182–2189 (2017).
- [15] Chung, F.: The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740 (2007).

- [16] Page, L., Brin, S., Motwani, R. and Winograd, T.: The PageRank citation ranking: Bringing order to the web. Stanford InfoLab (1999).
- [17] Boldi, P., Rosa, M., Santini, M. and Vigna, S.: Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th International Conference on World Wide Web*, 587–596, ACM (2011).
- [18] Brisaboa, N., Ladra, S. and Navarro, G.: Compact Representation of Web Graphs with Extended Functionality. *Information Systems*, 39(1):152–174 (2014).
- [19] Grabowski, S. and Bieniecki, W.: Merging adjacency lists for efficient Web graph compression. In *Man-Machine Interactions 2 AISC 103*, 385–392 (2011).
- [20] Claude, F. and Navarro, G.: Fast and Compact Web Graph Representations. *ACM Transactions on the Web (TWEB)*, 4(4):16 (2010).
- [21] Boldi, P. and Vigna, S.: Codes for the World-Wide Web. *Internet Mathematics*, 2(4):407–429 (2005).
- [22] Boldi, P., Marino, A., Santini, M. and Vigna, S.: BUbiNG: Massive Crawling for the Masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*, 227–228 (2014).
- [23] Chierichetti F., Kumar R., Lattanzi S., Mitzenmacher M., Panconesi A. and Raghavan P.: On compressing social networks. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 219–228 (2009).
- [24] Hernández, C. and Navarro, G.: Compressed Representations for Web and Social Graphs. *Knowledge and Information Systems*, 40(2):279–313 (2014).