Pushing the Envelope in Graph Compression

Panagiotis Liakos
University of Athens
Athens, Greece
p.liakos@di.uoa.gr

Katia Papakonstantinopoulou[†]
University of Athens
Athens, Greece
katia@di.uoa.gr

Michael Sioutis
Université Lille-Nord de France, CRIL-CNRS
Lens, France
sioutis@cril.univ-artois.fr

ABSTRACT

We improve the state-of-the-art method for the compression of web and other similar graphs by introducing an elegant technique which further exploits the clustering properties observed in these graphs. The analysis and experimental evaluation of our method shows that it outperforms the currently best method of Boldi et al. by achieving a better compression ratio and retrieval time. Our method exhibits vast improvements on certain families of graphs, such as social networks, by taking advantage of their compressibility characteristics, and ensures that the compression ratio will not worsen for any graph, since it easily falls back to the state-of-the-art method.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and Networks; E.4 [**Coding and information theory**]: Data compaction and compression; H.2.8 [**Database Applications**]: Data Mining

Keywords

Graph Compression; Compact Graph Representation; Social Network Graphs; Graph Clustering

1. INTRODUCTION

Real-world systems and phenomena that involve interactions among various entities are being modelled using graphs for decades now. The recent explosive growth of large-scale systems that are traditionally modelled as graphs, the world-wide web and social networks being typical examples, has intensified the need for compact, yet efficient, representations of graphs. In particular, we need compressed graph representations that allow mining without decompressing the graph. In this way, algorithms and applications with tasks that correspond to graph mining problems can take advantage of such representations to boost their performance. In particular, serving adjacency queries or maintaining and querying low-cost snapshots for archival purposes are common operations in such critical applications, and can benefit from the use of in-memory representations of graphs.

[†]This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: THALES. Investing in knowledge society through the European Social Fund.

The graphs we are interested in representing share some common features. First, they represent huge networks extending to millions of nodes, but the degrees (in/out-degrees) of the latter are power law distributed [9, 11], rendering the graphs to be rather sparse [14]. Moreover, the graphs exhibit the *locality of reference* property: nodes tend to have successors that are 'close' to them in a sense that depends on the context and the nature of the network. For instance, web pages often contain links to pages of the same web site or domain, and people in social networks are often friends with individuals from the same neighbourhood, university, or work. Furthermore, these graphs exhibit the *copy* property (or *similarity* property), which denotes that nodes occurring close to each other tend to have many common successors.

These properties induce various types of redundancy in the graphs' representations, and are taken into account when designing compression methods. The state-of-the-art approach to the compact representation of graphs is the method of Boldi and Vigna [7], further improved using a reordering of the graph [5] before compressing it. Several other approaches have been proposed, but they are slower, i.e., they improve the results of [7] only in terms of compression ratios and not in terms of access times of the graph's elements, they are efficient for small graphs only, or they are methods solely based on some usually computationally expensive reordering of the input graph. We note here that reordering a given graph results in an isomorphic graph, in which redundancy can be (hopefully) exploited by the algorithm more effectively. For example, in [5, 6] the authors introduce reorderings for which the Boldi and Vigna method yields an increased compression of web and social network graphs, when compared with the compression obtained using the graphs in their initial form. Hence, the reorderings can favour any compression algorithm that takes the aforementioned properties into account.

The web and social graphs may share the above properties, but feature a substantial difference in the way they are represented: while it is easy to order the nodes of a web graph in a meaningful way which favours its compression, there is no such obvious ordering for general networks, including social ones. As it is noted in [11], there exists some, yet unexplained, topological difference between social networks and web graphs that results in a less effective compression of the former, i.e., a larger compression ratio.

1.1 Related work

The need for compact representation of graphs emerged with the explosion of the size of the worldwide web, so the first such attempts focused on compressing web graphs. In the last dozen of years graph compression has turned into a very active research area and many algorithms have been proposed, some of them designed for more general graphs like the social network ones. Most algorithms in this direction try to offer a good space/time trade-off.

The structures traditionally used for the representation of graphs are the adjacency list and the adjacency matrix. The former is preferred for sparse graphs, i.e., graphs whose number of edges is $O(n)$, where n denotes the number of the graph's nodes, while the latter is used for dense graphs, i.e., graphs with $\Theta(n^2)$ edges. The locality of reference property, as well as the node similarity property, have been observed in most of the graphs we are interested in, and are often met in graphs that represent networks created by human activity. The central idea in graph compression algorithms is that they try to diminish the inner redundancy in the representation using the above structures, by exploiting the aforementioned properties.

The graph compression algorithms that have been proposed so far can be classified in the following three main categories: (*i*) algorithms for compressing web graphs, (*ii*) algorithms for compressing (also) more general graphs (mostly social network graphs), and (*iii*) algorithms that include or employ reordering of the graph in order to favour higher degree of compression. It is also very often the case that specific web graph compression algorithms were later enriched with new techniques in order to be able to compress social graphs as well.

In [22] the authors take into account the locality of reference and the copy properties for the case of the web and initiate research on web graph compression by maintaining compressed forms of the graph's adjacency lists. The highest compression ratios are achieved by the method of Boldi and Vigna [7], combined with a reordering using label propagation [5]. The WebGraph compression method introduced in [7] is indeed the most successful member of a family of approaches [1, 9, 21, 26] for compressing *web* graphs based on the statistical properties described in the introduction. In [7] Boldi and Vigna exploit the similarity of adjacency lists and the locality of reference of nearby pages using URL ordering for nodes. We present the techniques of the WebGraph framework briefly in Section 2.1.

In another line of work, Brisaboa et al. [8] propose a compact representation of the adjacency matrix that represents the graph. They partition the matrix in boxes and store each box in a way that allows quick access to it. In particular, they use a k^2-ary tree that records at each level which children contain at least one edge. The most important feature of this work is that it allows both forward and backward navigation of the graph. However, experiments show that even for the smallest possible value of k, viz., 2, which results in 4-bit sized leaves, the total compressed size wasted on leaves of the tree alone, is significantly greater than that achieved by other methods for graphs of greater size than the ones tested. The approach we propose in this work is to some extent similar to [8], in the sense that we represent parts of the adjacency matrix of a given graph. The difference with our approach is that we represent only some *dense* parts of the graph, those that are close to the main diagonal, and that we do not introduce extra overhead by using trees as indices.

