Graph Compression for Adjacency-Matrix Multiplication

Alexandre P. Francisco[1*], Travis Gagie[2], Dominik Köppl[3*], Susana Ladra[4] and Gonzalo Navarro[5,6]

[1]INESC-ID / IST, Universidade de Lisboa, Portugal.
[2]Faculty of Computer Science, Dalhousie University, Canada.
[3]M & D Data Science Center, Tokyo Medical and Dental University, Japan.
[4]CITIC, Universidade da Coruña, Spain.
[5]Millennium Institute for Foundational Research on Data, Chile.
[6]Department of Computer Science, University of Chile, Chile.

*Corresponding author(s). E-mail(s): aplf@ist.utl.pt; koeppl.dsc@tmd.ac.jp;
Contributing authors: travis.gagie@dal.ca; susana.ladra@udc.es; 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-graph and social graph compression formats are *computation-friendly*, in the sense that they allow boosting the computation. We focus on the compressed representations of (a) Boldi and Vigna and (b) Hernández and Navarro, and show that the product computation can be conducted 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.

Keywords: matrix multiplication, webgraph, bicliques, data compression

1 Introduction

Let $\mathbf{A} \in \{0,1\}^{n \times n}$ be an $n \times n$ binary matrix and $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ be a vector. Matrix-vector multiplication, either $\vec{x} \cdot \mathbf{A}$ or $\mathbf{A} \cdot \vec{x}^\top$, 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, Chapter 11.1]. Most real networks, and in particular webgraphs 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 applies compression formats that not only reduce the size of the representation of objects, but also speed up computations on them by directly operating on the compressed representations. Elgohary et al. [3] addressed this problem for structured matrices commonly found in machine learning for matrix-vector multiplication. However, Abboud et et. [4] have proven that with sophisticated compression techniques it is difficult or even impossible to compute basic linear-algebra operations like matrix-vector multiplication in subquadratic time. Additionally, Chakraborty et al. [5] showed that any data structure storing r bits with $n < r < n^2$ must have a query time t satisfying $tr \in \Omega(n^3 \text{polylog}(n))$. Other examples of computation-friendly compression are pattern matching in compressed strings [6], computation of edit distance between compressible strings [7], speedups for multiplying sequences of matrices and the Viterbi algorithm [8], representing bipartite graphs [9], building small and shallow circuits [10], among other tasks [11].

1.1 Our Contribution

In this article we exploit compressed representations of webgraphs 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 two representations: one proposed by Boldi and Vigna [12], and the other proposed by Hernández and Navarro [13]. For the former, the key 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 $\vec{x}^\top$. The latter representation works by extracting regular substructures in the matrix, on which the matrix multiplication becomes particularly simple.

A preliminary version of this article appeared at the Data Compression Conference 2018 [14]. The conference version did not consider Hernández and Navarro's representations, and we also experiment with slicing the matrix vertically into sub-matrices.

1.2 Structure of this Article

We describe previous work in the next section (Sect. 1.3). The following sections describe PageRank and the compression format of Boldi and Vigna (Sect. 2). We then describe how we exploit that compression format to speed up matrix multiplication (Sect. 3), and a vertical split of the matrix used for PageRank to boost up the compression in Sect. 4. Section 5 contains experimental results for this compression format. Subsequently, in Sect. 6, we show how to use the compression format of Hernández and Navarro [13]. We conclude this article with directions for future work. Compared to the conference version of this paper [14], we added Sect. 4, a more thorough evaluation including variable window sizes, and the second compression format of Hernández and Navarro in Sect. 6.

1.3 Previous Work

Matrix multiplication is a fundamental problem in computer science; see, e.g., a recent survey of results [15]. Computation-friendly matrix compression has been already considered by others, even if indirectly. Karande et al. [16] 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. [17]. 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. Their approach is to use an adjacency forest representing the rows of the matrix; this forest achieves compression by compacting common suffixes of the rows. 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 [18].

