# Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations

Towards Understanding of Modern Graph Processing and Storage

MACIEJ BESTA, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Various graphs such as web or social networks may contain up to trillions of edges. Compressing such datasets can accelerate graph processing by reducing the amount of I/O accesses and the pressure on the memory subsystem. Yet, selecting a proper compression method is challenging as there exist a plethora of techniques, algorithms, domains, and approaches in compressing graphs. To facilitate this, we present a survey and taxonomy on *lossless* graph compression that is the first, to the best of our knowledge, to *exhaustively* analyze this domain. Moreover, our survey does not only categorize existing schemes, but also explains *key ideas*, discusses formal underpinning in selected works, and describes the space of the existing compression schemes using three dimensions: *areas of research* (e.g., compressing web graphs), *techniques* (e.g., gap encoding), and *features* (e.g., whether or not a given scheme targets dynamic graphs). Our survey can be used as a guide to select the best lossless compression scheme in a given setting.

CCS Concepts: • **Information systems → Data compression**;

**ACM Reference format:**
Maciej Besta and Torsten Hoefler. 2018. Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations. 52 pages.

## 1 INTRODUCTION

Big graphs form the basis of many problems in machine learning, social network analysis, and various computational sciences [296]. Storage-efficient processing of such graphs is becoming increasingly important for HPC and Big Data. First, it may eliminate expensive I/O accesses. Moreover, it enables storing a larger fraction of data in caches, potentially increasing performance. Next, it could eliminate inter-node communication as graphs may fit in the memory of one node. Finally, reducing required amounts of memory lowers costs of necessary hardware.

There exists a plethora of graph compression schemes. They cover various fields such as web graphs [62], biology networks [125, 201], or social graphs [106]. Moreover, they follow different methodologies, for example attempting to build a graph representation that asymptotically matches the storage lower bound (so called *succinct* representations [155]) or permuting vertex integer labels to minimize the sum of differences between consecutive neighbors of each vertex and encoding these minimized differences with variable-length coding [61]. Next, there are different compression techniques that can be used in the context of any methodologies, for example reference encoding [62], Huffman degree encoding [400], and many others. Finally, compression can be general and target any graph [155] or may be designed for a particular class of graphs [55].

This paper aims to provide the first taxonomy and survey that attempts to cover all the associated areas of lossless graph compression. Our goal is to (1) *exhaustively* describe related work, (2) illustrate and explain the *key ideas* and the theoretical underpinning, and (3) *systematically* categorize existing algorithms, schemes, techniques, methodologies, and concepts.

**What Is The Scope of This Survey?** We focus on *lossless* approaches and leave all the *lossy* schemes for future work. The main reason for this is the fact that the scope of lossless graph compression is on its own very extensive, covering almost 450 papers and various approaches, numerous techniques, countless algorithms, and a large body of applications.

**What Is the Scope of Existing Surveys?** Existing surveys on graph compression cover only a small part of the domain. Zhou provides a brief survey [440] with 14 references. Maneth and Peternek [302] only cover a part of succint representations, RDF graph compression, and a few works categorized under the common name "Structural Approaches". Finally, there are other surveys that describe fields only partially related to graph compression: compressing polygonal meshes [300], summarizing graphs [291], and compressing biological data [213]. We discuss these surveys in more detail in the related parts of this work.

## 2 BACKGROUND

We first present concepts used in all the sections and summarize the key symbols in Table 1.

| | |
|---|---|
| $G$, $\mathbf{A}$ | A graph $G = (V, E)$ and its adjacency matrix; $V$ and $E$ are sets of vertices and edges. |
| $n$, $m$ | Numbers of vertices and edges in $G$; $\|V\| = n$, $\|E\| = m$. |
| $d$, $\hat{d}$, $D$ | Average degree, maximum degree, and the diameter of $G$, respectively. |
| $d_v$, $N_v$ | The degree and the sequence of neighbors of a vertex $v$. |
| $\mathscr{O}$, $\mathscr{A}$, $\mathscr{A}_v$ | Data structures for, respectively, the pointers to the adjacency data of each vertex, the adjacency data of a given graph $G$, and the adjacency data of vertex $v$. |
| $N_{in,v}$, $N_{out,v}$ | The in-neighbors and out-neighbors of a vertex $v$. |
| $N_{i,v}$ | The $i$th neighbor of $v$ (in the order of increasing labels). |

Table 1. The most important symbols used in the paper.

### 2.1 Graphs

We model an undirected graph $G$ as a tuple $(V, E)$ (denoted also as $G(V, E)$); $V$ is a set of vertices and $E \subseteq V \times V$ is a set of edges; $|V| = n$ and $|E| = m$. If $G$ is directed, we use the name *arc* to refer to an edge with a specified direction. We consider both labeled and unlabeled graphs. If a graph is labeled, $V = \{1, ..., n\}$, unless stated otherwise. $N_v$ and $d_v$ are the neighbors and the degree of a vertex $v$. The $i$th neighbor of $v$ (in the order of increasing labels) is denoted as $N_{i,v}$; $N_{0,v} \equiv v$. We use the name "label" or "ID" interchangeably. We denote the maximum degrees for a given $G$ as $\hat{d}$, $\hat{d}_{in}$ (in-degree), and $\hat{d}_{out}$ (out-degree). $G$'s diameter is $D$.

We also consider more complex graph properties: arboricity and genus. Arboricity is a minimum number of spanning forests that cover all the edges of a given graph. Next, a graph $G$ has a genus $g \geq 0$ if it can be drawn without crossing itself (i.e., its edges) on the surface of a sphere that has $g$ handles (*such as in a coffee mug*). For example, a graph with $g = 1$ can be drawn on a torus [409].

### 2.2 Graph Representations

$G$ can be represented as an *adjacency matrix* (AM) or *adjacency lists* (AL). An AL consists of a contiguous array with the adjacency data (denoted as $\mathscr{A}$) and a structure with offsets to the neighbors of each vertex (denoted as $\mathscr{O}$). AL uses $O(n \log m + m \log n)$ bits while AM uses $O\left(n^2\right)$.

AL needs $O\left(\hat{d}\right)$ time to check if two vertices are connected while obtaining $N_v$ or $d_v$ takes $O(1)$ time. For AM, it takes $O(1)$ to verify if two vertices are connected and $O\left(\hat{d}\right)$ to obtain $N_v$ and $d_v$.

### 2.3 Graph Families

We next describe families of graphs used in the following sections; we provide them to make our work self-contained. For simplicity, we focus on intuitive descriptions instead of formal definitions. **General graphs** can have multiple edges between the same two vertices (i.e., multigraphs) and loops (a loop is an edge starting and ending at the same vertex). **Simple graphs** have no loops and multiple edges between the same pair of vertices. **Loop-free graphs** are general graphs without loops. **Stick-free graphs** do not have vertices of degree one. A **transposed graph** has reversed arc directions of the input directed graph [61]. A **graph embedding** into a certain surface is a drawing of a graph $G$ on that surface so that $G$'s edges may intersect only at their endpoints [52]. A **planar graph** can be drawn on the plane so that its edges intersect only at their endpoints. An **outerplanar graph** has a planar drawing such that all vertices belong to the drawing of the outer face of the graph (intuitively, all the vertices lie on the boundary between the graph and the outer part of the "plane" surrounding the graph). A **plane graph** (also **planar drawing** or **plane drawing**) is a **planar embedding** of a planar graph. A $k$-**page graph** $G$ has its edges partitioned into $k$ sets $E_1, ..., E_k$ so that, for each $i$, a graph $G(V, E_i)$ is planar; $G(V, E_i)$ is a graph with a vertex set $V$ and edge set $E_i$ [237]. In a drawing of a graph $G$ on $k$ pages each vertex of $G$ constitutes a point on a "spine" of the book formed by these $k$ pages and each edge of $G$ is drawn as a curve within a single page. A $k$-**page graph embedding** on a given surface is a $k$-page drawing of a graph $G$ on this surface without any edge crossings. A $k$-**connected graph** $G$ (also called $k$-vertex connected) is such a graph in which one cannot find a set of $k-1$ vertices whose removal disconnects $G$. A **map** is a topological equivalence class of planar embeddings of planar graphs. In encoding a map we are required to encode the topology of the embedding, i.e., incidences among faces, edges, and vertexes, as well as the graph. A map is an embedding of a unique graph, but a given graph may have more than one embedding. Thus, a map usually requires more bits to encode than a corresponding graph [243]. A **plane triangulation** is a plane graph, each of whose faces has size exactly three; a plane triangulation may contain self-loops and multiple edges. An **Erdős–Rényi graph** $\mathcal{G}(n, p)$ [150] is a graph with a uniform degree distribution where $n$ is the number of vertices and $p$ is a probability that an arbitrary edge is present in the graph (independently of other edges). A **separable graph** is a graph in which we can divide $V$ into two subsets of vertices of approximately the same size so that the size of a vertex cut between these two subsets is asymptotically smaller than $|V|$. Other considered graph classes are **graphs with bounded arboricity** (intuitively, they are *uniformly sparse*) and **graphs with bounded genus**.

### 2.4 Codes

Finally, we summarize codes for encoding integers that are used in various works to encode adjacency arrays of vertices. Elias $\gamma$ [148] is a universal code for positive integers. It is often used when the maximum possible number to be encoded cannot be determined beforehand; when using this code, the size of a value $x$ is $2\lceil \log x \rceil + 1$ [bits]. Elias $\delta$ [148] is a universal and asymptotically optimal code for positive integers. With this code, the size of a number $x$ is $\lceil \log x \rceil + 2\lceil \log \lceil \log x \rceil + 1 \rceil + 1$ [bits]. Golomb [187] is an optimal non-universal prefix code for alphabets following a geometric distribution. The code is suitable for sequences of integers where small values are much more likely to occur than large values. Gray code [194] is an arrangement (i.e., a total ordering) of numbers (or other entities such as vectors) such that the binary representations of consecutive numbers or any other entities differ by exactly one bit. $\zeta$ [64] is a code suited for integers that

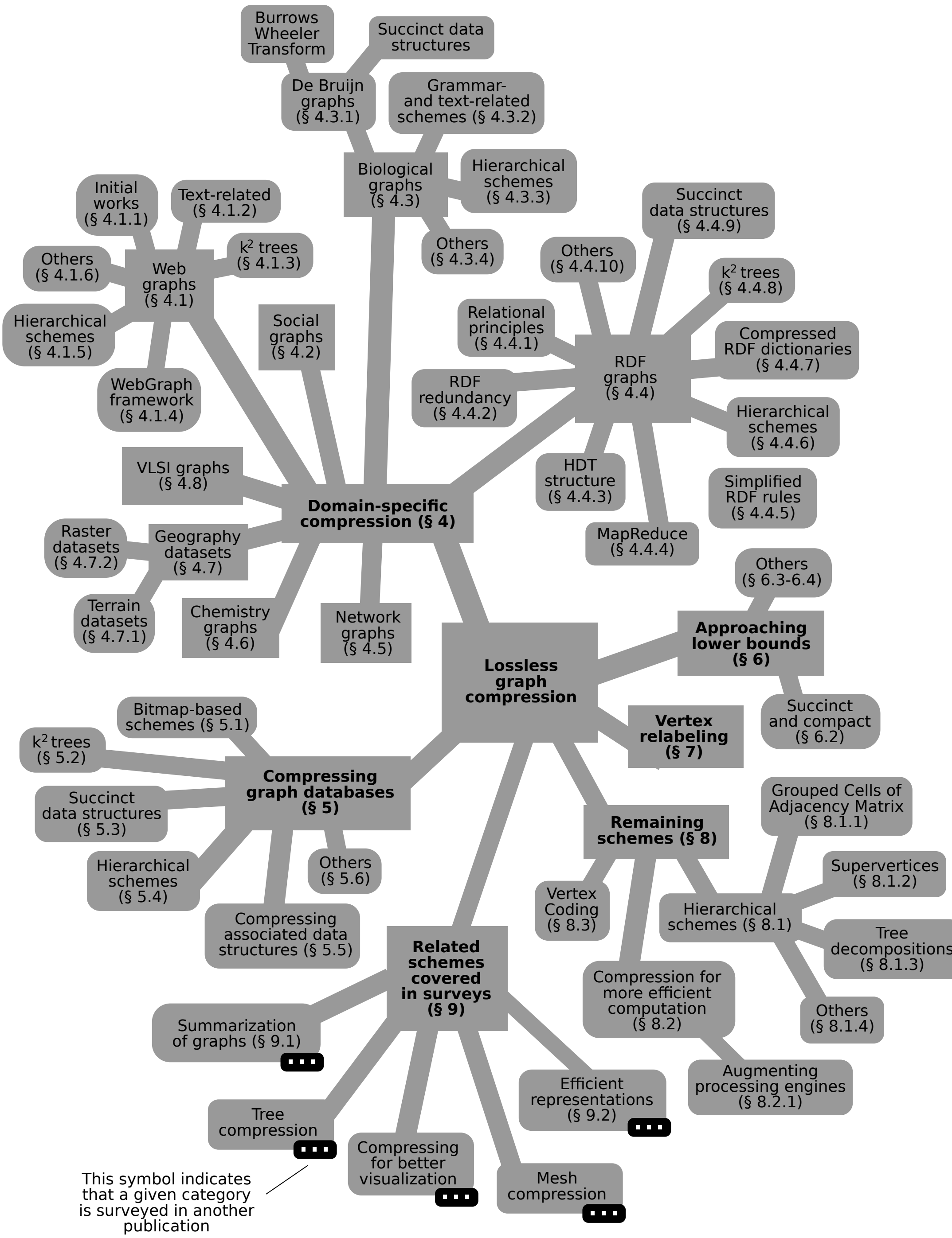


Fig. 1. (§ 3.1) The categorization of the considered domains of lossless graph compression.