Asano et al. [3] reorganize the adjacency matrix of the graph to bring the inter-host links close to the intra-host ones, and incorporate six different kinds of patterns to cover it. The compression ratio reported is impressive, but the additional cost imposed for the matching of the original indices of the inter-host links with the new local indices used is not considered in the presented results. This approach is similar to our proposed method, with the difference that in our method no overhead for lookup tables is introduced.

Claude et al. in [13] use a different compression scheme for web graphs that does not achieve better compression ratios than [7], but allows for faster navigation on the graph. However, their approach does not scale up due to the large amount of memory and long time required during the computationally expensive compression phase. Later on, Claude et al. combined this algorithm with the techniques of [8], partitioning the input graph and applying a separate technique to each part (i.e., applying [13] on one part and [8] on the other), to compress web as well as social network graphs [12]. A similar partitioning is applied in our approach as well, but the methods used to compress the subgraphs are entirely different.

In [11] Chierichetti et al. view the problem of graph compression from a theoretical point of view and study the extent to which a large social network can be compressed. They show that the compressibility of social networks is very different than that of web graphs. Their proposed method, however, is a compression scheme rather than a compressed data structure, as noted in [5], i.e., it aims solely at minimizing the size of the compressed graph (bits/edge) instead of providing fast access to each edge.

The locality of reference property of a graph reflects on its adjacency matrix in the following way: using a proper ordering of the nodes' labels, i.e., an ordering in which labels of densely connected nodes are close to each other, many edges fall close to the main diagonal of the adjacency matrix. Such orderings are preferred in practice, but finding the ordering that minimizes the distance of the edges from the main diagonal is NP-hard [25]. Intuitively, if we have some good clustering of the graph, based solely on the link structure, and assign consecutive labels to the nodes in each cluster, the lexicographic ordering of the labels is rather good in the above sense.

Such an extrinsic ordering appears naturally in the case of worldwide web. Web graph representations assume that each URL corresponds to some identifier. Moreover, it is assumed that URLs are alphabetically sorted [4], and this naturally puts together the pages of the same domain. As a result, the locality of reference translates into closeness of page identifiers. However, extrinsic orderings are not obvious for all graphs, so for social or bibliographic citation graphs, finding a good ordering is a challenging issue. In [6] Boldi et al. test some known orderings of the nodes and propose some new ones, and study their effect on the compression of web and social graphs. They show that using these orderings for the input non-web social graphs, the WebGraph framework [7] yields results that are very close to the results of [11]. In [5] the authors introduce a reordering algorithm called *Layered Label Propagation* (LLP), and employ it to compress social networks. This algorithm is based on clusterings and orderings and can reorder very large graphs quite fast. The experimental evaluation of this approach shows that combining the ordering produced by LLP with the WebGraph framework outperforms all currently known techniques, both for web graphs and for social networks. Some methods that claim to yield lower bits/edge ratios [10, 11, 20] do not address the issue of retrieving the edges fast. In [16] the authors introduce SLASHBURN, an ordering method that offers the best bits per egde ratio according to the information theoretic lower bound, among other competing methods.

Buehrer and Chellapilla [10] exploit complete bipartite subgraphs (bicliques) on web graphs, i.e., groups of pages that share the same outlinks, and replace them with virtual nodes. However, the compression they achieve is not better than the compression of [7] while they also fail to be competitive speedwise, since they fall into the class of compression schemes rather than compressed data structures [5]. In computing the compressed size they do not take into account the offset, which, however, increases significantly with the increase of the compression ratio.

Clustering according to some meaningful measure naturally brings together nodes that are connected with the locality of reference or copy property. This is particularly useful in social networks where there is no apparent numbering of the nodes that brings them close to each other. In [20] the authors decompose the graph into small dense subgraphs, which can be represented more efficiently in terms of space. Their comparison with [7] is based on the naive approach of maintaining both the original graph and its transposed version, whereas a more sophisticated approach, indicated in [5], outperforms them. In [15] the authors generalize on [10, 20], where the authors find bicliques and cliques respectively, and adapt clustering algorithms to find broader constructions that lie in between. They show that these more general dense subgraphs appear sufficiently more often than cliques and bicliques, thus designing a more general compact representation for them pays off.

Apostolico and Drovandi [2] visit the graph in a breadth-first fashion while compressing, and exploit *locality* and *similarity* by referencing the previous successor of the same node, or the successor of the previous list that is in the same ordinal position. The blocks of identical successors are recorded only once. However, their method is outperfomed by [5], and by a big margin as far as social network graphs are concerned.

In [23] Safro and Temkin present a multiscale approach for the network minimum logarithmic arrangement problem, i.e., the problem of finding an intrinsic ordering that optimizes directly the sum of the logarithms of the gaps (numerical difference between two successive neighbours). The resulting ordering may be used for graph compression if combined with a compression scheme like the WebGraph framework [7]. According to [5], some preliminary tests show that these orderings are promising especially on social networks; however, the implementation does not scale well for datasets with more than a few millions of nodes and so it is impractical for compressing large-scale graphs.

Our approach benefits from the reordering that results after applying *Layered Label Propagation* [5] on the input graph; we could also take advantage of other orderings such as the ones examined in [6, 16].

1.2 Short description of results

In this paper, we concentrate on the compression of web, social network and other similar graphs. Since the highest compression ratios are achieved by the state-of-the-art algorithm of Boldi et al. [5, 7], we build on it, making the following contributions: (*i*) we improve the compression scheme of Boldi and Vigna (BV) [7] by exploiting the locality of reference property observed in these kinds of graphs in a different way than in [7] and, thus, go beyond the state-of-the-art in graph compression, and (*ii*) we evaluate experimentally our algorithm and show that it achieves a better compression ratio than BV, while allowing the retrieval of elements of the graph faster than BV.

1.3 Organization

The organization of this paper is as follows. In Section 2 we present the overview of our approach along with some theoretical analysis. In Section 3 our algorithm is presented and its complexity is discussed. In Section 4 we evaluate our approach experimentally. Finally, in Section 5 we conclude and give directions for future research.

2. OVERVIEW OF OUR APPROACH

This section presents our approach for compressing directed graphs. Our approach builds on and improves the Boldi and Vigna compression method of the *WebGraph framework* [7]. For the sake of simplicity, we will refer to the Boldi and Vigna method as BV throughout the paper. We first review the BV techniques (Section 2.1), then we isolate a dense subgraph of the input graph (Section 2.2), in particular a stripe around its main diagonal, and provide an explanation of how we can compress it separately along with a theoretical analysis of this approach (Section 2.3).

2.1 The Boldi et al. techniques

In [7], Boldi and Vigna propose a number of techniques that exploit *locality* and *similarity*, two properties that are known to appear in the links of a web graph [22].

