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 adjacency (binary) 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 $A$ be an $n \times n$ binary matrix and $x \in \mathbb{R}^n$ a vector. Matrix vector multiplication, either $x \cdot A$ or $A \cdot x^\top$, is not only a fundamental operation in mathematics, but also a key operation in various graph analysis tasks, when $A$ 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 $A$, 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 con-
sider the representation proposed by Boldi and Vigna [13]. The relevant observation
for us is that adjacency lists, i.e., rows in $A$, are compressed differentially with re-
spect to other similar lists, and thus one can reuse and “correct” the result of the
multiplication of a previous similar row with $x^\top$.

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

Computation-friendly matrix compression has been already considered by others, even
if indirectly. Karande et al. [8] 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. [9]. 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 [10].

The question addressed here can also be of interest for the problem of Online
Matrix-Vector (OMV) multiplication. Given a stream of binary vectors, $x_1, x_2, x_3, \ldots$,
the results of matrix-vector multiplications $x_i \cdot A$ can be computed faster than comput-
ing them independently, with most approaches making use of previous computations
$x_j \cdot A$, for $j < i$, to speed up the computation of each new product $x_i \cdot A$ [11, 12].
Nevertheless, none of those approaches preprocess matrix $A$ to exploit its redundan-
cies. Hence, by exploiting a suitable succinct representation of $A$ as we do here, an
improvement for OMV can be easily obtained, with computational time depending
on the length of the succinct representation of $A$ instead.

**PageRank**

Given $G = (V, E)$ a graph with $n = |V|$ vertices and $m = |E|$ edges, let $A$ 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 $M = D^{-1} \cdot A$, where $D$ 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 $M$
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 $M$, $M^k$, 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 $M$. Given a constant $0 < \alpha < 1$ and a probability
vector $p_0$, the PageRank vector $p_\alpha$ is given by the following recurrence [14]

$$p_\alpha = \alpha p_0 + (1 - \alpha) p_\alpha \cdot M$$

The parameter $\alpha$ is called the teleport probability or jumping factor, and $p_0$ is the
starting vector. In the original PageRank [15], the starting vector $p_0$ is the uniform
distribution over the vertices of $G$, i.e., $p_0 = 1/n$. When $p_0$ is not the stationary
distribution, $p_\alpha$ is called a personalized PageRank. Intuitively, $p_\alpha$ 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 $\alpha$ 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 $t \geq 1$:

$$p_t = \alpha p_0 + (1 - \alpha) p_{t-1} \cdot M$$ (1)

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

**Our Approach**

Our main idea is to exploit the copy-property of adjacency lists observed in some
graphs, such as Web graphs [13]. 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. Most compressed
representations for sparse graphs rely on these properties [16–18]. 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 [13]. Among several compression
techniques used in WebGraph, our approach makes use of list referencing.

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

$$A = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$
where \( \mathbf{v}_i \in \{0, 1\}^n \) is the \( i \)-th row, for \( i = 1, \ldots, n \). Let \( \mathbf{r} \in \{0, 1, \ldots, n\}^n \) be a referencing vector such that, for \( i \in \{1, \ldots, n\} \), \( r_i < i \) 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., \( r_i \in \{\max(1, i - W), \ldots, i\} \), 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 [13, 19].

**Proposition 1.** Given an \( n \times n \) binary sparse 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:

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

\[
w_i = \mathbf{v}_{r_i} \cdot \mathbf{x}^T
\]

Then we have that:

\[
\mathbf{A} \cdot \mathbf{x}^T = \mathbf{A}' \cdot \mathbf{x}^T + \mathbf{w}^T
\]

**Proof.** By definition,

\[
\mathbf{A}' \cdot \mathbf{x}^T + \mathbf{w}^T = \begin{bmatrix}
\mathbf{v}_1 \cdot \mathbf{x}^T - \mathbf{v}_{r_1} \cdot \mathbf{x}^T \\
\vdots \\
\mathbf{v}_n \cdot \mathbf{x}^T - \mathbf{v}_{r_n} \cdot \mathbf{x}^T
\end{bmatrix} + \begin{bmatrix}
\mathbf{v}_{r_1} \cdot \mathbf{x}^T \\
\vdots \\
\mathbf{v}_{r_n} \cdot \mathbf{x}^T
\end{bmatrix} = \begin{bmatrix}
\mathbf{v}_1 \cdot \mathbf{x}^T \\
\vdots \\
\mathbf{v}_n \cdot \mathbf{x}^T
\end{bmatrix} = \mathbf{A} \cdot \mathbf{x}^T
\]

Let us compute \( \mathbf{y}^T = \mathbf{A} \cdot \mathbf{x}^T \) by iterating over \( i = 1, \ldots, n \). 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 \( i = 1, \ldots, n \), set \( y_i = y_{r_i} + \sum_j A'_{ij} x_j \).