follow the power law distribution with the exponent smaller than two. Finally, $\pi$ [25] is a universal code suited for integers that follow the power law distribution with the exponent close to one.

## 3 TAXONOMY AND DOMAIN DIMENSIONS

In this section, we describe how we categorize existing work in this survey. Figure 1 depicts the hierarchy of the considered domains.

### 3.1 How Do We Categorize Existing Work?

Graph compression is related to various areas such as databases or information theory. Schemes in these areas often share various common features, for example addressing static or dynamic graphs. This poses a question on how to categorize the rich world of graph compression studies to enable its systematic analysis. In the following, we dedicate a separate section (§ 4–§ 8) to one particular *area of research* such as compressing graph databases. Thus, we devote one section to the work done within one specific community. Such community-driven areas constitute *the first dimension* of our categorization. We describe how we identify these areas in § 3.1.1. Second, different areas of graph compression may use the same *techniques* for compressing graphs, or various techniques may be used in one publication or algorithm. For example, it is common to combine *gap encoding* and *variable-length codes* to compress web graphs. *The second dimension* in our categorization are thus *techniques* that reduce the size of graphs. We dedicate § 3.1.2 to describe the most important techniques. Finally, applications of a given technique within a certain area may have different *features*. For example, one can use a specific technique for either static or dynamic graphs. Consequently, the third dimension are features; we describe them in § 3.1.3.

We also discuss the existing categorizations and taxonomies in § 3.2.

*3.1.1 Areas.* Many papers are dedicated to compressing graphs from specific domains such as web graphs, biological networks, social graphs, and others; we describe them in § 4. Second, we describe works related to compressing graph databases (§ 5). Next, various schemes that we list in § 6 are devoted to approaching the storage lower bounds while ensuring fast (ideally constant time) queries. Moreover, we discuss optimization approaches to improve *graph layouts*, which ultimately reduces space occupied by a graph. Finally, we devote a separate section for various works that cannot be categorized in one of the above (see § 8) and to areas related to graph compression and covered in other surveys (see § 9).

*3.1.2 Techniques.* We now briefly present several common techniques used in various areas, as well as examples of their usage, to improve the clarity of the survey.

**Variable-Length Encoding** In this technique, vertex IDs stored in the adjacency array are encoded with one of the selected variable-length codes such as Varint.

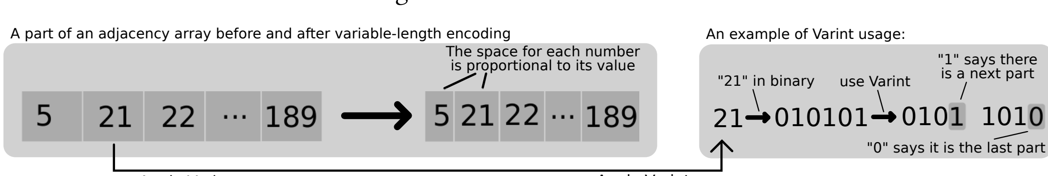

Fig. 2. An example of variable-length encoding.

**Vertex Relabeling** The main idea is to change the initial IDs of vertices so that the new IDs, when stored, use less space. We also use the name *vertex permutations* to refer to this technique. This scheme is usually combined with variable-length encoding.

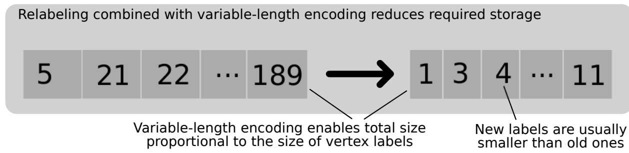


Fig. 3. An example of vertex relabeling combined with variable-length encoding.

**Reference Encoding** Here, identical sequences of vertices in the adjacency arrays of different vertices are identified. Then, all such sequences (except for a selected one) are encoded with references [5, 367]. One can implement reference encoding with *copy lists*: sequences of 1s and 0s that indicate whether or not a given number is retained in the current adjacency array.

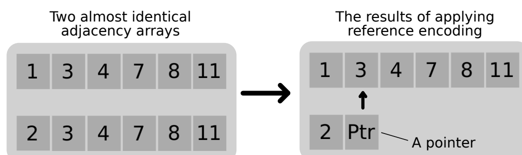


Fig. 4. An example of reference encoding (no copy lists).

**Run-length Encoding** This scheme enhances copy lists in reference encoding. The key idea is to provide the size of consecutive sequences of 1s and 0s instead of the actual 1 and 0 values [28].

**Huffman Degree Encoding** The core idea in this scheme is to use fewer bits to encode vertex IDs of higher degrees. Thus, $|\mathscr{A}|$ is reduced as vertex IDs that occur more often use fewer bits.

**Log Encoding** This scheme uses $\lceil \log n \rceil$ bits to encode each vertex ID in a graph with $n$ vertices.

**Interval Encoding** Here, consecutive vertex IDs (e.g., $x, x+1, ..., x+k$) are stored using the interval boundaries $x$ and $x+k$.

**Gap Encoding** This scheme preserves differences between vertex IDs rather than the IDs themselves. The motivation is that, in most cases, differences occupy less space than IDs. Several variants can be used here; the most popular is storing differences between the IDs of the consecutive neighbors of each vertex $v$, for example $N_1(v)-v, N_2(v)-N_1(v), ..., N_{d_v-1}(v)-N_{d_v-2}(v), N_{d_v}(v)-N_{d_v-1}(v)$ (the first of the above differences is sometimes called an *initial distance* and each following: an *increment*). Assuming each $\mathscr{A}_v$ is sorted, one must use an additional bit to indicate the sign of the first difference. Another variant stores the differences between $v$ and each of its neighbors: $N_1(v)-v, N_2(v)-v, ..., N_{d_v-1}(v)-v, N_{d_v}(v)-v$.

There are many more techniques used in various areas of graph compression. We defer their description to the relevent parts of the survey. These are, among others, $k^2$-trees and their variants, hierarchical schemes based on the concept of supervertices and superedges, and schemes that reorder the rows or columns of the adjacency matrix of a graph. Some techniques are used in various areas but they are themselves actively developed and they constitute a separate area of graph compression. Examples are succinct data structures or schemes that relabel vertices.

*3.1.3 Features.* We briefly present several features used in various compression areas to improve the clarity of the survey. We later (§ 10) discuss selected features in more detail.

- **Graph Dynamicity** This feature indicates whether a graph to be compressed is assumed static or dynamic (and thus allowing for any changes to its structure or labels).
- **Problem-Awareness** This feature determines whether a given compression schemes is tuned to some specific algorithm or graph problem to be solved over the compressed graph.
- **Graph-Awareness** Here, we determine whether a given scheme is tuned (or designed) for some specific graph classes or whether it works for generic graphs.

- **Streaming Graphs** This feature indicates whether a given scheme addresses graphs that are processed as a stream of edges.

### 3.2 Existing Categorizations

We also survey the existing categorizations of graph compression schemes. First, Boldi et al. [61] indicate that permutations that relabel vertices can be *intrinsic* (also called *coordinate-free*) or *extrinsic*. The former relabel IDs basing only on the graph structure (i.e., vertices and edges). The latter also rely on some additional information such as URLs. Next, Dhulipala et al. [143] identify three categories of schemes for graph (and index) compression. First, there are *structural* approaches that find and merge repeating graph patterns (such as cliques). Second, there exist schemes for encoding adjacency data represented by a sequence of integers. Finally, various works propose vertex relabelings (label permutations) that minimize a given metrics, for example the sum of differences between consecutive neighbors in each adjacency list. Another study [335] distinguishes between *structural* approaches and the ones based on using the notion of entropy [387]. Finally, schemes for compressing web graphs use the notions of *locality* and *similarity* [58]. Locality means that most links from page $x$ point to pages on the same host (that often share a long path prefix with $x$). Similarity means that pages from the same host have many links in common.

## 4 COMPRESSING GRAPHS IN SPECIFIC DOMAINS

A large portion of research in graph compression is dedicated to compressing graphs in some specific domains. We now present the related efforts.

### 4.1 Web Graphs

We start with web graphs.

*4.1.1 Initial Works.* The first works on web graph compression often use various combinations of techniques such as Huffman degree encoding, log encoding, gap encoding, differential encoding, and various variable-length codes. This domain was opened by the work on the Connectivity Server [53], a system for storing the linkage information found by the AltaVista [393] search engine. Connectivity Server associates each URL with an integer, sorts these integers (in each adjacency data structure $\mathscr{A}_v$) according to the URL lexicographic order, and uses gap encoding on the integers.

Another early work analyzes the web graph structure [81] and also enhances the original compression scheme in the Connectivity Server. Then, Wickremesinghe et al. [423] uses Huffman codes to encode references to links in the Connectivity Server. Moreover, Adler and Mitzenmacher [5] propose to compress web graphs with a Huffman-based scheme applied to in-degrees. They also use reference encoding and log encoding.

Suel and Yuan [400] compress URLs using common prefixes. To compress links, they distinguish between global links connecting sites on different hosts and local links connecting sites on the same host. For global links, they first identify $p$ URLs with the highest in-degree, and encode links to these URLs with a Huffman code. For other global links, they use log encoding or encode the link with a Golomb code, depending on the out-degree of a given URL. Next, they identify two local link classes. For each host $h$, they determine the given number of most popular destinations for local links within $h$. Links to these pages are encoded with a Huffman code.

The Link Database [367] is a system for storing web graphs. It uses various techniques (delta codes, Huffman codes, Grey orderings, and nybble codes); the related analysis identifies two important web graph properties, namely *locality* and *similarity*. The Link Database compresses offset arrays $\mathscr{O}$ by using different bit counts to store offsets for different degree ranges. For example, it uses a 32-bit index to the start of a given range of vertices, and then only 8-bit offsets for each of the following vertices.

*4.1.2 Text-Related Works.* Next, we present efforts that revolve around treating the input graph $G$ as text and using the associated compression methods. Navarro [328] proposes to regard $G$ as text and to utilize existing techniques for text compression and indexing. Specifically, he uses the *Compressed Suffix Array (CSA)* [376] structure as a basis for his graph representation. CSA is a *compressed full-text self-index*: a data structure built over a text $T = t_1 \ldots t_n$ (over an alphabet $\Sigma$) that requires space proportional to the size of the compressed text and simultaneously enables accessing any substring of $T$ (including the whole $T$ itself). Thus, it becomes unnecessary to store $T$ and some search operations on it are enabled. Now, the key idea is to treat an input graph as $T$ and use various CSA's functionalities to enable accessing the graph efficiently without decompressing it.