The adjacency lists of a graph are pictured firstly using a modified gap representation, that utilizes the *locality* property, and then as bit vectors, named *copy lists*, that take advantage of the fact that the adjacency lists share large subsequences of edges (*similarity* property). *Copy lists* are further compressed with a variation of run-length encoding, since they tend to contain runs of 0s and 1s.

The number of previous adjacency lists that are examined in order to discover possible reference lists is called *window*, and its size poses a tradeoff between compression ratio and compression/decompression time. The maximum reference count is a second parameter used by this scheme, that imposes a limit on the lengths of reference chains.

The remaining extra nodes exhibit consecutivity as well. Hence, integer intervals are used for their compression, but only for the subsequences that correspond to intervals whose length is not below a certain threshold (L_{min} in [7]). The list of the *residuals* (remaining integers) is compressed in a differential manner.

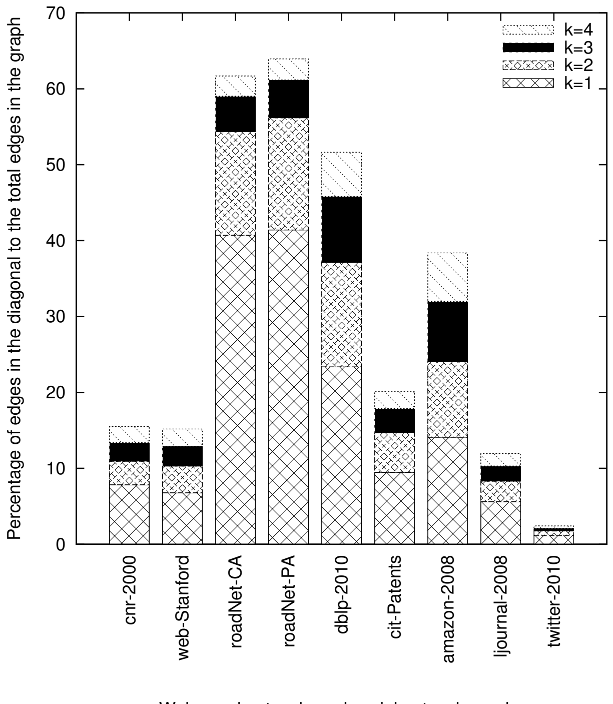


Figure 1: Percentage of edges contained in the diagonal area of web, road network and social network graphs.

In [5], Boldi et al. apply a reordering algorithm that brings the graph to a state where the aforementioned properties can be further exploited.

2.2 Exploiting the dense part of the graph

We improve the state-of-the-art algorithm BV for the compression of web graphs [7], by proposing to store separately the denser part of the graph, i.e., the part of the graph corresponding to the edges that are close to the main diagonal of the graph's adjacency matrix.

Graphs created by human activity usually possess the locality of reference property and the copy property, which are surfaced after applying LLP [5] on them, or any other clustering technique, e.g., [6, 16], that permutes the graph in a similar fashion. We exploit these properties to improve the compressed data structure. Due to the above properties, an edge is with high probability *close* to the main diagonal of the adjacency matrix representing the graph. Hence, given that these graphs are generally rather sparse due to the power law distributed nodes' degrees (indegrees and outdegrees) [14], the graph corresponding to the main diagonal and the area around it is denser than the rest of the graph. We call this area the *diagonal stripe*, and formally define it as follows:

DEFINITION 1. *For a graph $G = (V, E)$ and $k \in \mathbb{Z}_+$, the k-diagonal stripe of G comprises the following set of entries: $\{(i, j) \mid i - k \leq j \leq i + k$ and $i, j \in \{0, \dots, |V|\}\}$.*

To illustrate, in the left part of Figure 2 we present within a bold line the 3-diagonal stripe of an example adjacency matrix.

In the graphs we examined experimentally, large number of edges tend to be in the diagonal stripe, meeting our expectations regarding the locality of reference property. This trend for $k \in \{1, 2, 3, 4\}$ for the graphs of our dataset, described in detail in Section 4.2, is illustrated in Figure 1.

As k increases, the bits/edge ratio required to store the stripe increases as well. The density of edges in the diagonal stripe decreases as we are moving farther away from the diagonal, where edges are met less frequently. However, even a stripe of smaller density is sometimes useful, as it may lead to higher compression of the whole graph.

The computation of the bits/edge needed to represent the diagonal stripe is straightforward if we know the percentage of edges in it: Consider $k \in \mathbb{Z}_+$ and a graph $G = (V, E)$ with a percentage p of E belonging in the k-diagonal stripe. These edges are represented with $\frac{(2k+1)|V|}{p|E|}$ bits/edge. For example, graph `roadNet-PA` of our dataset, described in Section 4.2, has 68.41% of its edges in the 7-diagonal. Therefore these edges are represented with $\frac{15|V|}{0.6841|E|} = 7.76$ bits/edge.

Isolating the diagonal stripe in a way similar to [19] and the rest of the graph as explained in [7], achieves better compression than using the method presented in [7] alone, as we later demonstrate experimentally (Section 4). Table 1 presents the compression ratio achieved by BV for the remaining graph for various diagonal stripe widths, which is essentially the lower bound of our overall compression.

2.3 Compressing the diagonal stripe

In this section we describe the motivation and sketch the techniques behind isolating a dense subgraph of the input graph, in particular a stripe around its main diagonal, and compressing it separately. We also present some theoretical analysis for our proposed approach.

Adjacency matrix format.

In order to exploit the high concentration of edges in the diagonal stripe, and, thus, take advantage of the locality of reference property, we store it separately from the rest of the graph in the format of an adjacency matrix. We opted for the adjacency matrix representation of the diagonal as the high concentration of edges in the diagonal forms a dense graph. Formally, a graph $G = (V, E)$ is dense if $|E| = \Theta(|V|^2)$.

Data compression.

Using data compression techniques that exploit the redundancy of the diagonal stripe, represented by an adjacency matrix as described above, allows us to reduce the size of the stripe significantly. Shannon's source coding theorem states that it is impossible to compress with an average number of bits per symbol less than the entropy of the source. We present a proposition that imposes an upper bound to that limit, and provides us with an estimation of the space requirements of our method for the dense part of the graph. Comparing this estimation for various widths of the diagonal stripe of a graph, to the compression ratio of the state-of-the-art method, allows us to assess the overall room for improvement and the optimal width of the stripe. However, the estimation on the latter is far from accurate due to the delicate balance between easing the task of compressing the rest of the graph by including as many edges as possible in the diagonal stripe and minimizing its ratio.