The question addressed here can also be of interest for the problem of Online Matrix-Vector (OMV) multiplication. Given a stream of binary vectors, $\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots$, the results of matrix-vector multiplications $\vec{x}_i \cdot \mathbf{A}$ can be computed faster than computing them independently, with most approaches making use of previous computations $\vec{x}_j \cdot \mathbf{A}$, for $j < i$, to speed up the computation of each new product $\vec{x}_i \cdot \mathbf{A}$ [19, 20]. Nevertheless, none of those approaches preprocess matrix $\mathbf{A}$ to exploit its redundancies. Hence, by exploiting a suitable compressed 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 compressed representation of $\mathbf{A}$ instead.

2 PageRank

Given $G = (V, E)$, a directed 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. We assume that for each vertex u, there is at least one vertex v with $A_{uv} = 1$. 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., $D_{uu} = d_u = \sum_v A_{uv} \geq 1$. 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^k_{uv} 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 p_0, the PageRank vector $\vec{p}_\alpha$ is given by the following recurrence [21]:

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

The parameter α is called the teleport probability or jumping factor, and $\vec{p}_0$ is the starting vector. In the original PageRank [22], the starting vector $\vec{p}_0$ is the uniform distribution over the vertices of G, i.e., $\vec{p}_0 = \vec{1}/n$. When $\vec{p}_0$ is not the stationary distribution, $\vec{p}_\alpha$ is called a personalized PageRank. Intuitively, $\vec{p}_\alpha$ 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 $t \geq 1$:

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

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

3 Computation on the WebGraph Format

Our main idea is to exploit the copy-property of adjacency lists observed in some graphs, such as Web graphs [12]. 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 [23]. Most compressed representations for sparse graphs rely on these properties [18, 24, 25]. 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 [12]. 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,

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

where $\vec{v}_i \in \{0,1\}^n$ is the i-th row, for $i = 1, \ldots, n$. Let $\vec{r} \in \{0, 1, \ldots, n\}^n$ be a referencing vector such that, for $i \in \{1, \ldots, n\}$, $r_i < i$ and $\vec{v}_{r_i}$ is some previous row used for representing $\vec{v}_i$. Let also $\vec{v}_0 = \vec{0}$ and $r_1 = 0$. The reference r_i is found in the WebGraph framework within a given window size W, i.e., $r_i \in \{\max(1, i - W), \ldots, i\}$, and it is optimized to reduce the length of the representation of $\vec{v}_i$. The line $\vec{v}_i$ is then represented by adding missing entries and marking spurious ones, with respect to $\vec{v}_{r_i}$, and encoded using several techniques, such as differential compression and codes for natural numbers [12, 26].

Proposition 1 *Given an $n \times n$ matrix $\mathbf{A}$, $\vec{x} \in \mathbb{R}^n$, and a referencing vector $\vec{r}$ for $\mathbf{A}$, let*

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

Further let $\vec{w}$ be the vector with $w_i := \vec{v}_{r_i} \cdot \vec{x}^\top$. Then we have that:

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

Proof By definition,

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

□

Let us compute $\vec{y}^\top = \mathbf{A} \cdot \vec{x}^\top$ by iterating over $i = 1, \ldots, n$. Then $\vec{w}$ 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 $\mathbf{A}'$, $\vec{r}$ and $\vec{x}$, the algorithm to compute $\vec{y}$ is as follows:

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

Note that the number of operations to obtain $\vec{y}^\top = \mathbf{A} \cdot \vec{x}^\top$ 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 in the computation of PageRank.