Claude and Navarro [117] use Re-Pair [270]: a phrase-based compressor that permits fast and local (i.e., without having to access the whole graph) decompression. Re-Pair repeatedly finds the most frequent pairs of symbols in a sequence of integers and replaces them with new symbols, until no more replacements reduce storage. An example is in Figure 5.

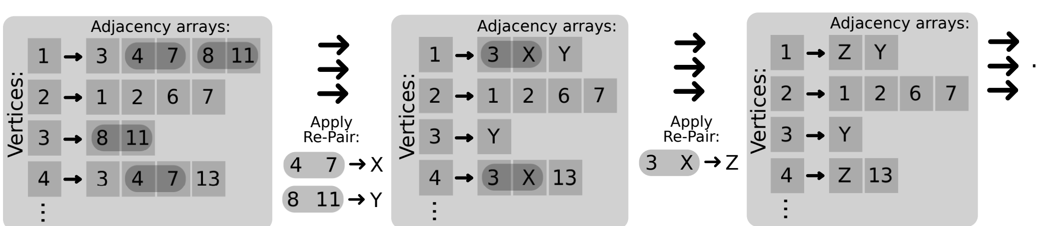


Fig. 5. Several example iterations of the Re-Pair scheme.

The approach based on Re-Pair was further extended by Claude and Navarro [118]. They combine it with an approach based on perceiving $G$ as a binary relation on $V \times V$ and use several techniques developed specifically for answering queries over binary relations [40, 41]. The motivation for such a combination is to derive *both* $N_{out,v}$ *and* $N_{in,v}$ fast.

*4.1.3 $k^2$ Trees.* We now describe efforts related to so called $k^2$ trees. In their seminal work, Brisaboa et al. [78, 188] present a Web graph representation that uses a tree structure that takes advantage of the structure of the adjacency matrix $A$ of web graphs. Specifically, it uses the sparseness and clustering properties of $A$. An example is presented in Figure 6. Initially, $A$ is divided into $k^2$ submatrices of the same size ($k$ is a parameter); these submatrices are recursively divided in the same way. Now, the key idea is to represent $A$ as a $k^2$-ary tree (called a $k^2$ tree) that corresponds to the above recursive "partitioning" of $A$. Each tree node contains a single bit of data. Each internal tree node has $k^2$ children. At each partitioning level, if a given submatrix to be partitioned contains only 0s, the corresponding tree node contains 0. Otherwise, it contains a 1. The resulting tree is encoded using a special simplified tree encoding that ensures asymptotically low compression ratio [365]. The $k^2$ representation ensures obtaining $N_{out,v}$ and $N_{in,v}$ fast; it also enables extended functionality not usually considered in compressed graph representations.

Claude and Ladra [116] further extended the $k^2$ tree idea by combining it with the Re-Pair scheme. Specifically, they first split the graph into subgraphs that correspond to different web domains. After that, the key idea is to encode each subgraph with a $k^2$ tree representation and encode the remaining inter-subgraph edges with Re-Pair.

A study on the best data distribution for query processing on graphs compressed with $k^2$ trees can be found in the work by Alvarez et al. [21].

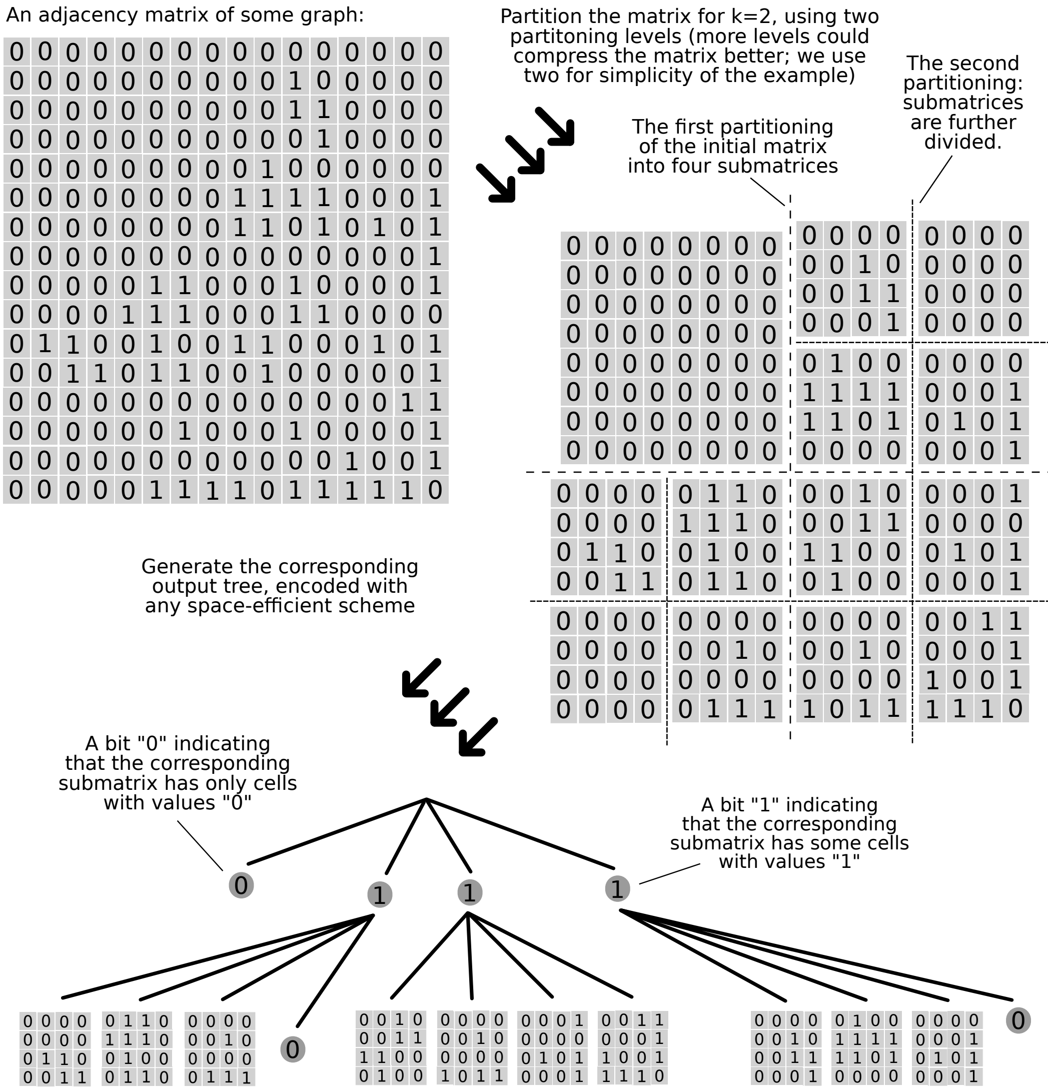


Fig. 6. An example application of the $k^2$ tree scheme.

The work on $k^2$ trees was also extended by Brisaboa et al. [79] to ensure obtaining some properties of the graph structure fast. For example, they accelerate finding the successors of a certain web page that are within a given range of web pages. Moreover, additional enhancements ensure better compression ratios. First, they vary $k$ across the tree levels: they use a larger $k$ for higher levels of the tree and a smaller $k$ for the last tree levels. This enhancement uses the observation that lower tree levels require fewer bits to cover 1s (i.e., the adjacency matrix in the considered graphs is usually very sparse and the existing 1s can be covered by square submatrices of small dimensions, thus the values of $k$ can be low for the last levels of respective $k^2$ trees). Second, they use multiple $k^2$ trees for one adjacency matrix by partitioning the matrix and building one tree per partition independently of others. This decreases construction times and enables better adjustment between the granularity of the tree ($k$) and the submatrix sparsity. Third, the last level is additionally compressed with a variant of Huffman encoding: respective submatrices are sorted by frequency and are assigned numerical values that identify (and point to) each submatrix; lower numbers are assigned to more frequent submatrices to lower the total number of bits used to represent the matrices.

*4.1.4 WebGraph Framework.* We devote a separate subsection to describe the efforts related to the WebGraph Framework, a mature framework targeted at compressing web graphs.

Boldi and Vigna provide several works on compressing Web graphs. First, they developed the WebGraph framework [62] that is a freely available suite of codes, algorithms, and tools for compressing graphs with a special focus on web graphs. In addition to exploiting the two well-known web graph properties (locality and similarity) they also identify three further properties: (1) "similarity concentration", i.e., either two adjacency lists have almost nothing in common, or they are characterized by large overlap, (2) "consecutivity is common", i.e., many links within a page are consecutive with respect to the lexicographic order, and (3) "consecutivity is the dual of distance-one similarity", i.e., if there is a lot of similarity between vertices and their respective successor lists in lexicographical ordering, the transpose of the given graph must contain large intervals of consecutive links. Now, WebGraph uses the following compression techniques: (1) gap encoding, (2) reference encoding, (3) differential encoding, and (4) interval encoding. Reference encoding may have an arbitrary number of levels, bounded by a user parameter. Another parameter controls the minimum number of integers that enables a given sequence of numbers to be taken into consideration when applying reference encoding. Finally, for high performance, adjacency lists are build *lazily* (i.e., the data available through reference encoding is only fetched when required).

Moreover, Boldi et al. [60, 61] test several existing vertex permutations. They also propose two new permutations based on the Gray [258] ordering. The key idea is to permute the rows of the adjacency matrix so that the adjacent rows change according to the Grey code. Moreover, they analyze transposed graphs and illustrate that coordinate-free permutations improve compression rates (coordinate-free permutations are permutations that achieve almost the same compression performances regardless of the initial ordering). They conclude that the Grey ordering may improve the compression rates and that coordinate-free orderings are particularly efficient for transposed graphs (in some cases reaching the level of 1 bit per edge).

Boldi et al. [57] conduct an analysis that aims to formally understand why the existing approaches compress web graphs well. For this, they first observe that it is important for high compression ratios to use an ordering of vertex IDs such that the vertices from the same host are close to one another. To understand this notion more formally, they propose *measures* of how well a given vertex ordering $\pi$ respects the partitioning $\mathscr{H}$ induced by the hosts. Their first measure is the probability to experience a host transition (HT), i.e., a fraction of vertices that are followed (in the $\pi$ ordering) by another vertex at a different host:

$$HT(\mathscr{H}, \pi) = \frac{1}{n} \cdot \sum_{i=1}^{n-1} \delta\left(\mathscr{H}\left[\pi^{-1}(i)\right], \mathscr{H}\left[\pi^{-1}(i-1)\right]\right) \quad (1)$$

where $\delta$ is the usual Kronecker's delta and $\mathscr{H}[x]$ denotes the equivalence class of a vertex $x$: the set of vertices with the same host as $x$.

Moreover, the second used measure is adapted from work by Meilă [315] and is called the Variation of Information (VI); it enables comparing two partitions $\mathscr{H}$ and $\mathscr{H}_\pi$ that are associated with the original vertex ordering and the $\pi$ ordering. First, VI uses a notion of entropy $H(\mathscr{P})$ associated with a host partitioning $\mathscr{P}$:

$$H(\mathscr{P}) = -\sum_{S \in \mathscr{P}} P(S) \log(P(S)) \quad (2)$$

where $P(S) = \frac{|S|}{n}$. Then, the *mutual information between two partitions* is defined as

$$I(\mathscr{P}, \mathscr{T}) = \sum_{S \in \mathscr{P}} \sum_{T \in \mathscr{T}} P(S, T) \log \frac{P(S, T)}{P(S)P(T)} \tag{3}$$

where $P(S, T) = \frac{|S \cap T|}{n}$. Finally, the VI measure is then defined as

$$VI(\mathscr{P}, \mathscr{T}) = H(\mathscr{P}) + H(\mathscr{T}) - 2I(\mathscr{P}, \mathscr{T}) \tag{4}$$

Now, substituting $\mathscr{P}$ and $\mathscr{T}$ with $\mathscr{H}$ and $\mathscr{H}_\pi$ and observing that $I(\mathscr{H}, \mathscr{H}_\pi) = H(\mathscr{H})$ gives

$$VI(\mathscr{H}, \mathscr{H}_\pi) = H(\mathscr{H}_\pi) - H(\mathscr{H}) \tag{5}$$

the authors use the HT and VI measures to compare various vertex orderings used for web graphs. In the second part of the paper, they show that their ordering called Layered Label Propagation (explained in more detail in § 4.2 dedicated to social networks as it was introduced mostly in the context of social networks) outperforms other orderings proposed in the literature.