PROPOSITION 1. *Consider $k \in \mathbb{Z}_+$ and a graph $G = (V, E)$ with a percentage p of its edges belonging in the k-diagonal*

graph	**BV**	*lower bound of our compression*			
		$k = 1$	$k = 2$	$k = 3$	$k = 4$
`cnr-2000`	3.71	3.37	3.28	3.22	3.16
`web-Stanford`	4.06	3.76	3.63	3.56	3.48
`roadNet-CA`	13.30	10.39	9.31	8.89	8.58
`roadNet-PA`	12.86	10	8.85	8.41	8.09
`dblp2010`	8.63	7.47	6.75	6.37	6.02
`cit-Patents`	14.72	14.21	13.88	13.67	13.51
`amazon-2008`	10.77	10.32	9.93	9.62	9.29
`ljournal-2008`	11.84	11.60	11.47	11.39	11.31
`twitter-2010`	14.52	14.41	14.35	14.32	14.29

Table 1: Lower bound of our compression: The bits/edge required by BV for the graph apart from the diagonal stripe.

stripe. The minimum expected compression ratio of the diagonal stripe is upper bounded by $\frac{\log\binom{(2k+1)|V|}{p|E|}}{p|E|}$ *bits/edge.*

PROOF. The diagonal stripe consists of $(2k+1)|V|$ bits and exactly $p|E|$ of them represent edges. We model the stripe as a random variable $X \in \{0,1\}^{(2k+1)|V|}$.

Shannon's source coding theorem states that the minimal possible expected length of codewords, which in our case is the best attainable compressed size of the diagonal stripe, is no less than the entropy of the input word (diagonal stripe) [24].

The entropy of X is $H(X) = -\sum_{i=1}^{n} p_i \log p_i$, where n is the number of all possible diagonal stripes and p_i is the probability of stripe i. As $n = \binom{(2k+1)|V|}{p|E|}$, and assuming that all possible stripes are equally likely, the maximum entropy becomes

$$H(X) = \log\binom{(2k+1)|V|}{p|E|}.$$

According to Shannon, the minimal expected wordlength S is $\mathbf{E}[S] = H(X)$. Thus, since we have $p|E|$ edges, the minimum expected compression ratio is

$$\frac{H(X)}{p|E|} = \frac{\log\binom{(2k+1)|V|}{p|E|}}{p|E|}.$$

□

For example, for the graph `roadNet-PA` the upper bound of the minimum expected compression ratio of the 7-diagonal stripe is $\frac{\log\binom{15|V|}{0.6841|E|}}{0.6841|E|} = 2.98$ bits/edge.

Minimizing the compression ratio with techniques such as Huffman or Arithmetic coding [27] may have a negative impact on the time needed to access the elements of the graph, as we will then need to decompress large parts, or even the whole diagonal stripe, in order to answer simple queries. We wish to retain the ability to access the elements of the stripe in constant time after compressing them. Thus, we encode them using a form of lossy, but fixed-length encoding to preserve the direct access of the edges.

3. COMPRESSING THE GRAPH

This section presents BV+, an algorithm for compressing directed graphs that is the product of our line of thinking in Section 2. BV+ is outlined in Algorithm 1. BV+ receives as input a directed graph $G = (V, E)$, and parameters k and b, and gives as output a compressed representation of G.

Algorithm 1: BV+(G, k, b)

```
input : A directed graph $G = (V, E)$, and parameters $k$ and $b$.
output: A compressed representation of $G$.
1  begin
2      setNonD ← set();
3      $k$-diagonalStripe ← array(array($\underbrace{[000\dots0]}_{2k+1 \text{ bits}}$) × $|V|$);
4      foreach $(u, v) \in E$ do
5          if $u - k \le v \le u + k$ then
6              $k$-diagonalStripe[$u$][$v$] ← 1;
7          else
8              setNonD ← setNonD ∪ $(u, v)$;
9      seqDict ← dict();
10     foreach *seq* ∈ *k-diagonalStripe* do
11         if *seq* ∉ *seqDict* then
12             seqDict[seq] ← 1;
13         else
14             seqDict[seq]++;
15     foreach *(key, value)* ∈ *seqDict* do
16         seqDict[key] ← value × # of 1s ∈ key;
17     seqDict ← sort seqDict by value (desc. order);
18     seqSet ← {first $2^b - 1$ sequences (keys) of seqDict};
19     foreach *seq* ∈ *k-diagonalStripe* do
20         if *seq* ∈ *seqSet* then
21             *use b bits to compress seq*;
22         else
23             bestSeq ← bestSubset(seqSet, seq, $k$);
24             *use b bits to compress bestSeq*;
25             setNonD ← setNonD ∪ {edges of seq that
                   were left out of bestSeq};
26     *compress setNonD using* BV;
```

As a first step, the algorithm constructs the k-diagonal stripe of graph G and the set of all edges that do not belong in the k-diagonal stripe (lines 2-8). The k-diagonal stripe can be considered as an array of bit arrays where each bit array consists of exactly $2k+1$ elements and corresponds to a row of the diagonal stripe as illustrated in Figure 2. The value of an element equal to 1 signifies the presence of an edge. Likewise, the value of an element equal to 0 signifies the absence of an edge. The first and last rows in the k-diagonal stripe are complemented with 0s to fix the number of $2k+1$ elements for each row.

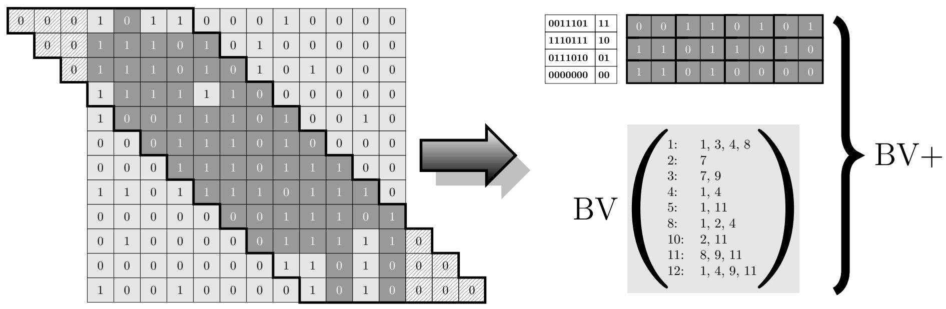


Figure 2: Compressing the graph with BV+.

```
Function: bestSubset(seqSet, seq, k)
input  : A set of candidate sequences, seqSet,
         parameter $k$, and a given sequence seq.
output: Best available sequence bestSeq.
1  begin
2      bestSeq ← array($\underbrace{[000\ldots0]}_{\text{2k+1 bits}}$);
3      max ← 0;
4      foreach candidateSeq ∈ seqSet do
5          counter ← 0;
6          flag ← False;
7          foreach $i \leftarrow 1$ to $2k+1$ do
8              if candidateSeq[i] = 0 and seq[i] = 1 then
9                  flag ← True;
10             if candidateSeq[i] = seq[i] = 1 then
11                 counter++;
12         if counter ≥ max and !(flag) then
13             bestSeq ← candidateSeq;
14             max ← counter;
15     return bestSeq;
```

Diagonal stripe compression.