4 Vertically Slicing into Sub-Matrices

The quality of our approach hinges on the quality of $\vec{r}$. The wider the matrix $\mathbf{A}$ is, the more difficult it can become to find a previous row adequately matching the entries. It therefore could make sense to vertically split $\mathbf{A}$ into submatrices $\mathbf{A}_1, \ldots, \mathbf{A}_\tau$, each with n rows and $\Theta(n/\tau)$ columns, in the hope that applying the techniques of Section 3 to every submatrix gives a better chances of finding good references, and therefore eases the chances of obtaining a representation that is more compact than using the technique on the entire matrix $\mathbf{A}$. This somewhat reflects real-world examples where a matrix does not usually contain complete row repetitions, but rather clustered repetitions of a certain length, which are hopefully of length $\Omega(n/\tau)$. To see that we still can compute the matrix-vector multiplication with the submatrices efficiently, take a vector $\vec{x} \in \mathbb{R}^n$. We can compute the product $\vec{y} = \mathbf{A} \cdot \vec{x}^\top$ with $\vec{y}_i = \mathbf{A}_i \cdot \vec{x}_i^\top$ for $i \in [1..\tau]$ and then $\vec{y} = \sum_{i=1}^{\tau} y_i$ summing up all computed products. Setting $\tau = 1$ disables this technique, i.e., just using the technique of Section 3. Setting $\tau = n$ gives us the school-book matrix-vector multiplication algorithm.

5 Experimental Evaluation

We computed the number of nonzero entries m' in $\mathbf{A}'$ for the adjacency matrix $\mathbf{A}$ of several graphs available at `http://law.di.unimi.it/datasets.php` [12, 23, 27]. We show in Table 1 some characteristics of the used graphs, including the number of vertices n and the number of edges m, for each graph. We categorize our studied graphs into the following three classes:

page graph : each node represents a single web page, and each arc is a link between two pages;
host graph : each node is a (sub)domain, i.e., a host, and an arc between two hosts exists if one host has a page having a link to the page of the other host;

Table 1 Datasets used in the experimental evaluation, where n is the number of vertices and m is the number of edges (i.e., nonzeros in $\mathbf{A}$). All datasets are available at `http://law.di.unimi.it/datasets.php`.

Graph	type	n [M]	m [M]
`amazon-2008-hc`	social network	0.74	5.16
`enwiki-2021-hc`	social network	6.26	150.12
`eu-2015-hc`	page graph	1070.56	91792.26
`eu-2015-host-hc`	host graph	11.26	386.92
`gsh-2015-hc`	page graph	988.49	33877.40
`it-2004-hc`	page graph	41.29	1150.73
`twitter-2010-hc`	social network	41.65	1468.37
`uk-2014-hc`	page graph	787.80	47614.53

social network : each node represents an entity such as a person or user, and an arc represents a social relation.

We call page graphs and host graphs together web graphs (not to be confused with the WebGraph framework storing graphs in its WebGraph representation).

As a preprocessing step for our experiments, we extracted $\mathbf{A}'$ and $\vec{r}$ from the WebGraph representation of $\mathbf{A}$, and compressed $\mathbf{A}'$ as a WebGraph, before actually starting the computation of the PageRank algorithm. For each window size W, we first recompressed a graph with the selected W and compressed the references with Elias-γ via the parameter `-c REFERENCES_GAMMA`. We did so because the WebGraph representation achieves high compression, but is limited to storing adjacency matrices, which we needed for PageRank. Hence, this approach allowed us to do all of the computation in compressed space, which would not be possible for commodity computers to run in RAM if we had extracted $\mathbf{A}$ in its plain form. As an optimization, whenever $|\vec{v}_i - \vec{v}_{r_i}| \geq |\vec{v}_i|$, we kept $\vec{v}_i$ as the row in $\mathbf{A}'$. By doing so, we obtained fewer nonzero entries.

We implemented PageRank using the algorithm described in Sect. 2 computing 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 CPU i3-9100 (4 cores, cache 256 KB/6144 KB) and with 128 GB of RAM. Java code was compiled and executed with OpenJDK 11.0.9.1 and the parameters `-Xmx100G -Xss100M` to access 100 GB of RAM and keeping an execution stack of size at most 100 MB. We ran each PageRank computation for ten iterations, starting with the initial vector $\vec{p}_0$ representing the uniform distribution.

We have the benchmark results of the PageRank evaluation in the last columns of Table 2 for different window sizes. As expected, our approach works 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

Table 2 Evaluation of different window sizes W on the datasets of Table 1, where m' is the number of nonzero entries 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 t/t' is the speedup observed in the computation of PageRank.