Other works due to Boldi, Vigna, and others include developing complete instantaneous $\zeta$ codes for integers distributed as a power law with the exponent smaller than two [64], and presenting how to implement WebGraph with Java [63]. Some benchmarks about Web Graph can also be found in several empirical analyses [181, 271].

*4.1.5 Hierarchical Schemes.* We now survey methods where a given part of the input graph (e.g., a clique or a subgraph) is collapsed into a smaller entity (e.g., a vertex) to ultimately reduce space.

Raghavan and Garcia-Molina [363] propose a representation called S-Node. They find and collapse subgraphs into supervertices and then use Huffman encoding on in-degrees as well as reference encoding. Edges between supervertices are merged and form superedges, with information on which single edges should be added or removed to restore the original graph structure. To further enhance the representation, they incorporate various types of graph partitioning and they utilize contiguous vertex IDs with respect to their orderings in supervertices.

Buehrer and Chellapilla [84] generate *virtual nodes* from frequent itemsets in the adjacency data (i.e., dense subgraphs are replaced with sparse ones). For example, consider a fully connected directed bipartite subgraph where $n_1$ and $n_2$ denote the numbers of vertices in each of the two color classes; all the edges are directed towards one color class. The proposed scheme collapses the edges in such a subgraph by introducing a virtual node $v$. Now, $n_1$ edges from one class point to $v$ and $n_2$ edges point from $v$ to each vertex in the other color class. This reduces the edge count from $n_1 n_2$ to $n_1 + n_2$ and is able to achieve a high compression ratio as bipartite subcliques are frequent in web graphs [264]. In this work, the complexity of the mining phase (i.e., where itemsets are found) is bounded to $O(m \log m)$. To ensure this complexity they first cluster similar vertices in the graph (vertices are similar if a significant portion of their outlinks point to the same neighbors). Next, they find patterns in the clusters, remove the patterns, and replace these patterns with virtual nodes. Buehrer and Chellapilla's work was extended by Mondal [320] by providing insights into which values of respective algorithm parameters are best suited in a given scenario.

Karande et al. [241] complement the work of Buehrer and Chellapilla [84] by showing that various web graph algorithms can be extended to run on graphs compressed with virtual nodes so that their running times depend on the size of the compressed graph instead of the original $G$. They consider algorithms for link analysis, estimating the size of vertex neighborhoods, and various schemes based on matrix–vector products and random walks (PageRank [73], HITS [255], and SALSA [274]). For example, they show how to transform PageRank results for a compressed graph to obtain the PageRank results for the original graph.

Khalili et al. [245] relabel vertices so that similar vertices have closer IDs. Second, they group similar vertices together and collapse edges between groups into single *superedges*. To keep track of the collapsed edges they use an auxiliary data structure.

Anh and Moffat propose a hierarchical scheme for compressing web graphs [24]. The key idea is to partition the adjacency arrays into groups of $h$ consecutive arrays. Then, sequences of consecutive integers in each of the $h$ arrays are replaced with new symbols to reduce $|\mathscr{A}|$; this can be seen as grammar-based compression conducted in a local manner [192].

Grabowski and Bieniecki [191, 192] present two schemes; the first one offers higher compression ratios while the second one is faster. The first scheme extends the reference encoding from the WebGraph framework. Among others, it extends the binary format of reference encoding by using more than two values to indicate more possible referenced values. Another one creates blocks of $h$ adjacency lists. Then, within each such block, $h$ adjacency lists are merged and then all duplicate numbers in each merged sequence of lists are removed. Simultaneously, every number receives an associated list that indicates which neighborhoods it originally belongs to.

Hernandez and Navarro [206] combine several techniques to accelerate obtaining various graph properties and to further reduce the required storage. First, they combine reducing the number of graph edges (through virtual nodes [84]) with vertex reordering [25, 61, 62] to better exploit ordering, locality, similarity, and interval/integer encoding. Next, they combine $k^2$ trees [78] with virtual nodes [84] as well as several other techniques, including Re-Pair [117]. They address both web and social graphs. They first analyze various existing compression methods for web graphs and show that a combination of schemes that collapse edges and reorder vertices can offer outstanding compression ratios and performance of graph accesses. Second, they propose a novel compression method that uses compact (as defined in § 6) data structures to represent social communities.

In other studies, Hernandez and Navarro [207, 208] propose to find dense subgraphs and represent $G$ as the set of dense subgraphs $\mathcal{H}$ plus a remaining graph $\mathcal{R}$. For this, they first investigate the notion of dense subgraphs and define dense subgraphs to be pairs $(S, C)$ of subsets of vertices, such that every vertex in $S$ points to every vertex in $C$, but with $S$ and $C$ not necessarily disjoint. Thus, the case where $S = C$ corresponds to cliques and the case where $S \cap C = \emptyset$ corresponds to bicliques. The authors show that subgraphs where $S \neq C$ (without being disjoint) occur often in web (and social) graphs and designing a compressed representation for them pays off. Second, they store $\mathcal{H}$ using a combination of integer sequences and bitmaps. To store $\mathcal{R}$, they use several existing techniques such as $k^2$ trees [78, 79, 268], WebGraph schemes with the Layered Label Propagation ordering [57, 62], and $k^2$ partitioning [116].

Maneth and Peternek [303, 304] recently proposed a scheme that recursively detects repeated substructures and represents them using grammar rules. They show that some queries such as reachability between two nodes can be evaluated in linear time over the grammar, enabling speedups proportional to the compression ratio. The key idea, similarly to Claude and Navarro [117], is to use Re-Pair [270]: a phrase-based compressor that permits fast and local decompression. However, contrarily to Navarro's work, they do not use Re-Pair over a string built from the adjacency list, but instead represent frequent substructures with grammar rules.

*4.1.6 Others.* Guillaume et al. [197] aim to efficiently encode large sets of URLs using only widely available tools, namely `gzip` and `bzip` [141]. They also aim to make the mapping between URLs and their identifiers as fast as possible, and to compute $N_{out,v}$ efficiently. To avoid decompressing the entire list of URLs, they split the sequence to be compressed into blocks and compress each block independently. They utilize gap encoding, focusing on differences between a given vertex and each of its neighbors to derive the *length* of each link. They also observe that such link lengths follow a power distribution. Each length of a link is represented in either a Huffman code, 16-bit integer, or 32-bit integer according to its absolute value.

Asano et al. [28] encode integers which have a power distribution with a generalization of the variable-length nybble code. They use Kraft's inequality [262] about instantaneous codes to show that, when a random variable $X$ has a probability function $f(X)$, the instantaneous code which minimizes average codeword length when used to represent $X$ is $\log f(x)$ bits long when encoding $x$. Thus, if $X$ follows the power distribution with the exponent $-\alpha$, the instantaneous code minimizing the average codeword length is the variable-length block code with $\frac{1}{\alpha-1}$-bit blocks. Next, they show that, when each $\mathscr{A}_v$ is gap encoded, the first numbers in each $\mathscr{A}_v$ and the accompanying increments follows power distributions of different exponents. They use this to develop a new encoding of the web graph. Consider the $\mathscr{A}_v$ of any $v$. Suppose $v$ has out-degree $d$. Then $\mathscr{A}_v$ has one initial distance and $d-1$ increments. Consecutive 1s in the list of increments are compressed using the run-length encoding [359]. Finally, the initial distance, the increments, and the run-length codes are represented in the variable-length block codes with 6-bit, 3-bit, and 1-bit blocks, respectively.

The main idea due to Asano et al. [29] is to identify identical blocks in the adjacency matrix $A$ and then represent $A$ with a sequence of blocks combined with some metadata information on the block type and others. Now, they propose to use six different types of such blocks that correspond to different types of locality within each host in the input web graph. For a thorough analysis, they provide a detailed classification and an extensive discussion on the proposed locality (and thus block) types. Inter-host links are treated as related to a special type of locality.

Apostolico and Drovandi target both web graphs and more general graphs [25]. Instead of naming vertices based on the lexicographical ordering of their URLs (and thus being tailored for web graphs only), they conduct a breadth-first search traversal of the input graph and assign an integer to each vertex according to the order in which it is visited. This ensures significant storage reductions after gap encoding is applied. They also introduce a new class of $\pi$-codes: universal codes for integers that follow power law distribution with an exponent close to one.

Dhulipala et al. [143] extend the work due to Chierichetti et al. [106] and show how it can be employed for compression-friendly vertex relabeling in social networks and web graphs. They first note that optimal compression-friendly relabeling of vertices is NP-hard. Their key idea is to reduce the size of the problem domain. For this, they recursively bisect the graph and, once the size of the partitions is small enough, compute a selected (possibly optimal) reordering for each partition. Finally, these partial results are combined to obtain the solution for the whole graph.

Analysis of the impact of various coding schemes on the compressibility of link graphs was done by Hannah et al. [199]. Breyer [72] presents the MarkovPR software that optimizes storing URLs in web graphs with a large trie and hashtables that alleviate navigating in the trie. Finally, W-tree [33] is a space-efficient representation for web graphs optimized for external memory settings.

### 4.2 Social Networks

Several recent works aim to specifically condense social networks. Some offer novel schemes while others investigate how to reuse the schemes developed for web graphs.

Chierichetti et al. [106] provide three contributions targeted at social network compression. First, they prove hardness results about several types of vertex reordering; we provide more details in a section devoted to vertex relabeling (§ 7). Second, they propose the BL compression scheme that extends Boldi and Vigna's BV scheme from the WebGraph framework. BL takes advantage of a certain property common in social networks, namely *reciprocity*, next to the properties of locality and similarity. Reciprocity means that most unidirectional links are reciprocal, i.e., there is a link with a reverse direction connecting the same vertices. Third, the final design contribution is the *shingle* ordering that preserves both locality and similarity. Intuitively, it treats $N_{out,v}$ as a set and derives a special value called the *shingle* $M_\sigma(N_{out,v})$ of this set where $\sigma$ is a suitably selected permutation (or hash function). Then, the vertices of the input graph are ordered by their shingles. The authors show that, if two vertices have many outneighbors in common, then with high probability they will have an identical shingle and they will be close in the shingle ordering.

Maserrat and Pei [311, 312] aim to answer both $N_{in,v}$ and $N_{out,v}$ in sublinear time in $n$ and $m$ while compressing the input graph. For this, they propose an *Eulerian data structure*: a structure that stores a linearization of the input graph in a space-efficient way and uses it to answer the neighborhood queries efficiently.

Boldi et al. [57] propose Layered Label Propagation (LLP), a compression-friendly vertex ordering targeting social networks. They start their work with an analysis that aims to formally understand why the existing approaches compress web graphs well (see § 4.1.6 for a detailed discussion on this part of their work). Still, the bulk of the paper is dedicated to the LLP ordering that targets social networks in the first place. To understand LLP, we first explain three other related schemes: a generic label propagation algorithm, a simple label propagation algorithm (LPA) [364] and a variant of the Absolute Potts Model (APM) scheme [373] that builds upon LPA.

First, any label propagation algorithm executes in rounds. At each round, every vertex updates its label according to some rule; this rule's exact design constitutes the difference between various label propagation algorithms. Before the first iteration each vertex has a different label; the algorithm terminates when no more update takes place.

Second, in LPA, a vertex decides to adopt a label that is used by most of its neighbors. Its main problem is that it tends to produce one giant cluster with the majority of vertices.