As already mentioned, the k-diagonal stripe consists of $|V|$ rows where each row comprises $2k+1$ elements. We create a dictionary to hold information about our rows (line 9). Every row, that is, every sequence of $(0,1)$-elements, yields an integer value. We have empirically observed that the frequencies of these integer values tend to follow a power law distribution, so we decided to pick the values that contain the higher volume of information.

We iterate over the set of rows, and store each distinct $(2k+1)$-bit array along with its frequency in the set of rows in the k-diagonal stripe in our dictionary (lines 10-14). We do not rest on using the $(2k+1)$-bit arrays met most frequently among the rows, but we also take into account the number of edges they represent. More explicitly, let us suppose we had used $k=3$ and we had observed the sequences 0001000 and 0010110 occurring 500 and 300 times respectively. While the first sequence is more frequent, it fails to represent more than one edge. Hence, the second sequence is in fact preferable. The multiplication of the frequencies with the number of bits that are set guarantees that the chosen representations will not only occur often in the particular diagonal, but will represent a significant amount of edges as well, thus minimizing the overall bits per edge ratio (lines 15-16).

Then, we choose an integer number b of bits to use for their representation and represent only the 2^b-1 most appropriate of these sequences, each one using a binary number of b bits. This is done by sorting these sequences by the product of their frequency times the number of edges each one contains (line 17), i.e., the number of 1s in their binary representation, and then picking the first 2^b-1 of them (line 18). In this way, we make sure that we will pick not just the most frequent values in the diagonal, but also the most important ones, because picking representations that hold a limited number of edges would lead to a waste of bits. The role of b is to determine the channel capacity, and with it, the loss rate of our scheme. The optimal value for b is highly dependent on the distribution that the frequencies of the values follow. We keep one b-bit binary number to denote the absence of edges in a specific row of the diagonal stripe.

As a final step, we iterate again over the set of rows of the k-diagonal stripe, and for each row of the diagonal stripe we use its compressed representation if the corresponding sequence exists among the ones picked in the aforementioned step (lines 20-21), or the compressed representation of the *best available* sequence and add the missing edges to the set of edges that do not belong in the k-diagonal stripe (lines 22-25). In the latter case, by *best available*, we denote the sequence that has the most $1s$ in the same position with the sequence in question from the ones picked, and does not have a single 1 where this sequence has a 0. The best available sequence is provided by function bestSubset, which receives as input a set of candidate sequences, parameter k, and a given sequence. Function bestSubset first initializes a best sequence candidate, represented by a bit array, with $2k+1$ number of 0s (line 2), and it also initializes a counter that holds the number of 1s that two sequences have in common (line 3). The best available sequence, i.e., the one whose set of positions of 1s in it is the best (maximal) subset of the sequence in question, is then calculated by iterating the set of candidate sequences and performing some checks (lines 4-14), and, finally, returned (line 15). Note that even if no best available sequence is found among the set of candidate sequences, the initialized best available sequence candidate (line 2) will be returned. In the aforementioned example, suppose that the second sequence, viz., 0010110, was indeed elected among the 2^b-1 ones, while another sequence 0011110 was not. We utilize their similarity by using the representation of the former one for the latter one as well, and manage to capture 3 of its 4 edges without extra cost.

In the upper right part of Figure 2 we can see the mapping of the selected values to their b-digit representation and the compressed diagonal that results after applying the actions described above.

The edges that are excluded during this step are added to the ones existing outside of the diagonal stripe. These edges will be then compressed using BV, thus, our overall method is lossless. In Figure 2 the edges of the diagonal that are compressed as described above are contained in dark grey cells, while those that are compressed using BV are in light grey cells. When every row has been substituted with a compressed representation, the compressed diagonal is ready.

As a result of using the above procedure, for the graph `roadNet-PA` the compression ratio of the 7-diagonal stripe with b set to 2 is 1.95 bits/edge. However, even a compression ratio greater than the upper bound of the minimum expected (2.98 bits/edge as obtained in the example in Section 2.3 for `roadNet-PA`), and in any case smaller than that of the uncompressed graph, would be acceptable as we do not need to decompress the whole stripe to access the desired edges, in contrast to more compact entropy encoding approaches, such as Huffman or Arithmetic coding [27].

Non Diagonal part compression.

The final step for algorithm BV+ is to compress the remaining edges, i.e., the edges that initially belonged outside of the k-diagonal stripe (line 8), together with the edges that were left out during the compression of the diagonal stripe (line 25), with the BV method (line 26). As shown in Figure 2, the output of BV+ is the lossy compressed representation of the diagonal stripe plus the output of BV for the remaining elements. Besides using exclusively the BV method, the straightforward nature of our approach and the structure of our algorithm makes it an attractive technique that any compression scheme can benefit from.

3.1 Size of the compressed graph

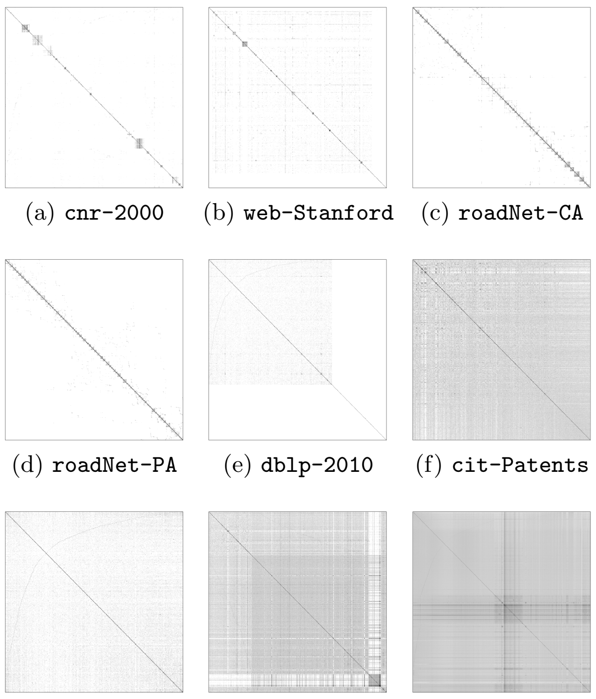
Let n be the number of nodes in the graph (i.e., $n = |V|$) and b be the parameter that ultimately defines the width of a compressed representation of the diagonal stripe, as described earlier. The size of the compressed graph is equal to $bn + S_{BV}$ bits, where S_{BV} is the size of the set of edges that are outside of the diagonal stripe plus the set of edges that were left out of the fixed length encoding of the diagonal (during the compression of the diagonal), compressed using the BV algorithm.