Graph	W	m' [M]	m/m'	t' [s]	t [s]	t/t'
amazon-2008-hc	16	4.60	1.12	0.21	0.22	1.03
	32	4.60	1.12	0.22	0.22	1.03
	64	4.60	1.12	0.22	0.22	1.04
	128	4.60	1.12	0.21	0.22	1.02
	256	4.60	1.12	0.21	0.23	1.09
enwiki-2021-hc	16	146.23	1.03	5.00	5.17	1.03
	32	146.33	1.03	4.96	5.09	1.03
	64	146.22	1.03	5.01	5.15	1.03
	128	146.20	1.03	5.06	5.26	1.04
	256	145.98	1.03	4.93	5.26	1.07
eu-2015-host-hc	16	126.51	3.06	4.43	6.18	1.40
	32	120.78	3.20	4.30	6.10	1.42
	64	117.16	3.30	4.39	6.20	1.41
	128	115.89	3.34	4.32	6.30	1.46
	256	115.76	3.34	4.19	6.10	1.46
eu-2015-hc	16	16485.44	5.57	353.13	874.14	2.48
	32	15488.57	5.93	338.03	849.88	2.51
	64	15657.24	5.86	336.23	875.60	2.60
	128	16674.24	5.51	355.38	903.18	2.54
	256	18299.52	5.02	360.11	935.32	2.60
gsh-2015-hc	16	8912.54	3.80	249.04	419.86	1.69
it-2004-hc	16	283.83	4.05	7.04	12.89	1.83
	32	276.65	4.16	6.74	12.92	1.92
	64	281.34	4.09	6.73	12.90	1.92
	128	291.50	3.95	6.82	13.06	1.91
	256	306.03	3.76	6.86	13.12	1.91
twitter-2010-hc	16	1435.10	1.02	56.80	58.43	1.03
	32	1434.16	1.02	56.49	53.00	0.94
	64	1432.90	1.02	57.83	57.01	0.99
	128	1431.42	1.03	56.80	58.38	1.03
	256	1431.36	1.03	57.11	57.06	1.00
uk-2014-hc	16	8791.33	5.42	203.39	462.63	2.27
	32	8193.65	5.81	196.07	463.49	2.36
	64	8106.13	5.87	196.71	473.10	2.41
	128	8467.23	5.62	198.09	469.76	2.37

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 [28]. There may exist, however, other representations for these networks that may benefit from other compression approaches (see the next section). In general, large reductions in the numbers of

Table 3 Impact of vertically slicing $\mathbf{A}^\top$ into c submatrices $\mathbf{A}_1^\top, \cdots, \mathbf{A}_c^\top$ as described in Section 4. We parameterize each instance additionally with a window size $W \in \{16, 128\}$. For $i \in [1..c]$, m_i is the number of non-zero entries in $\mathbf{A}_i^\top$, and m'_i is the number of non-zero entries in $\mathbf{A'}_i^\top$. $\varnothing m_i$ and $\varnothing m'_i$ denote the average over all m_i and m'_i values, respectively. The last column denotes the fraction $\sum_{i=1}^{c} m'_i/m'$.