APM addresses this issue. Assume a vertex $v$ has $k$ neighbors and let $\lambda_1, ..., \lambda_k$ be the labels belonging to $v$'s neighbors. Let also $k_i$ and $v_i$ be the number of $v$'s neighbors with a label $\lambda_i$ and the total number of vertices in $G$ with $\lambda_i$, respectively. Now, when updating its label, instead of selecting a label $\lambda_i$ that has the maximum value of $k_i$, $v$ selects a label that maximizes the value $k_i - \gamma(v_i - k_i)$. Intuitively, this rule does not only increase the density (i.e., the number of edges) of a given community (which happens because $k_i$ new edges adjacent to $v$ join a given community), but also decreases it because of $v_i - k_i$ non-existing edges. The $\gamma$ parameter controls the importance of each of these two effects. This strategy prevents generating one huge giant cluster.

Now, to understand the idea behind the LLP scheme, first observe that different values of $\gamma$ unveil clusters of different resolutions. If $\gamma$ is close to 0 it highlights large clusters (when $\gamma = 0$ then APM degenerates to LPA); increasing $\gamma$ unveils small clusters. LLP attempts to obtain a labeling that considers clusters of various resolutions. In general, it iteratively executes APM with various $\gamma$ values. Now, each such iteration outputs a vertex labeling. Vertices within the same cluster maintain the same order from past iteration. Vertices that acquired the same label are attempted to be placed as close to one another as possible.

Boldi and Santini [58] show in more detail advantages of using a clustering algorithm described in the LLP paper [57] for web graphs. They discuss how to use it to enhance both locality and similarity and provide several interesting examples visualized with associated adjacency matrices.

Shi et al [390] illustrate that the $k^2$ tree representation (§ 4.1.3) can be enhanced in several ways. Among others, they propose to use the DFS vertex order combined with a heuristic that reorders the adjacency matrix to make sure the cells with "1" are concentrated in few submatrices. To acheive this, the heuristic huses the Jaccard coefficient for the structural similarity of any two vertices.

Liakos et al. [280, 281] use the fact that the LLP reordering enhances the locality in such a way that the corresponding AM contains a large "stripe" around its diagonal that groups a large fraction of edges. They use a bitvector to represent these edges and ultimately reduce space to store a network. Finally, Cohen briefly discusses various strategies for social network compression [121].

*4.2.1 Combining Web and Social Networks.* Among the works described in § 4.1 and § 4.2, some are dedicated to compressing both web and social networks [58, 60, 106, 116, 143, 206–208, 312]. Zhang et al. [438] propose the *bound-triangulation* algorithm. The main idea is to use a data structure that stores triangles efficiently. The motivation is that many web graphs and social networks contain a large number of triangles, thus priority placed over storing this motif efficiently reduces the required storage. Angelino [23] proposes a new vertex ordering that considers semantic data associated with the graph. For example, they propose to sort vertex neighborhoods by a selected property such as "name". Miao [316] extracts dense subgraphs from web and social graphs and encodes them using succinct data structures such as wavelet trees.

### 4.3 Biological Networks

A significant amount of work is dedicated to compressing biological networks. The vast majority are related to genome assembly networks [388]. Besides that, few others exist, for example on compressing gene regulatory networks [145] and metabolic graphs [34], or optimizing protein network alignment [233]. There also exists a survey [213] on compressing various types of biological data (not necessarily graphs).

*4.3.1 Schemes Based on De Bruijn Graphs.* De Bruijn graph [133, 189, 380] is a directed graph that represents overlaps between sequences of symbols. For a given set of symbols of cardinality $s$, the corresponding $N$-dimensional De Bruijn graph has $s^N$ vertices consisting of all possible sequences of these symbols; each symbol may appear more than once in a sequence. An edge from a vertex $v_1$ to a vertex $v_2$ exists iff we can shift all the symbols associated with $v_1$ by one place to the left and add a symbol at the end, and ultimately obtain the sequence associated with $v_2$.

De Bruijn graphs are commonly used in the *de novo* genome assembly [37, 88, 124, 127, 171, 278, 347, 397, 434], which is one of fundamental bioinformatics projects. Some specific applications include assembly of DNA sequences [216], mRNA [190] assembly, metagenome assembly [348], genomic variants detection [217, 351] and *de novo* alternative splicing calling [375].

Genome assembly process builds long contiguous DNA sequences (called contigs) from a set of much shorter DNA fragments (called reads). Assemblers based on De Bruijn graphs first extract subsequences (mers) of length $K$ from reads ($K$ is a parameter). Then, a De Bruijn graph consisting of mers as vertices is built and then simplified, if possible. Now, contigs are simple paths in this graph and then can be extracted by finding a Hamiltonian or (more preferably) an Eulerian path.

Construction and navigation of the graph is a practical space and time bottleneck, which is why space-efficient representations of de Bruijn graphs have been researched intensely. The storage lower bound ([bits]) of a De Bruijn graph constructed from $K$-mers is $\log_2 \binom{4^{K+1}}{m}$ ($m = |E|$).

Li et al. [278] were the first to use De Bruijn graphs in assembly of human genome with mers large enough to detect structural variation between human individuals, to annotate genes, and to analyze genomes of novel species. They used minimum-information de Bruijn graphs without the information on read locations and paired-end information. There were various previous short-read assemblers, including EULER [353], Velvet [435], ALLPATHS [88], and EULER-SR [96]. Yet, they all are targeted at bacteria- or fungi-sized genomes, and are mostly unable to manage large genomes. ABySS is another assembler, implemented with MPI on distributed-memory machines for more performance [396]. It avoids using pointers in a De Bruijn graph representation for memory savings; the graph is represented as a distributed hash table, acting as a mapping from a $K$-mer to a byte with the connectivity information related to this mer.

Ye et al. [430] show how to construct a graph equivalent to the de Bruijn graph by storing only one out of $g$ vertices ($g \in [10; 25]$) Their approach involves skipping a fraction of $K$-mers to reduce memory consumption. As an example, assume that there are two pairs of overlapping vertices: $A, B$ and $B, C$. The authors simply store the overlap $(A, C)$ instead of two overlaps $(A, B)$ and $(B, C)$, eliminating read $B$ from the graph. They attempt to sample one out of every $g$ ($g < K$) $K$-mers.

Cazaux et al. [93], Minkin et al. [318], and others [48, 179, 283, 369, 374] show fast and space-efficient algorithms for constructing compact De Bruijn graphs.

Other works include building a space- and time-efficient index used for pattern-matching in De Bruijn graphs [13] and compacting De Bruijn graphs with little memory [109]. Various other works exist [14, 49, 92, 108, 186, 224, 284, 306, 349]. Finally, for completeness, we also mention studies into probabilistic De Bruijn graphs [50, 51, 345].

Current main approaches for a compact De Bruijn graph representation are based on Bloom filters [56], a variant of Burrows-Wheeler Transform [87], and succinct data structures (§ 6).

**Schemes Based on Bloom Filters** Chikhi and Rizk [110] use a Bloom filter to store edges (with additional structures to avoid false positive edges that would affect the assembly). They traverse the graph by generating all possible outgoing edges at each vertex and testing their membership in the Bloom filter. Next, Salikhov et al. [381] design *cascading* Bloom filters to outperform storage requirements of Chikhi and Rizk's approach. They change the representation of the set of false positives. The key idea is to iteratively apply a Bloom filter to: (1) represent the set of false positives, (2) then the set of "false false positives", and so on. This cascade enables 30% to 40% less memory with respect to Chikhi and Rizk's method [381]. Other authors used Bloom filters to implement de Bruijn graphs for pan-genomics [210] and to enhance connecting reads [418]. A redesign of the ABySS scheme was recently implemented using Bloom filters [220].

**Schemes Based on Succinct Data Structures** Conway and Bromage [125] use succinct (entropy-compressed, see § 6) data structures for a practical representation of the De Bruijn assembly graph [125]. They use [336] as succinct representations of a bitmap used to represent De Bruijn graphs. Bowe et al. [67] also incorporate succinctness. They show a representation that uses $4m + o(m)$ bits of a De Bruijn graph with $m$ edges and ensure various graph queries in constant time (for example, computing the in- and out-degree of a vertex). The structure is constructed in $O(NK \log m / \log \log m)$ time using no additional space where $K$ and $N$ and lengths of mers and the whole DNA, respectively. The authors combine (1) succinct static strings due to Ferragina et al. [169], (2) succinct dynamic strings [330], and (3) the XBW-transform structure [167]. Bowe et al.'s work was expanded by Boucher et al. [66], Belazzougui et al. [45, 46], and Pandey et al. [340]. Succinct colored De Bruijn graphs were also discussed [12, 47].

**Schemes Based on Burrows-Wheeler Transform** Various works incorporate the Burrows-Wheeler Transform for more space efficiency [35, 182, 288, 372]

*4.3.2 Grammar- and Text-Related Works.* Peshkin [350] uses the notions from both graph grammars and graph compression to understand the structure of DNA and simultaneously be able to represent it compactly. He proposes the Graphitour algorithm that finds a simplified graph to construct the input graph representing the structure of a given DNA sequence. The simplification scheme is based on contracting edges that satisfy certain criteria regarding their similarity. Next, Hayashida and Akutsu [201] use and extend Graphitour to be able to compare two different biological networks. Specifically, they assess the similarity of two networks by comparing the compression ratios of these two networks when compressed using the modified Graphitour variant. The work is applied to various metabolic networks. Finally, there are other grammar- and text-related works [186] that treat genome sequence as piece of text.

*4.3.3 Hierarchical Approaches.* Hierarchical approaches based on merging groups of vertices into supervertices can also be found in this domain. Brown et al. [83] consider two vertices similar if a high proportion of their neighbours are common. Such vertices are merged to form supervertices. Other similar approaches uses genetic algorithms to find similar vertices efficiently [122, 433].

*4.3.4 Others.* Other approaches include novel types of space-efficient graphs such as Superstring graphs [94] or compressing frequent motifs in a given biological network for not only storage reductions but also faster discovery of various patterns [420].

### 4.4 RDF Graphs

The Resource Description Framework (RDF) is a set of World Wide Web Consortium (W3C) specifications designed to provide the semantic information in a format interpretable by machines. An RDF graph is a set of triples consisting of a *subject*, a *predicate*, and an *object*. Any of the triple elements can be a string; storing the triples explicitly can be memory intensive. One can thus assign identifiers to such values and use a dictionary to map them to concrete value. Consequently, to compress RDF graphs one can compress the dictionary or the underlying graph structure.

*4.4.1 Modeling RDF Graphs As Relational Databases.* Early approaches for compressing RDF graphs map the graphs to relational databases. One way is to simply store all RDF triples in a *triple store*: a table with 3 attributes (columns), an approach used in RDF storage systems such as Jena [244, 313], Sesame [82], and 3store [200]. Another approach is to use *property tables*. In an example scheme, several tables can be built and the attributes in each are properties shared by the triples; the remaining triples that do not fit into the property tables are stored in a triple store [424]. Third, researchers also proposed vertical partitioning [1], a scheme where there is one table per one property. The core idea is thus to group triples by predicate, generating many 2-attribute tables (one for a single predicate value). Finally, other works include space reduction schemes in the Hexastore [422], RDF-3X [333], TripleBit [432], or BitMap [32] systems. RDF-3X and BitMap use gap compression in various parts of the system; for example, RDF-3X condenses indexes in leaves of the underlying B+-tree [3].

*4.4.2 Understanding and Utilizing RDF Redundancy.* Pan et al. [338] first categorize the redundancy in RDF graphs into three different types: *semantic*, *syntactic*, and *symbolic*. Semantic redundancy can be found in RDF graphs that use more triples than necessary to describe a given set of data (i.e., they are not semantically richer than their subgraphs with fewer triples). Syntactic redundancy can be found in graphs that use excessive syntax (e.g., a plain list of triples) instead of a more compact one (e.g., binary serialization). Finally, symbolic redundancy takes place when the average number of bits needed for encoding a basic symbol (RDF resource) is not optimal. After the redundancy analysis, the authors propose to compress RDF datasets by using frequent graph patterns to remove all three aforementioned types of redundancies.