3.2 Time Complexity

Here, we discuss the time complexity of BV+ and compare it to the complexity of BV. The diagonal stripe is compressed in linearithmic time in the worst case ($O(n \log n)$) and the remaining edges in time less than with BV, as they form a graph that is smaller than the initial one.

- The time complexity of verifying the existence of a specific edge is $O(1)$ if the edge belongs in the *compressed* diagonal stripe, and less than that of BV otherwise[1].
- The time complexity of retrieving all neighbours of some node is $O(b)$ for the neighbours that belong in the *compressed* diagonal stripe, and less than that of BV for the rest of the neighbours[1].

[1]Since edges have to be retrieved from a compressed by BV graph, which is initially smaller than the input graph.



(a) `cnr-2000` (b) `web-Stanford` (c) `roadNet-CA`
(d) `roadNet-PA` (e) `dblp-2010` (f) `cit-Patents`
(g) `amazon-2008` (h) `ljournal-2008` (i) `twitter-2010`

Figure 3: Heat maps of the adjacency matrices of web (a, b), road network (c, d), citation (e, f), and social network (g, h, i) graphs.

The efficiency of our approach benefits from the existence of multi-core processors, since in any multi-core (e.g., 2-core) machine the aforementioned queries in and outside the diagonal stripe take place in parallel, thus, the makespan is the longer among the two tasks.

We infer that BV+ outperforms BV in terms of time needed for searching/retrieving edges in a graph compressed using any one of them, and we also show this experimentally later in Section 4.

The high compression of BV has a negative impact on the time needed to access some of the graph's elements: the retrieval of the incoming edges of a specific node becomes involved [7]. BV+ induces an improvement in this aspect, as part of the graph's edges, i.e., the part that belongs in the diagonal stripe, is accessed in constant time.

4. EXPERIMENTAL EVALUATION

We implemented and tested our approach on a wide variety of large scale graphs. In Section 4.1 we list the technical specifications of the machine used for implementing and testing our algorithm, while in Section 4.2 we describe the dataset used for our experiments. We present the results of our experiments, i.e., compression ratios and times, in Sections 4.3 and 4.4 respectively, and discuss the role of the algorithm's input parameters in Section 4.5.

4.1 Technical specifications

We implemented and ran algorithm BV+ using Java 7; our code is available upon request. The experiments were carried out on a computer with an Intel®Core^{TM} 2 Duo CPU E8400, with a CPU frequency of 3.00GHz, a 6MB L2

graph	*# nodes*	*# edges*	*# edges in compressed diagonal*	*compression ratio (bits/edge)*		*BV+ parameters*	
				BV	**BV+**	***k***	***b***
cnr-2000	325,557	3,216,152	194,639	3.71	3.62	17	2
web-Stanford	281,903	3,985,272	270,220	4.06	3.90	1	2
roadNet-CA	1,965,206	5,533,214	3,569,145	13.30	10.58	7	6
roadNet-PA	1,088,092	3,083,796	2,062,741	12.86	10.07	7	6
dblp-2010	326,186	1,615,400	928,702	8.63	7.2	24	7
cit-Patents	3,774,767	33,037,894	6,303,138	14.72	14.25	9	6
amazon-2008	735,323	5,158,388	3,057,268	10.77	10.07	23	15
ljournal-2008	5,363,260	79,023,142	6,045,619	11.84	11.78	2	4
twitter-2010	41,652,230	1,468,365,182	37,906,525	14.52	14.42	17	6

Table 2: Comparison with BV method.

cache and a total of 8GB DDR2 800MHz RAM. Only one of the CPU cores was used for the experiments.

4.2 Dataset

The dataset that we used to apply and test our compression technique, comprises nine well-studied [5, 7, 11, 18, 20] web, road network, citation, and social network graphs. Figure 3 provides an illustration of their adjacency matrices, where one can clearly see how the diagonal stands out for all the graphs. The origin and characteristics of our graphs are summarized in the following list:

- `cnr-2000`: a web graph from a crawl of the Italian CNR domain. It comprises 325,557 nodes and 3,216,152 edges.[2]
- `web-Stanford`: a web graph from Stanford University, collected in 2002. It comprises 281,903 nodes and 3,985,272 edges.[3]
- `roadNet-CA`: the road network of California. It comprises 1,965,206 nodes and 5,533,214 edges.[3]
- `roadNet-PA`: the road network of Pennsylvania. It comprises 1,088,092 nodes and 3,083,796 edges.[3]
- `dblp-2010`: an undirected scientific collaboration network graph from the DBLP bibliography service. Each vertex represents an author and an edge links two vertices if they have worked together. It comprises 326,186 nodes and 1,615,400 edges.[2]
- `cit-Patents`: a citation graph which includes all citations made by U.S. patents granted between 1975 and 1999. It comprises 3,774,767 nodes and 33,037,894 edges.[3]
- `amazon-2008`: a symmetric graph describing similarity among books as reported by the Amazon store. It comprises 735,323 nodes and 5,158,388 edges.[2]
- `ljournal-2008`: LiveJournal is a virtual community social site started in 1999; in this social network friendship is non-symmetric so the graph is directed. It comprises 5,363,260 nodes and 79,023,142 edges.[4]
- `twitter-2010`: Twitter is a social networking and microblogging service; To the best of our knowledge, this is the largest available social network graph. It comprises 41,652,230 nodes and 1,468,365,182 edges.[5]

The aforementioned graphs vary in category, size, and type, and are therefore very good candidates for examining the effectiveness of our proposed method.