3. Return \( \mathbf{y} \).

Note that the number of operations to obtain \( \mathbf{y}^T = \mathbf{A} \cdot \mathbf{x}^T \) 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 $A$), $m'$ is the number of nonzeros in $A'$, 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. Speedups were only computed for web graphs.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$n$</th>
<th>$m$</th>
<th>$m'$</th>
<th>$m/m'$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-2015-hc</td>
<td>1,070,557,254</td>
<td>91,792,616,600</td>
<td>11,147,094,998</td>
<td>8.26</td>
<td>2.95</td>
</tr>
<tr>
<td>eu-2015-host-hc</td>
<td>11,264,052</td>
<td>386,915,963</td>
<td>110,215,204</td>
<td>3.52</td>
<td>1.42</td>
</tr>
<tr>
<td>gsh-2015-hc</td>
<td>988,490,691</td>
<td>33,877,399,152</td>
<td>7,080,094,116</td>
<td>4.78</td>
<td>1.89</td>
</tr>
<tr>
<td>it-2004-hc</td>
<td>41,291,594</td>
<td>1,150,725,436</td>
<td>226,809,391</td>
<td>5.08</td>
<td>2.05</td>
</tr>
<tr>
<td>uk-2014-hc</td>
<td>787,801,471</td>
<td>47,614,527,250</td>
<td>6,264,224,336</td>
<td>7.58</td>
<td>3.05</td>
</tr>
<tr>
<td>twitter-2010-hc</td>
<td>41,652,230</td>
<td>1,468,365,182</td>
<td>1,437,856,892</td>
<td>1.02</td>
<td>–</td>
</tr>
<tr>
<td>amazon-2008-hc</td>
<td>735,323</td>
<td>5,158,388</td>
<td>4,476,620</td>
<td>1.15</td>
<td>–</td>
</tr>
<tr>
<td>enwiki-2013</td>
<td>4,206,785</td>
<td>101,355,853</td>
<td>96,154,833</td>
<td>1.05</td>
<td>–</td>
</tr>
<tr>
<td>wordassociation-2011</td>
<td>10,617</td>
<td>72,172</td>
<td>71,463</td>
<td>1.01</td>
<td>–</td>
</tr>
</tbody>
</table>

Experimental Evaluation

We computed the number of nonzeros $m'$ in $A'$ for the adjacency matrix $A$ of several graphs available at http://law.di.unimi.it/datasets.php [13, 20, 21]. Whenever $|v_i - v_r| \geq |v_i|$, we kept $v_i$ as the row in $A'$, 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 $A'$ and $r$ were obtained directly from the Webgraph 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 $A'$ 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 [22]. 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 $A'$ 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-2014-host-hc and it-2004-hc. Our implementation took 81.5 and 120.0 seconds for eu-2014-host-hc and it-2004-hc, respectively. An equivalent implementation of PageRank, using the adjacency ma-
trix $A$ instead of $A'$, represented with WebGraph, took 115.5 seconds and 246.5 for 
**eu-2014-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 $A'$ has 3.52 times fewer nonzeros than $A$ for 
**eu-2014-host-hc**, and 5.08 times fewer for **it-2004-hc**. After profiling we could 
observe that, although $A'$ had much fewer nonzeros than $A$, the nonzeros in $A'$ are 
more dispersed than those in $A$, with $A$ 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 $A$ instead of matrix $A'$. 

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

**Final Remarks**

We have shown that the adjacency matrix compression scheme of Boldi and Vigna [13] 
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 [23]. 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 $A \cdot x^\top$, 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 
$y^\top$, 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 [16–18, 23], and study which other relevant 
matrix operations can be boosted by which compression formats.

**Acknowledgments**

Parts of this research were completed while the first author was visiting the University of Chile, funded by Marie Sklodowska-Curie [grant agreement No 690941], and 
while the second author was affiliated with the University of Helsinki and visiting 
the University of A Coruña. 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].