In another piece of work, Pan et al. [339] exploit the RDF graph structure to enhance compression at both the semantic and syntactic level. For the semantic level, they develop a generic framework to replace instances of the bigger graph patterns with smaller instances of the smaller graph patterns (i.e., they eliminate semantic redundancies). This approach is similar to the grammar-based compression schemes for web graphs where more complex generation rules were replaced with simpler and smaller ones. Moreover, for the syntactic level, they illustrate that the same set of RDF triples can occupy various amounts of space depending on how triples are serialised in an RDF file. They identify *intra-structure redundancies* (multiple occurrences of identical RDF resources within the same RDF subgraph) and *inter-structure redundancies* (multiple occurrences of identical resources across different RDF subgraphs).

Moreover, Fernandez et al. [161] analyzes the compressibility of RDF data sets. Specifically, the authors show that large RDF graphs can effectively be compressed because of the power law vertex degree distribution, the hierarchical organization of URLs, and the verbosity of the RDF syntax. Esposito et al. [151] develop algorithms for detecting various RDF redundancies. Pichler et al. [356] analyze the complexity of detecting redundancy in RDF datasets. Wu et al. [426] categorize RDF redundancy redundancy and design new methods for detecting redundancy. Finally, various schemes and analyses of redundancy elimination in RDF graphs were proposed and conducted by Pichler et al. [354, 355], Meier [314], Iannone et al. [215], Grimm and Wissmann [195],

*4.4.3 Incorporating HDT Structure.* Fernandez et al. [163, 165] design Header-Dictionary-Triples (HDT): an RDF representation that partitions RDF graph data into three modules dedicated to the RDF header information, a dictionary, and the actual triples' structure. The modular design reduces redundancy and limits required storage by up to more than an order of magnitude. HDT takes advantage of the power law distribution in the items in RDF datasets. The size reduction is achieved due to a more condensed representation rather than a reduction in the number of triples. Next, Fernandez et al. [162] compress RDF streams by proposing the Efficient RDF Interchange (ERI) format that exploits the regularity of RDF streams. Hernandez-Illerai et al. [209] extend HDT with HDT++. HDT++ alleviates various redundancies (e.g., they group objects per predicate). They ultimately compress some popular RDF datasets by more than 50% and outperform the state-of-the-art $k^2$ trees in size by 10–13%.

*4.4.4 Incorporating MapReduce.* Several works attempt to reduce the size of RDF datasets with MapReduce (MR) [135]. Gimenez-Garcia et al. [185] use MR to process large RDF datasets and serialize them into the Header-Dictionary-Triples (HDT) format [165] that reduces storage overheads behind RDF graphs. Urbani et al. [417] propose to use MR to overcome the scalability problems of compressing large RDF graphs. Specifically, they use MR to construct an RDF dictionary. Similarly, Cheng et al. [104] also reduce the size of RDF graphs; they use the X10 language [98] to construct RDF dictionaries. There are other similar approaches [184, 214, 416].

*4.4.5 Generating Equivalent and Smaller Rules.* Joshi et al. [228–230] propose Rule Based Compression (RBC): a compression technique for RDF datasets that generates a set of new logical rules from a given dataset and removes triples that can be inferred from these rules. The authors show that RBC can prune up to 50% of the original triples without destroying data integrity. For example, a triple $< A, \text{grandfather-of}, C >$ can be generated from triples $< A, \text{father-of}, B >$, $< B, \text{father-of}, C >$ assuming the introduction of an ontology appropriately connecting relations father–of and grandfather–of.

Fernandez et al. [166] propose a scheme called RDF Differential Stream. It uses structural similarities among items in a stream of RDF triples and combines differential encoding with zlib. Zhang et al. [436] compress RDF datasets with Adaptive Structural Summary for RDF Graph (ASSG): a compression method that uses bisimulation [146] to create an equivalent graph of smaller size where vertices with identical labels are collapsed into fewer vertices. Lyko et al. [297] use logical implications contained in the data to develop rules and simultaneously minimize the number of triples that need to be stored. Gayathri et al. [178] mine logical Horn rules [212] from the RDF datasets and then store only the triples matching the antecedent part of the mined rules. Triples matching the head part of the rules are deleted because they can be inferred by applying the rules. Guang et al. [196] propose rule-based methods to find and delete semantically redundant triples.

*4.4.6 Incorporating Hierarchical Schemes.* Fernandez et al. [164] compress RDF graphs by grouping triples with the common subject into adjacency lists. Then, for each RDF property value and subject, it stores ordered IDs of the associated objects. The derived ID sequences are treated with Huffman encoding and PPMd 7-zip [342]. Next, Jiang et al. [225] propose two schemes. First, they assign a *type* to each RDF object and subject and then reduce the number of vertices in the RDF graph by grouping and collapsing RDF entities with the same type. Second, they compress the RDF graph by removing vertices with only one neighbor and maintaining the information on the removed vertex at its neighbor. Finally, Bazoobandi et al. [42] propose a new variant of the Trie structure [134] and use it as a dictionary for RDF datasets in a dynamic and streaming setting. They specifically alleviate common prefixes found in strings in RDF datasets.

*4.4.7 Compressing RDF Dictionaries.* Martinez et al. [309, 310] specifically compress RDF dictionaries. Among others, they apply existing techniques for compressing string dictionaries, including a compact form of *hashing* [74, 126], *Front-Coding* [425] (both Plain Front-Coding [74] and Hu-Tucker Front-Coding [257]), and various forms of *self-indexing* [74, 87, 168, 329]. Moreover, Dawelbeit and McCrindle [130] compress RDF dictionaries that are used in Google BigQuery.

*4.4.8 Using $k^2$ Trees.* Another recent work [18–20] combines vertical partitioning [1] with $k^2$-trees [78, 79]. The core technique is called $k^2$-triples. It first vertically partitions the dataset into disjoint subsets of pairs (subject, object), one subset per predicate. Next, these subsets of pairs are represented as binary matrices where one cell with “1” indicates that a given triple exists in a given RDF graph. These matrices turn out sparse and they are then encoded with $k^2$-trees. A related work by Alvarez et al. [22, 175] advocates *Interleaved* $k^2$ trees: a compressed and self-indexed representation with efficient querying of general ternary relations, similar to $k^2$ trees and their application in compressing binary relations. The main idea is to represent a given set of triples as $x$ binary relations and then use $x$ $k^2$ trees and gather them within a single tree. Interleaved $k^2$ trees can be applied to generic ternary relations; they are evaluated on RDF. Next, Brisaboa et al. [76] propose a dynamic data structure to compactly represent binary relations; the structure is a dynamic variant of the $k^2$ tree. The above-described efforts into combining RDF datasets and $k^2$ trees were also described in theses by Alvarez [17] and Roca [132].

*4.4.9 Using Succinct Data Structures.* Cure et al. [128] use succinct data structures to compress RDF data and to ultimately alleviate scalability issues.

*4.4.10 Others.* Other efforts include the following related work. Swacha and Grabowski [402] compress RDF datasets with a combination of techniques. They separate semantic (i.e., RDF specialized) and general-purpose encoding. They also separate graph and dictionary compression, and combine various techniques over the contents (e.g., run-length encoding or reordering the content). Zneika et al. [443] summarize RDF datasets by adding the information on various instances of data patterns for more performance. Fernandez [160] compacts RDF datasets with a combination of various techniques. Jagalpure [223] designs novel indexing techniques for more scalable and storage-efficient RDF databases. Weaver and Williams incorporate a subset of the Turtle syntax [43] and Lempel-Ziv-Oberhumer (LZO) compression to reduce I/O load in parallel RDF systems. Joshi et al. [231] exploit ontology alignments and application context in RDF graphs for compression. Gallego et al. [173] focus on compressing RDF data in the context of multimedia retrieval. Deme et al. [138] enhance the design of the RDSZ scheme. Brisaboa et al. [77] propose a novel RDF storage scheme called RDFCSA that combines the data and the associated index in a single representation and builds on suffix arrays. Bit vectors for compressing RDF are used by Atre [31].

### 4.5 Network Graphs

Some works target compressing graphs originating in the area of networking. Gilbert el al. [183] summarize IP networks to facilitate visualization. They refer to it as the *semantic graph compression* to distinguish it from the *algorithmic graph compression* where a graph is condensed to reduce the time or space complexity of an associated graph algorithm. They preserve selected properties, for example connectivity: if an underlying graph is connected, the compressed one should also be connected. Moreover, they also develop compression schemes that collapse similar vertices into one (hierarchical schemes). The similarity measure is derived from only topological information or from vertex or edge properties included in the dataset.

Jusko et al. [232] use Bloom filters [56] to develop representations of connection graphs (e.g., P2P overlays) that reduce the amount of consumed memory; the targeted setting is Software Defined Networking (SDN) [263]. The representation enables network elements to determine which connections are to be escalated for further processing and it simultaneously prohibits extracting any other information from the graph for security. The connection graph is assumed to be dynamic.

Shi et al.[389] compress network traffic graphs by grouping motifs such as a clique into single vertices; thus again incorporating a hierarchical compression scheme.

Other works include compressing changes in network monitoring data [103].

### 4.6 Chemistry Networks

There are only very few works related to compressing graphs used in various chemical sciences. In general, there exist some studies on applying graph theory in chemistry [36]. Compression of such graphs was mostly not addressed. Burger et al. [85, 86] address compressing graphs used to model Super Carbon Nanotubes (SCNTs). Example such graphs are Hierarchically Symmetric Graphs (HSG) [384]. They can model hierarchical structure of SCNTs. The authors present the Compressed Symmetric Graphs (CSG) that is constructed out of the description of an HSG while exploiting the structural symmetry in the HSG to only store nodes and edges required for efficiently reconstructing requested parts of the original graph on-the-fly.

### 4.7 Geographical Datasets

One may distinguish two subareas in compressing graph-related datasets in geographical sciences.

*4.7.1 Compressing Terrain Datasets.* Geography Information System (GIS) data is usually 3D terrain data. Thus, is it stored with 3D meshes. For compressing such datasets, one could use any of the available generic schemes for mesh compression; they were covered in several surveys [300, 346, 403]. There are also works related to specifically compressing GIS data. For example, Pradhan et al. [360] target GIS data with Delaunay triangulation [180].

*4.7.2 Compressing Raster Datasets.* This subdomain is mildly related to graph compression, we still present it for completeness. In short, raster data is commonly used in GIS to represent attributes of the space such as temperatures or elevation measures. Now, these sets can often be represented as matrices. Ladra et al. [266, 267] use two ideas to compress such matrices: they first construct a special structure called $k^2$ raster that is based on $k^2$ trees. On top of $k^2$ trees, the nodes in the tree maintain the maximum and minimum values of each submatrix, which are associated with the representation of the raster data. This also provides the indexing functionality. Selected compact data structures are then used to encode elements of $k^2$ raster, such as the tree structure.

### 4.8 VLSI Graphs

Yang et al. [429] compress VLSI structures. They focus on the EDIF (Electronic Data Interchange Format) [123] data format. Their main idea is to use various data mining algorithms to discover redundancies, for example multiple identical subgraphs, and use the redundancies for compression.

## 5 COMPRESSING GRAPH DATABASES

We now consider works dedicated to compressing graph databases. Graph database is a database that stores graphs and enables semantic queries over them. An example of such a database is neo4j [421] or G* [265] that specifically targets compressing dynamic graphs.

### 5.1 Bitmap-Based Schemes

DEX [308] is a general-purpose system for managing and processing large graphs. It uses an internal representation based on compressed bitmaps [307] for efficient basic navigation operations.