[2]Collected by LAW: http://law.di.unimi.it/

[3]Collected by SNAP: http://snap.stanford.edu/snap/

[4]Collected in [11], retrieved from LAW: http://law.di.unimi.it/

[5]Collected in [17], retrieved from LAW: http://law.di.unimi.it/

method	*graph*	*compression ratio (bits/edge)*	
		other	**BV+**
[2]	ljournal-2008	14.97	11.78
	amazon-2008	12.39	10.07
	dblp-2010	7.47	7.2
[20][6]	web-Stanford	9.88	3.90
	cit-Patents	25.69	14.25

Table 3: Comparison with other methods.

4.3 Compression ratio comparison

We compared our approach with BV as well as with other graph compression methods. The results are outlined in Tables 2 and 3 respectively.

Table 2 shows the number of nodes and edges of each graph, the compression ratio achieved by the BV technique [7], and the one achieved by our proposed method (BV+) for all the graphs tested. The number of edges that was ultimately included in the compressed diagonal stripe and its parameters are also displayed.

For the two web graphs, we observed an improvement of 2.4% for `cnr-2000` and 3.9% for `web-Stanford`. Our method attained impressive results for the two road network graphs. An improvement of 20.5% and 21.7% was achieved with the use of BV+ for `roadNet-CA` and `roadNet-PA` respectively. Regarding the citation and social network graphs, our method had a large impact on `dblp-2010` and `amazon-2008`, achieving a 16.6% and a 6.5% compression ratio improvement over the compression ratio of BV respectively, and offered smaller, but significant nonetheless, compression ratio improvements for the other three graphs.

The higher levels of compression for certain graphs is due to the fact that a higher percentage of their edges exist in the diagonal stripe, as opposed to the rest of the graphs. However, even with the less intense clustering, our technique manages to reduce the compression ratio of BV significantly. As it can be seen in Figure 1, these graphs are `roadNet-CA`, `roadNet-PA`, `dblp-2010`, and `amazon-2008`. The somewhat smaller impact of BV+ on the compression ratio of `web-Stanford`, `cnr-2000`, `cit-Patents`, `twitter-2010`, and even `ljournal-2008` is due to the fact that BV does not leave enough room for improvement, as Table 1 illustrates. That is, the remaining edges that are assigned to BV, occupy most

[6]In [20] the algorithm provides both predecessors and successors but a simple strategy proposed in [5] indicates that less than double bits per link are needed for a fair comparison.

graph		cnr-2000	roadNet-CA	ljournal-2008
Edge Exists	**BV+ (D)**	645	600	663
	BV+ (Non-D)	2,286	832	4,373
	BV+ (total)	2,187	728	4,089
	BV	2,397	923	4,518
Successors	**BV+ (D)**	643	668	627
	BV+ (Non-D)	1,947	849	1,940
	BV+ (total)	1,868	768	1,840
	BV	2,159	891	2,009

Table 4: Access times (in ns) for a web, a road network, and a social network graph.

of the final compressed file. The lower bound essentially signifies the compression that we would be able to achieve if we could represent the stripe using 0 bits/edge. For the latter graphs the lower bound does not deteriorate at a satisfying rate as k increases, i.e., it remains almost stable.

In case our method provided no improvement for a given graph, it could easily fall back to BV instead of BV+ by setting b to zero.

Table 3 shows the comparison of our approach to two other recent methods [2, 20]. The methods were examined in [5] and were shown to be inferior than BV, but we chose to include this comparison for reasons of completeness.

We also evaluated BV+ for graphs of our dataset after SlashBurn [16] had been applied on them. Even though the reordering of SlashBurn favoured BV+ over BV for the graphs tested, the compression ratio achieved using this representation was significantly larger, as exploiting SlashBurn does not seem to aid the compression techniques of BV.

4.4 Access time comparison

Table 4 presents the results obtained by the comparison of BV and BV+ as far as access times are concerned. We tested the responsiveness of the two methods when asking if a node is a successor of another one (*Edge Exists*), and when inquiring all the successors of a node (*Successors*).

For the former query, we searched for every edge present in the graph and calculated the mean average of those that were held in the compressed diagonal representation and of those that were compressed using the BV method. In the first case, a simple test –if the corresponding bit of the uncompressed diagonal representation is set– is enough. In the second case, we iterate over the successors of a node using a *LazyIntIterator*, until we find the edge or run out of them.

For the latter query, we asked for the successors of all nodes of each graph in Table 4 and calculated the mean average time of the responses. The built-in *successors()* method of class *BVGraph* was used for the part of the graph that was compressed with the BV method. For the rest of the edges, i.e., the edges belonging in the compressed diagonal stripe, a list was populated by adding the successors whose corresponding bits in the original diagonal stripe are set.

We took into account the percentage of edges existing in the diagonal of each graph to the total edges of the graph (calculated using the data in Table 2), to estimate a median access time of operations *EdgeExists* and *Successors* in our experimental setup. This median access time combines the access times of operations *EdgeExists* and *Successors* by considering the probability of an edge existing either in the compressed diagonal stripe or in the rest of the graph, and is represented by BV+ (total) in Table 4. Of course, we also present the average time these queries needed when the full graph is compressed using the BV method.

The tests were applied to a web (`cnr-2000`), a road network (`roadNet-CA`), and a social network (`ljournal-2008`) graph. The k and b parameters of BV+ were the ones used in Table 2. We see that BV+ can answer both queries in constant time as far as the edges in the compressed diagonal are concerned. This time is much smaller than the time needed for the BV method to answer for the rest of the edges. In addition to this, we notice that for all graphs the BV method benefits from having to compress less edges. Unsurprisingly, the time needed to answer the two queries is larger when all the edges of the graph are compressed with the BV method. The BV+ method needs to address queries for both cases (edge lies inside or outside the compressed diagonal), but this does not impose an additional overhead to any non-single core environment, as the tasks are clearly separated. Thus, BV+ outperforms BV as far as access times are concerned for all the graphs tested. The better access times for edges belonging either to the diagonal stripe or to the rest of the graph reflect to BV+ (total). In particular, operations *EdgeExists* and *Successors* run faster with BV+ than with BV by 8.75% and 13.48% for graph `cnr-2000`, by 21.13% and 13.80% for graph `roadNet-CA`, and by 9.50% and 8.41% for graph `ljournal-2008` respectively.

As is the case with the compression ratio comparison that took place in Section 4.3, BV+ outperforms BV regarding access times too.