Graph	W	c	$\varnothing m_i$ [M]	$\varnothing m'_i$ [M]	$\sum_{i=1}^{c} m'_i$ [M]	Frac
amazon-2008-hc	16	32	0.16	0.14	4.38	0.95
	16	128	0.04	0.03	4.37	0.95
	128	32	0.16	0.14	4.37	0.95
	128	128	0.04	0.03	4.36	0.95
enwiki-2021-hc	16	32	4.69	4.49	143.75	0.98
	16	128	1.17	1.12	143.01	0.98
	128	32	4.69	4.48	143.28	0.98
	128	128	1.17	1.11	142.43	0.97
eu-2015-host-hc	16	32	12.09	3.81	122.04	0.96
	16	128	3.02	0.94	120.70	0.95
	128	32	12.09	3.43	109.80	0.95
	128	128	3.02	0.85	108.34	0.93
eu-2015-hc	16	32	2868.51	503.17	16101.48	0.98
	16	128	717.13	125.53	16067.36	0.97
gsh-2015-hc	16	32	1058.67	265.42	8493.31	0.95
	16	128	264.67	66.15	8466.91	0.95
it-2004-hc	16	32	35.96	8.75	280.12	0.99
	16	128	8.99	2.18	279.59	0.99
	128	32	35.96	8.87	283.87	0.97
	128	128	8.99	2.21	283.18	0.97
twitter-2010-hc	16	32	45.89	44.21	1414.84	0.98
	16	128	11.47	11.00	1408.06	0.99
	128	32	45.89	43.96	1406.74	0.98
	128	128	11.47	10.93	1399.00	0.98
uk-2014-hc	16	32	1487.95	272.21	8710.79	0.99
	16	128	371.99	67.99	8702.81	0.99
	128	32	1487.95	255.97	8191.02	0.97

nonzero entries tend to give bigger speed-ups, although the relationship may be complicated by algorithmic details, system characteristics and the interaction between the two.

Let us now consider the graphs `eu-2015-host-hc` and `it-2004-hc`. Observed speedups are lower than we would expect given that $\mathbf{A'}$ has roughly 3 times fewer nonzeros than $\mathbf{A}$ for `eu-2015-host-hc`, and roughly 4 times fewer for `it-2004-hc`. After profiling we could observe that, although $\mathbf{A'}$ had much fewer nonzero entries than $\mathbf{A}$, the nonzero entries in $\mathbf{A'}$ are more dispersed than those in $\mathbf{A}$, with $\mathbf{A}$ benefiting from contiguous memory accesses. The speedups are nevertheless significant, especially when we are dealing with larger graphs like `uk-2014-hc`.

In a subsequent experiment, we vertically split the matrix $\mathbf{A}^\top$ into c submatrices as described in Section 4, and evaluated the compression gain in light of the compression technique of the WebGraph framework, which can find more suitable references as the matrices become slimmer. We present our evaluation in Table 3, where we can observe a slight reduction in the number of nonzero entries when comparing the sums of the submatrices with the original matrix $\mathbf{A}^\top$. We can observe that scaling up c reduces the sum of all nonzero entries in the submatrices, while scaling up W can have a beneficial effect on large graphs. This effect is minor compared to some of the reductions in Table 2, however, so we did not evaluate whether it further sped up matrix-vector multiplications. Moreover, the WebGraph framework seems not to take particular advantage of this special structured set of submatrices, since the sum of the file sizes of the submatrices grows considerably with the number of submatrices c.

The main bottleneck of the whole computation was the recompression of the graphs or the compression of the submatrices. A larger graph instance can take several hours of pre-computation, or even longer on large graphs such as `eu-2015-hc` or `gsh-2015-hc`, where we omitted some parameter choices in Tables 3 and 2 because they would take too long to compute.

6 Computation with Bicliques

Another suitable format is the biclique extraction method of Hernández and Navarro [13]. They decompose the edges of G into a list of bicliques and a residual set of edges. A biclique is a pair of sets of nodes of the form (S_r, T_r), where every node of S_r has an edge to every node of T_r. We represent the $|S_r| \cdot |T_r|$ edges of each biclique (S_r, T_r) by listing both sets, which gives us $|S_r| + |T_r|$ integers, i.e., the identifiers of the nodes. These can be compressed by differential encoding with a universal coder. It has been shown that this format is competitive in compressing both webgraphs and social graphs.

Let $\mathbf{A}'$ denote the adjacency matrix representing the residual set of edges. To compute $\mathbf{A} \cdot \vec{x}^\top$, we compute for each biclique (S_r, T_r) the value $c_r = \sum_{j \in T_r} x_j$. We then allocate the vector $\vec{y}$ whose entries are initially set to zero. Subsequently, for each biclique (S_r, T_r) and each node identifier $i \in S_r$, we add c_r to y_i. Finally, for each residual edge $A'_{ij} = 1$, we add x_j to y_i. By doing so, the resulting vector $\vec{y}^\top$ is equal to the product $\mathbf{A} \cdot \vec{x}^\top$, and we obtained $\vec{y}$ in time proportional to the size of the compressed matrix.