### 5.2 $k^2$ Tree-Based Schemes

Lehmann and Perez [273] report empirical results on implementing graph queries over graph representations compressed with $k^2$ trees. They focus on two-way regular-path queries (2RPQs) as these queries can express navigating graphs with paths defined by regular expressions. Alvarez et al. [16] present a new model and representation of general graph databases, where nodes and edges are typed, labeled, and possibly attributed, and graphs may be multigraphs. They also discuss efficient implementation of graph navigation operations. Specifically, they propose the Compact Graph Database structure in which any multigraph is represented using three $k^2$ trees for three relations: (1) a relation between nodes and their attributes, (2) a relation between edges and their attributes, and (3) a relation between nodes (i.e., the actual edges). An interesting aspect of their work is a formal model of a labeled, attributed, and typed multigraph, that is a 10-tuple $(\Sigma_N, \Sigma_E, N, E, ST, \Sigma_A, NS, ES, NA, EA)$. $\Sigma_N, \Sigma_E$ are sets with node and edge types; $N, E$ are sets of pairs that associate numeric node or edge numeric identifiers with their types; $ST$ is a set of pairs that associate an edge numeric identifier with a pair of this edge source and destination nodes; $\Sigma_A$ is a set containing attribute names; $NS, ES$ are schemes that describe attributes of each node or edge type; $NA, EA$ are sets with pairs associating node or edge attributes and their values.

### 5.3 Succinct Data Structures

ZipG [253] is a system that transforms the input graph data into two flat unstructured files that contain the vertex and edge info, respectively. In addition, these files also store small amount of metadata together with the original input graph data for efficient interactive queries. Now, the ZipG is implemented on top of Succinct [6], a distributed compressed data store offering random access and arbitrary substring search queries on compressed unstructured data and key-value pairs.

### 5.4 Hierarchical Schemes

Maccioni and Abadi [298] introduce a compression scheme where certain subgraphs are collapsed to reduce the number of edges at the cost of introducing an additional vertex called a *compressor vertex*. The considered subgraph is identical to a 3-stage Clos topology [119] and the compression method removes the middle stage of vertices and instead introduces the additional vertex connected to all vertices in the original first and third Clos stage. This work is extended [299] to cover other subgraphs that can be replaced with sparser subgraphs.

### 5.5 Compressing Associated Data Structures

Several works compress various data structures related to the graph data and used to, for instance, speed up some queries or for indexing purposes. Ferragina et al. [170] compress indexing schemes for large graphs that have vertices labeled with variable-length strings. Jin et al. [226] compress transitive closures with spanning trees.

### 5.6 Others

Gbase [235, 236] is a graph management system that takes as input a single big file with a list of edges and partitions it into several homogeneous blocks. Second, vertices are reshuffled, i.e., they are placed in the blocks where the majority of their neighbors reside. Thanks to it, the resulting blocks are either sparse or dense. Next, Gbase compresses all non-empty blocks through standard compression such as gzip. Finally, the compressed blocks (and some meta information (e.g., the block row id) are stored in the graph databases.

## 6 APPROACHING STORAGE LOWER BOUNDS

The core idea behind these schemes is to encode a given graph so that the representation explicitly approaches the storage lower bound. Unless stated otherwise, all schemes operate on static graphs. Tables 2 and 3 feature the considered representations together with storage complexities.

### 6.1 Related Concepts

We first explain all the related notions: *succinct* or *compact* graph representations. We define them in § 6.1.1. Next, these terms are sometimes used imprecisely; we clarify such issues and provide a taxonomy of these terms used in this survey in § 6.1.2. We also illustrate the main techniques used in achieving succinctness in the majority of graph representations and others in § 6.1.3.

*6.1.1 Succinctness and Compactness: Definitions.* Assume $\mathcal{N}$ is the optimal number of bits to store some data. A representation of this data is *compact* if it uses $O(\mathcal{N})$ bits and *succinct* if it uses $\mathcal{N} + o(\mathcal{N})$ bits. These definitions are used in various modern works on succinct data structures [11, 137, 323]. Many of these representations simultaneously support *a reasonable set of queries fast* (e.g., in $O(1)$ or $O(\log n)$ time) [54], others only consider reducing space complexity. Finally, some designs accelerate the process of generating (encoding) a given data representation.

Specifically, when considering an arbitrary (undirected) graph with $n$ vertices and $m$ edges, the number of such graphs is $\mathcal{N} = \binom{\binom{n}{2}}{m}$, the storage lower bound is $\lceil \log \mathcal{N} \rceil$, and thus a succinct and compact representation respectively take $\lceil \log \mathcal{N} \rceil + o(\log \mathcal{N})$ and $O(\log \mathcal{N})$ bits.

One can equivalently define succinctness via entropy. For example, Aleardi et al. [9, 11] state that a data structure is succinct if its asymptotic size matches the entropy of the class of represented structures and compact if it matches it up to a constant factor. In the language of the terms used above, the class of represented structures are arbitrary graphs.

*6.1.2 Succinctness and Compactness: Taxonomy of Concepts.* We now clarify certain issues related to succinctness and compactness. Recent works associate these terms with the definitions in § 6.1.1. Now, there exist various works that use different, although related, definitions of succinctness and compactness. For example, Galperin and Wigderson state that a succinct graph representation takes $o(n)$ space [174]. We now describe these aspects and introduce notions that are used in the remainder of this section and in Table 2 and 3. Specifically, we use the following terms:

**succinct** This term indicates that a representation uses the definition of succinctness from § 6.1.1 or the definition of succinctness based on entropy [9, 11] provided in the last paragraph of § 6.1.1.

| Reference* | Size [bits] | Labels ** | Edges *** | Fast access / encoding**** | Targeted graph family (see § 2.3) | Scheme type (see § 6.1) |
|---|---|---|---|---|---|---|
| **Targeted graphs: planar and planar-like (maps, triangulations)** | | | | | | |
| Itai [218] | $\frac{3}{2} n \log n + O(n)$ | yes | undir. | no / yes | Triangulation | Space/work-optimal |
| Turan [415] | $\leq 12n$ | no | undir. | no / yes | Simple | "succinct" |
| Turan [415] | $n\lceil \log n \rceil + 12n$ | yes | undir. | no / yes | Simple | "succinct" |
| Tamassia [406] | $O(n)$ | unsp. | both | yes / yes | Planar embedding | compact |
| Keeler [243] | $n \log n + m \log 12 + o(n)$ | yes | undir. | no / yes | General | space-efficient |
| Keeler [243] | $3m + O(1)$ | no | undir. | no / yes | Map, stick-free, loop-free | space-efficient |
| Keeler [243] | $m \log 12 + O(1)$ | no | undir. | no / yes | General | space-efficient |
| Keeler [243] | $m \log 12 + O(1)$ | no | undir. | no / yes | Map, stick-free | space-efficient |
| Keeler [243] | $m \log 12 + O(1)$ | no | undir. | no / yes | Map, stick-free | space-efficient |
| Keeler [243] | $(3 + \log 3)m/3 + O(1)$ | no | undir. | no / yes | Triangulation | space-efficient |
| Munro [324, 325] | $8n + 2m + o(n + m)$ | yes | undir. | yes / no | General | succinct |
| Chuang [114] | $2m + \left(5 + \frac{1}{k}\right) n + o(m + n)$ | no | undir. | yes↑ / yes | General, loop-free, $k > 0$ | compact |
| Chuang [114] | $2m + \frac{14}{3} n + o(m + n)$ | no | undir. | yes / yes | General, loop-free | compact |
| Chuang [114] | $\frac{5}{3} m + \left(5 + \frac{1}{k}\right) n + o(n)$ | no | undir. | yes↑ / yes | Simple, $k > 0$ | compact |
| Chuang [114] | $\frac{4}{3} m + 5n + o(n)$ | no | undir. | yes / yes | Simple | compact |
| Chuang [114] | $2m + 3n + o(m + n)$ | no | undir. | yes / yes | General, 3-connected, loop-free | compact |
| Chuang [114] | $2m + 2n + o(n)$ | no | undir. | yes / yes | Simple, 3-connected | compact |
| Chuang [114] | $2m + 2n + o(m + n)$ | no | undir. | yes / yes | General, triangulated, loop-free | compact |
| Chuang [114] | $2m + n + o(n)$ | no | undir. | yes / yes | Simple, triangulated | compact |
| Chuang [114] | $(n + m) \log 3 + 1$ | no | undir. | no / yes | Simple, 3-connected | compact |
| Chuang [114] | $(\min\{n, f\} + m) \log 3 + 2$ | no | undir. | no / yes | Simple, 3-connected | compact |
| Chuang [114] | $2m + \left(6 + \frac{1}{k}\right) n + o(m + n)$ | no | undir. | yes↑ / yes | General, $k > 0$ | compact |
| Chuang [114] | $2m + \frac{17}{3} n + o(m + n)$ | no | undir. | yes / yes | General | compact |
| Chuang [114] | $\frac{5}{3} m + \left(6 + \frac{1}{k}\right) n + o(n)$ | no | undir. | yes↑ / yes | Simple, loops, $k > 0$ | compact |
| Chuang [114] | $\frac{4}{3} m + 6n + o(n)$ | no | undir. | yes / yes | Simple, loops | compact |
| Chuang [114] | $2m + 4n + o(m + n)$ | no | undir. | yes / yes | General, 3-connected | compact |
| Chuang [114] | $2m + 3n + o(n)$ | no | undir. | yes / yes | Simple, 3-connected, loops | compact |
| Chuang [114] | $2m + 3n + o(m + n)$ | no | undir. | yes / yes | General, triangulated | compact |
| Chuang [114] | $2m + 2n + o(n)$ | no | undir. | yes / yes | Simple, triangulated, loops | compact |
| Chuang [114] | $(n + m) \log 3 + n + 1$ | no | undir. | no / yes | Simple, 3-connected, loops | compact |
| Chuang [114] | $(\min\{n, f\} + m) \log 3 + n + 2$ | no | undir. | no / yes | Simple, 3-connected, loops | compact |
| King [254] | $\frac{11}{3} n$ | yes | unsp. | no / yes | Planar, triangle, loop-free | "compact" |
| He [202] | $4n - 9 = \frac{4}{3} m - 1$ | no | undir. | no / yes | Plane triangulation | "succinct" |
| He [202] | $(\frac{5}{2} + 2 \log 3) \min\{n, f\} - 7$ | no | undir. | no / yes | Plane 3-connected | "succinct" |
| He [203] | $\beta(n) + o(\beta(n))$ | yes | both | no / yes | Plane triangulation | "succinct" |
| He [203] | $\beta(n) + o(\beta(n))$ | yes | both | no / yes | Plane or planar | "succinct" |
| Chiang [105] | $2m + 3n + o(m + n)$ | no | undir. | yes / yes | General, loop-free | space-efficient |
| Chiang [105] | $2m + 2n + o(n)$ | no | undir. | yes / yes | Simple, loop-free | space-efficient |
| Poulalhon [357, 358] | $4n$ | unsp. | unsp. | no / no | Triangulation | "succinct" |
| Aleardi [9, 10] | $2.175t + O\left(t \frac{\log \log t}{\log t}\right)$ | unsp. | unsp. | yes / yes | Triangulation | succinct |
| Aleardi [9, 10] | $2.175t + 36(g - 1) \log t$ $+O\left(t \frac{\log \log t}{\log t} + g \log \log t\right)$ | unsp. | unsp. | yes / yes | Triangulation of a surface with genus $g$ | succinct |
| Barbay [38, 39] | $2m \log 6 + o(m)$ | no | unsp. | yes / no | Planar triangulation | succinct |
| Barbay [38, 39] | $t \log \sigma + t \cdot o(\log \sigma)$ | no | unsp. | yes / no | Triangulation | succinct |
| Barbay [38, 39] | $t \log \sigma + t \cdot o(\log \sigma)$ | yes | unsp. | yes / no | Triangulation | succinct |
| Barbay [38, 39] | $n + t(\log \sigma + o(\log \sigma))$ | edges | unsp. | yes / no | Outerplanar | succinct |
| Barbay [38, 39] | $4n + t(\log \sigma + o(\log \sigma))$ | edges | unsp. | yes / no | General | succinct |
| Aleardi [11] | $2m + o(n)$ | unsp. | unsp. | yes / yes | 3-connected | succinct |
| Aleardi [11] | $3.24n + o(n)$ | unsp. | unsp. | yes / yes | Triangulated | succinct |
| Blelloch [55] | $\mathcal{H}_p(n) + o(n)$ | no | unsp. | yes / yes | Map | succinct |
| Yamanaka [428] | $6n + o(n)$ | yes | unsp. | yes / no | Plane triangulation | compact |