4.5 Effect of BV+ parameters

The results illustrated in Table 2 highlight among other things the important role parameters k and b play in obtaining a good compression ratio for a given graph. We remind the reader that parameter k determines the width of the diagonal stripe of a graph; the width is equal to $2k + 1$. For example, a 3-diagonal stripe of a particular graph is illustrated in Figure 2. Parameter b denotes the number of bits that comprise a row of the compressed diagonal stripe. In particular, b-digit binary numbers are used to represent the $2^b - 1$ numbers that are met most frequently among the rows of the diagonal stripe. Using Proposition 1, we can estimate how much better than the state-of-the-art-method we can represent the dense part of the graph for a given k, but the pair that produces the optimal result is highly dependent on the structure of the graph and parameter b, since there is a trade-off between keeping the ratio of the compressed diagonal stripe low and including as many edges as possible in it. For example, for the graph `roadNet-PA`, the best pair turned out to be $\{k = 7, b = 6\}$ which gives a ratio of 3.27 bits/edge, which is worse than the one given for $\{k = 7, b = 2\}$ (1.95 bits/edge), but includes almost twice as many edges.

The values of parameters k and b are fixed by performing a statical analysis per given graph prior to its compression. For the dataset that we have experimented with, we have found that a good selection of values for parameter k ranges from 2 to 20 for k, and b should be at most equal to k. However, for the sake of presenting the best possible results in this paper, we even went further and tested values that were outside the aforementioned ranges, to come up with a pair that results in the best compression ratio for each graph.

We can see that for graphs `dblp-2010` and `amazon-2008` the selected value of parameter k is outside the range $[2, 20]$, thus, seemingly unsettling our initial argument about having come up with a proper range of k. However, the fact is that we had obtained a very good compression ratio, very close to the one presented in Table 2, with a value of k between 2 and 20 for both of these graphs. In particular, for `dblp-2010` we achieved a compression ratio of 7.23 for $k = 16$ and $b = 6$, which is only slightly worse to the compression ratio of 7.20 for $k = 24$ and $b = 7$. And for `amazon-2008`, we achieved a compression ratio of 10.078 for $k = 20$ and $b = 15$, which is almost identical to the compression ratio of 10.074 for $k = 23$ and $b = 15$.

In any case, we chose to use the values of parameters k and b that provided us with the best compression ratio, since we felt that it was very important for us to present the best results obtained in the experimental evaluation of our method. This goes to say that we have identified a strong trend for the values of parameters k and b, but not a pattern, as there are times that our statical analysis proposes values out of the aforementioned range. The important fact to note is that by thoroughly testing several graphs for various values of k we observed that the compression using our algorithm, viz., BV+, is better than the compression using algorithm BV for all of the graphs tested.

5. DISCUSSION

We went beyond the state-of-the-art method of Boldi et al. for the compression of web graphs [5, 7] by exploiting the clustering properties observed in graphs that represent networks created by human activity, such as social networks and the worldwide web, in a way different than in [7]. In particular, we modified the way [7] represents a dense subgraph of such graphs, by exploiting their properties, namely, *locality of reference* and *similarity*. Our implementation is very abstract and straightforward and thus comprises a generic framework which is capable of employing different graph compression schemes. Experimental evaluation of our approach on a wide and well-studied dataset of graphs shows significant decrease of the graphs' compressed size, that reaches up to 21.7%. Moreover our approach provides up to 21.13% faster access on the graphs' elements. In the scope of this work, we thoroughly investigated the research activity on compressed data structures for graphs, and presented here the most eminent of those approaches that managed to introduce significant advances in the field of graph compression.

Choosing an appropriate set for representing values of the diagonal stripe may become more effective by using a heuristic different than promoting the ones with a high frequency and a significant amount of edges. Future work consists of exploring such heuristics for an even better compression of the graph. Further, we would like to investigate other reordering algorithms that could have a positive impact on our approach.

Acknowledgments

We are grateful to Elias Koutsoupias, Nikos Leonardos and Alex Delis for fruitful discussions on aspects of this work.

6. REFERENCES

[1] M. Adler and M. Mitzenmacher. Towards Compressing Web Graphs. In *DCC*, 2001.

[2] A. Apostolico and G. Drovandi. Graph Compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[3] Y. Asano, Y. Miyawaki, and T. Nishizeki. Efficient Compression of Web Graphs. In *COCOON*, 2008.

[4] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: fast access to linkage information on the Web. In *WWW*, 1998.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *WWW*, 2011.

[6] P. Boldi, M. Santini, and S. Vigna. Permuting Web and Social Graphs. *Internet Mathematics*, 6(3):257–283, 2009.

[7] P. Boldi and S. Vigna. The WebGraph Framework I: Compression Techniques. In *WWW*, 2004.

[8] N. R. Brisaboa, S. Ladra, and G. Navarro. k2-Trees for Compact Web Graph Representation. In *SPIRE*, 2009.

[9] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the Web. *Computer Networks*, 33(1-6):309–320, 2000.

[10] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, 2008.

[11] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, 2009.

[12] F. Claude and S. Ladra. Practical Representations for Web and Social Graphs. In *CIKM*, 2011.

[13] F. Claude and G. Navarro. A Fast and Compact Web Graph Representation. In *SPIRE*, 2007.

[14] C. I. Del Genio, T. Gross, and K. E. Bassler. All Scale-Free Networks Are Sparse. *Phys. Rev. Lett.*, 107:178701, 2011.

[15] C. Hernández and G. Navarro. Compressed representation of web and social networks via dense subgraphs. In *SPIRE*, 2012.

[16] U. Kang and C. Faloutsos. Beyond "caveman communities": Hubs and spokes for graph compression and mining. In *ICDM*, 2011.

[17] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, 2010.

[18] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, 2005.

[19] P. Liakos, K. Papakonstantinopoulou, and M. Sioutis. On the Effect of Locality in Compressing Social Networks. In *ECIR*, 2014.

[20] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *KDD*, 2010.

[21] S. Raghavan and H. Garcia-Molina. Representing Web Graphs. In *ICDE*, 2003.

[22] K. H. Randall, R. Stata, J. L. Wiener, and R. G. Wickremesinghe. The Link Database: Fast Access to Graphs of the Web. In *DCC*, 2002.

[23] I. Safro and B. Temkin. Multiscale approach for the network compression-friendly ordering. *J. of Discrete Algorithms*, 9(2):190–202, 2011.

[24] C. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 1948.

[25] J. Síma and S. E. Schaeffer. On the NP-Completeness of Some Graph Cluster Measures. In *SOFSEM*, 2006.

[26] T. Suel and J. Yuan. Compressing the Graph Structure of the Web. In *DCC*, 2001.

[27] I. H. Witten, T. C. Bell, and A. Moffat. *Managing Gigabytes: Compressing and Indexing Documents and Images*. John Wiley & Sons, Inc., 1st edition, 1994.