We carry out a proof-of-concept implementation of this idea by building on top of the current biclique extraction software [13]. This software has some limitations that carry over our implementation. The most crucial one is that the number of total edges is expected to fit into 32 bits, which limits complete graphs to small sizes of $n \leq 2^{16}$. This was also the knock-out criteria for several aforementioned graph instances. We present the evaluation on the graphs that could be successfully processed by the software in Table 4. There, instances with names having a `-t` suffix represent the adjugate matrix $A^\top$.

Table 4 Evaluation of PageRank with either the plain matrix or with the adjacency matrix of the remaining nodes after biclique extraction. m' is the number of nonzero entries in $\mathbf{A}'$. b_S and b_T denote the total sizes of the left hands and the right hands of all bicliques. Δ_+ and Δ_- denote the number of spurious self-loops that have been added by bicliques or have been erroneously omitted in the set of remaining edges, respectively. t and t' denote the time for computing PageRank on the original adjacency matrix and on the adjacency matrix of the remaining nodes with the bicliques, respectively. Finally, t/t' is the speedup (or slowdown if < 1) observed in the computation of PageRank.

Graph	m' [M]	m/m'	b_S [K]	b_T [K]	Δ_+ [M]	Δ_- [M]	t' [ms]	t [ms]	t/t'
`amazon-2008-hc`	5.14	1.00	1.56	1.41	0.00	0.00	16.89	16.82	1.00
`amazon-2008-hc-t`	5.15	1.00	0.68	0.88	0.00	0.00	17.59	17.44	0.99
`enwiki-2021-hc`	119.13	1.26	8919.64	5241.63	0.05	0.05	532.39	517.51	0.97
`eu-2015-host-hc-t`	79.53	4.86	5539.82	10852.81	0.01	6.89	532.71	1022.59	1.92
`it-2004-hc`	154.28	7.46	30875.40	28862.12	0.65	11.66	996.45	2354.58	2.36

Another limitation we managed to overcome is that the biclique extraction software implicitly assumes that all nodes have self-loops. To compute the later PageRank evaluation based on the compressed representation with bicliques correctly, we perform two post-processing steps: First, we filter out those nodes that are simultaneously in both S_r and T_r for some r, but originally did not have a self-loop. Secondly, we add self-loops to each node that originally had a self-loop, but not in both S_r and T_r simultaneously for any r. We denote these erroneously added edges or erroneously removed edges by Δ_+ and Δ_-, respectively.

In the table, we additionally measure the sizes of the extracted bicliques with $b_S = \sum_r |S_r|$, and $b_T = \sum_r |T_r|$. We can see that the biclique sizes have a super-linear impact on the number of remaining edges m'. This is good for the PageRank computation, since the relatively small overhead of the additional computation for the bicliques dwarfs the computation with the adjacency matrix $\mathbf{A}'$, which has much fewer entries than the original matrix $\mathbf{A}$. Especially for large graphs with many bicliques such as `it-2004-hc`, we have a speed-up of 2.36. However, if the ratio m/m' between original edges and remaining edges is roughly at 1, our proposed technique is slightly slower. For these experiments we used the same machine as in Sect. 5, but devised an implementation in Rust, which is available at https://github.com/koeppl/matrixbiclique.

7 Conclusion

We have shown that the adjacency matrix compression scheme of Boldi and Vigna [12] as well as the biclique extraction of Hernández and Navarro [13] are suitable representations for computing matrix-vector products in time proportional to the *compressed* matrix sizes. We therefore can conclude that these compression formats not only save space but also speed up an operation that is crucial for graph analysis tasks. We plan to consider other formats where it is less clear how to translate the reduction in space into a reduction in computation time [18, 24, 25], and study which other relevant matrix operations can be boosted by which compression formats.

We also plan to investigate whether there is a better way to speed up matrix-matrix multiplications via compression, than by treating them as repeated matrix-vector multiplications. For example, suppose we have compressed two matrices

$$\mathbf{A} = \begin{bmatrix} \vec{v}_1 \\ \vdots \\ \vec{v}_n \end{bmatrix} \qquad \mathbf{B}^\top = \begin{bmatrix} \vec{w}_1 \\ \vdots \\ \vec{w}_n \end{bmatrix}$$

according to the Webgraph framework, with referencing vectors $\vec{r}$ and $\vec{c}$, and now we want to compute $\mathbf{C} = \mathbf{AB}$. Calculation shows

$$\begin{aligned} \mathbf{C}_{i,j} &= \vec{v}_i \cdot \vec{w}_j \\ &= \mathbf{C}_{i,\vec{c}_j} + \mathbf{C}_{\vec{r}_i,j} - \mathbf{C}_{\vec{r}_i,\vec{c}_j} + (\vec{v}_i - \vec{v}_{\vec{r}_i})(\vec{w}_j - \vec{w}_{\vec{c}_j})\,. \end{aligned}$$

In standard matrix-matrix multiplication, when we want to compute $\mathbf{C}_{i,j}$, we have already computed $\mathbf{C}_{i,\vec{c}_j}$, $\mathbf{C}_{\vec{r}_i,j}$ and $\mathbf{C}_{\vec{r}_i,\vec{c}_j}$, so we need only compute the product $(\vec{v}_i - \vec{v}_{\vec{r}_i})(\vec{w}_j - \vec{w}_{\vec{c}_j})$ of two vectors each of which — as the differences between a vector and its reference — is likely to be sparse.

If we compute that product by always scanning the nonzero entries of $\vec{v}_i - \vec{v}_{\vec{r}_i}$ and checking the corresponding entries of $\vec{w}_j - \vec{w}_{\vec{c}_j}$ (or vice versa) and performing a multiplication whenever one of those corresponding entries is also nonzero, then we are essentially computing $\mathbf{C}$ by repeated matrix-vector multiplications. If we always choose whichever of $\vec{v}_i - \vec{v}_{\vec{r}_i}$ and $\vec{w}_j - \vec{w}_{\vec{c}_j}$ has the smaller support, then we may obtain an additional speedup. Finally, it may be even faster to compute adaptively the intersection of the sets of positions of nonzero entries (see, e.g., [29]).

Acknowledgments. We thank Cecilia Hernández for providing us with her software extracting the bicliques, and a helpful description in how to run it. 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 UIDB/50021/2020 and PTDC/CCI-BIO/29676/2017]; the second author was funded by Academy of Finland [grant number 268324], Fondecyt [grant number 1171058] and NSERC [grant number RGPIN-07185-2020]; the third author was funded by JSPS KAKENHI [grant numbers JP21K17701 and JP21H05847]; the fourth author was funded by AEI and Ministerio de Ciencia e Innovación (PGE and FEDER) [grant number PID2019-105221RB-C41] and Xunta de Galicia (co-funded with FEDER) [grant numbers ED431C 2021/53 and ED431G 2019/01]; and the fifth author was funded by ANID – Millennium Science Initiative Program – Code ICN17_002.

Declaration

On behalf of all authors, the corresponding authors state that there is no conflict of interest.

References

[1] Newman, M.: Networks: An Introduction. OUP Oxford, Oxford, UK (2010)

[2] Chung, L.L.F.: Complex Graphs and Networks. Conference Board of the mathematical science. American Mathematical Society, Providence, RI, USA (2006)

[3] Elgohary, A., Boehm, M., Haas, P.J., Reiss, F.R., Reinwald, B.: Compressed linear algebra for declarative large-scale machine learning. Commun. ACM **62**(5), 83–91 (2019)

[4] Abboud, A., Backurs, A., Bringmann, K., Künnemann, M.: Impossibility results for grammar-compressed linear algebra. In: Proc. NeurIPS, pp. 1–14 (2020)

[5] Chakraborty, D., Kamma, L., Larsen, K.G.: Tight cell probe bounds for succinct boolean matrix-vector multiplication. In: Proc. STOC, pp. 1297–1306 (2018)

[6] Gagie, T., Gawrychowski, P., Puglisi, S.J.: Approximate pattern matching in LZ77-compressed texts. J. Discrete Algorithms **32**, 64–68 (2015)

[7] Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: Unified compression-based acceleration of edit-distance computation. Algorithmica **65**(2), 339–353 (2013)

[8] Lifshits, Y., Mozes, S., Weimann, O., Ziv-Ukelson, M.: Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica **54**(3), 379–399 (2009)

[9] Yang, E., Bian, J.: Bipartite grammar-based representations of large sparse binary matrices: Framework and transforms. In: Proc. ISITA, pp. 241–245 (2016)

[10] Ganardi, M., Hucke, D., Jez, A., Lohrey, M., Noeth, E.: Constructing small tree grammars and small circuits for formulas. J. Comput. Syst. Sci. **86**, 136–158 (2017)

[11] Lohrey, M.: Algorithmics on SLP-compressed strings: A survey. Groups Complex. Cryptol. **4**(2), 241–299 (2012)

[12] Boldi, P., Vigna, S.: The webgraph framework I: compression techniques. In: Proc. WWW, pp. 595–602 (2004)

[13] Hernández, C., Navarro, G.: Compressed representations for web and social graphs. Knowl. Inf. Syst. **40**(2), 279–313 (2014)

[14] Francisco, A.P., Gagie, T., Ladra, S., Navarro, G.: Exploiting computation-friendly graph compression methods for adjacency-matrix multiplication. In: Proc. DCC, pp. 307–314 (2018)

[15] Alman, J., Williams, V.V.: Further limitations of the known approaches for matrix multiplication. In: Proc. ITCS. LIPIcs, vol. 94, pp. 25–12515 (2018)

[16] Karande, C., Chellapilla, K., Andersen, R.: Speeding up algorithms on compressed web graphs. Internet Math. **6**(3), 373–398 (2009)

[17] Nishino, M., Yasuda, N., Minato, S., Nagata, M.: Accelerating graph adjacency matrix multiplications with adjacency forest. In: Proc. SDM, pp. 1073–1081 (2014)

[18] Brisaboa, N.R., Ladra, S., Navarro, G.: Compact representation of web graphs with extended functionality. Inf. Syst. **39**, 152–174 (2014)

[19] Henzinger, M., Krinninger, S., Nanongkai, D., Saranurak, T.: Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In: Proc. STOC, pp. 21–30 (2015)

[20] Larsen, K.G., Williams, R.R.: Faster online matrix-vector multiplication. In: Proc. SODA, pp. 2182–2189 (2017)

[21] Chung, F.: The heat kernel as the pagerank of a graph. Proceedings of the National Academy of Sciences **104**(50), 19735–19740 (2007)

[22] Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999)

[23] Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Proc. WWW, pp. 587–596 (2011)

[24] Grabowski, S., Bieniecki, W.: Merging adjacency lists for efficient web graph compression. In: Proc. ICMMI. Advances in Intelligent and Soft Computing, vol. 103, pp. 385–392 (2011)

[25] Claude, F., Navarro, G.: Fast and compact web graph representations. TWEB **4**(4), 16–11631 (2010)

[26] Boldi, P., Vigna, S.: Codes for the world wide web. Internet Math. **2**(4), 407–429 (2005)

[27] Boldi, P., Marino, A., Santini, M., Vigna, S.: Bubing: Massive crawling for the masses. ACM Trans. Web **12**(2), 12–11226 (2018)

[28] Chierichetti, F., Kumar, R., Lattanzi, S., Mitzenmacher, M., Panconesi, A., Raghavan, P.: On compressing social networks. In: Proc. SIGKDD, pp. 219–228 (2009)

[29] Barbay, J., López-Ortiz, A., Lu, T., Salinger, A.: An experimental investigation of set intersection algorithms for text searching. J. Experimental Algorithmics **14**, 3–7 (2010)