Table 2. (§ 6.2) Compact and succinct graph representations for graphs that are **planar** or **planar-like** (maps, plane graphs, etc.). * To save space, we only show the first name. ** "yes", "no", "edges" indicate that a graph has vertex labels, has no labels at all, has edge labels; "unsp." means labeling is not mentioned. *** "undir.", "dir.", "both" indicate that a scheme targets undirected graphs, directed graphs, or both; "unsp." means it is unspecified. **** "Fast" indicates that a given scheme attempts to reduce the time complexity of a certain query (queries) or the time to create (i.e., encode) or decode a given representation from the input graph representation (an AL or an AM); "yes↑" indicates that the scheme in the given row offers more efficient operations on the graph than the corresponding scheme in the *below*, possibly at the cost of more storage.

| Reference* | Size [bits] | Labels ** | Edges *** | Fast access / encoding**** | Targeted graph (family; see § 2.3) | Scheme type (see § 6.1) |
|---|---|---|---|---|---|---|
| **Targeted graphs: "middle-ground" ($k$-page, separable) that are more general than the planar ones** | | | | | | |
| Jacobson [221, 222] | $O(kn)$ | no | unsp. | yes / yes | $k$-page | succinct |
| Cohen [120] | $O(n)$ ["space"] | yes | unsp. | yes / yes | $k$-connected | "compact" |
| Munro [324, 325] | $2kn + 2m + o(kn + m)$ | no | yes | | $k$-page | succinct |
| Deo [140] | $O(n + g)$ | yes | undir. | no / yes | bounded genus ($\leq g$) | "compact" |
| Deo [140] | $O(n)$ | yes | undir. | no / yes | bounded arboricity, separable | "compact" |
| Lu [293, 294] | $\leq \beta(n) + o(\beta(n))$ | yes | undir. | no / yes | genus $g = \left(\frac{n}{\log^2 n}\right)$, others (see [293, 294]) | "compact" |
| Blandford [54] | $O(n)$ | no | both | yes / no | separable | compact |
| Barbay [38, 39] | $n + 2m \log k + o(m \log k)$ | no | unsp. | yes / no | $k$-page | succinct |
| Barbay [38, 39] | $n + (2 + \epsilon) m \log k + o(m \log k)$ | no | unsp. | yes / no | $k$-page | succinct |
| Barbay [38, 39] | $kn + t(\log \sigma + o(\log \sigma))$ | edges | unsp. | yes / no | $k$-page | succinct |
| Barbay [38, 39] | $n + (2m + \epsilon) \log k + o(m \log k)$ $+m(\log \sigma + o(\log \sigma))$ | edges | unsp. | yes / no | $k$-page | succinct |
| Gavoille [176] | $2m \log k + 4m$ | no | undir. | no / no | $k$-page, $k \leq \frac{1}{2} kn / \log k$ | "compact" |
| Gavoille [176] | $2m \log k + 4m + o(m \log k)$ | no | undir. | yes↑ / no | $k$-page, $k \leq \frac{1}{2} kn / \log k$ | "compact" |
| Gavoille [176] | $2m \log k + 4m + o(m)$ | no | undir. | yes / no | $k$-page, $k \leq \frac{1}{2} kn / \log k$ | "compact" |
| Blelloch [55] | $\mathcal{H}(n) + o(n)$ | no | yes | yes / no | separable | succinct |
| **Targeted graphs: arbitrary (no or little assumptions on the structure)** | | | | | | |
| Turan [415] | $\binom{n}{2} - \frac{1}{8} n \log n + O(n)$ | no | unsp. | no / no | - | "succinct" |
| Naor [327] | $\binom{n}{2} - n \log n + O(n)$ | no | unsp. | yes / yes | arbitrary adjacency **matrix** | Approach the AM bound $\binom{n}{2}$ $-n \log n + O(n)$ |
| Raman [366] | $\left\lceil \log \binom{n^2}{m} \right\rceil + o(m)$ | no | dir. | yes / no | - | succinct |
| Farzan [154] | $\log \binom{n^2}{m} + o\left(\log \binom{n^2}{m}\right)$ | yes | dir. | yes / no | $m > \frac{n^2}{\log^{1/3} n}$ | succinct |
| Farzan [154] | $(1 + \epsilon) \log \binom{n^2}{m}$ | yes | dir. | yes / no | $\frac{n^2}{\log^{1/3} n} \geq m > \frac{n}{2}$ | succinct |
| Farzan [154] | $\log \binom{n^2}{m} + \epsilon m \log m$ | yes | dir. | yes / no | $\epsilon > 0, m \leq \frac{n}{2}$ | succinct |
| Farzan [154] | $\log \binom{n^2/2}{m}$ | yes | undir. | yes / no | $\frac{n^2}{4} > m > \frac{n^2}{\log^{1/3} n}$ | succinct |
| Farzan [154] | $(1 + \epsilon) \log \binom{n^2/2}{m}$ | yes | undir. | yes / no | $\frac{n^2}{\log^{1/3} n} \geq m > \frac{n}{2}$ | succinct |
| Farzan [154] | $\log \binom{n^2/2}{m} + \epsilon m \log m$ | yes | undir. | yes / no | $\epsilon > 0, m \leq \frac{n}{2}$ | succinct |

Table 3. (§ 6.2) Compact and succinct graph representations for **"middle-ground"** and **arbitrary** graphs. * To save space, we only show the first name. ** "yes", "no", or "edges" indicate that a graph has vertex labels, has no labels at all, or has edge labels; "unsp." means labeling is not mentioned. *** "undir.", "dir.", and "both" indicate that a scheme targets undirected graphs, directed graphs, or both; "unsp." means it is unspecified. **** "Fast" indicates that a given scheme attempts to reduce the time complexity of a certain query (or queries) or the time to create (i.e., encode) or decode a given representation from the input graph representation (an AL or an AM); "yes↑" indicates that the scheme in the given row offers more efficient operations on the graph than the corresponding scheme in the *below*, possibly at the cost of more storage.

**"succinct"** An encoding of a graph $G$ from a class of graphs $\mathcal{G}$ is described with this term if it is described as succinct but the definition does not match the one from § 6.1.1 or the one based on entropy and stated in past work [9, 11]. Example such definitions are "the length of the encoding of $G$'s representation is not too large compared to $\log |\mathcal{G}|$" [415], "the length of this encoding is not much larger than its information-theoretic tight bound, i.e., the shortest length over all possible coding schemes" [202].
**compact** This term indicates that a representation uses the definition of compactness from § 6.1.1.
**"compact"** This term describes a representation that does not explicitly use the definition of compactness from § 6.1.1 but directly compares to and discusses such representations [114].

**space-efficient** We use this term when a given representation claims relationships to succinctness or compactness, and/or extensively discusses succinct or compact representations, but when it is not itself based on any precise optimality conditions. For example, Chiang et al. [105] states that a "space-efficient" representation (1) minimizes the length of a given encoding, (2) minimizes the time required to compute and decode the encoding, and (3) supports queries on this encoding efficiently.

*6.1.3 Main Techniques for Achieving Succinctness.* We now describe generic techniques for obtaining succinctness and compactness that are often used in various representations.

**Hierarchical Decomposition** Assume that the size of a considered class of objects is $\mathcal{N}$. For example, for a class of arbitrary graphs we have $\mathcal{N} = \binom{\binom{n}{2}}{m}$. The high-level key idea is to divide the object to be encoded (e.g., an arbitrary graph) into *small* parts, group these parts in an auxiliary table, and represent them with the indices into this table. This table should contain *all* possible parts so that *any* object from a given class could be constructed from them. These parts are again divided into yet smaller (*tiny*) parts, stored similarly in yet other auxiliary tables. Now, the size of both small and tiny parts is selected in such a way that the sum of the sizes of all the indices and all the auxiliary tables is $O(\mathcal{N})$ (for compactness) and $\mathcal{N} + o(\mathcal{N})$ (for succinctness) bits. The central observation that enables these bounds is that the representation consisting of small and tiny parts can be *hierarchical*: tiny parts only need pointers that point to other tiny parts within a *single* small part because small parts use other pointers that link them to other small parts.

More formally, an object to be encoded (e.g., a graph) is first split into *tiny* parts of size $O(\log \mathcal{N})$. Here, *tiny* means that the catalog of all these parts (i.e., an auxiliary table explicitly storing these parts) must take $o(\mathcal{N})$; in most cases it is $O\left(\frac{\mathcal{N}}{\log \mathcal{N}}\right)$. Any tiny part is then represented with its index in this catalog and the sum of the sizes of all such indices.

Second, one must encode how these tiny parts together form the initial object. Now, the number of these parts is $O\left(\frac{\mathcal{N}}{\log \mathcal{N}}\right)$, the number of connections between them is $O(\mathcal{N})$, and a pointer to any such part takes $O(\log \mathcal{N})$ bits. Thus, a classical representation of the connections between tiny parts is $O(\mathcal{N})$, *giving a compact representation.*

To achieve succinctness, $\log \mathcal{N}$ tiny parts are combined into *small* parts, each of which uses $O(\log^2 \mathcal{N})$ space. This enables using pointers of size $O(\log \mathcal{N})$ between small parts while tiny parts can use pointers of size $O(\log \log \mathcal{N})$ because they now need to point to each other only within one small part. Now, the total count of small and tiny parts is $O\left(\frac{\mathcal{N}}{\log^2 \mathcal{N}}\right)$ and $O\left(\frac{\mathcal{N}}{\log \mathcal{N}}\right)$, respectively. This gives the total size of $O\left(\frac{\mathcal{N}}{\log \mathcal{N}}\right)$ bits and $O\left(\frac{\mathcal{N} \log \log \mathcal{N}}{\log \mathcal{N}}\right)$ bits, respectively.

**Parentheses Encoding** Another general succinct or compact encoding uses parentheses. To explain the idea intuitively, consider a tree and a Depth-First Search traversal of this tree. One can represent this tree with a string consisting of two parentheses, "(" and ")". Namely, during the traversal, when a vertex is visited for the first time, an opening parenthesis "(" is appended to the string. When a vertex is visited for the second and the last time (while moving backwards in the tree structure), the other parenthesis ")" is added. Thus, a tree with $n$ vertices uses $2n$ bits (one bit per one parenthesis type). Now, a graph could be represented in a similar way. For example, if one decompose a graph into a set of spanning trees, each tree could be represented with such a string of parentheses that are in practice encoded with "0"s and "1"s.

### 6.2 Succinct and Compact Schemes

We next describe several concrete succinct and compact representations. We summarize all the considered schemes in Table 2 and Table 3. The former presents planar graphs, subclasses of planar graphs, and planar-related ones such as maps. The latter summarizes "middle-ground" graphs and graphs of arbitrary structure; "middle-ground" are graphs that are more generic than planar ones (edges can cross outside their adjacent vertices), but do have some strong assumptions on the structure, including bounded genus, bounded arboricity, bounded number of pages, or separability. These classes are explained in § 2.3. Finally, arbitrary graphs are graphs with any structure where the only assumption can be related to the number of edges.

The majority of succinct and compact schemes listed in Tables 2 and 3 use one of a few "standard" mechanisms for achieving compactness or succinctness, described in § 6.1.3. We now group these schemes basing on the associated mechanism.

*6.2.1 Schemes Based on Hierarchy.* There are numerous succinct and compact representations that use the generic hierarchical way of obtaining the storage lower bounds. They include planar graphs [9, 11, 55], "middle-ground" graphs [55], and arbitrary graphs [155]. The key idea is as stated in § 6.1.3: all possible parts of any input graph (in a given class) are indexed in a lookup table and pointers of specially engineered sizes are used to ensure that any input graph can be constructed from the indexed elements to provide the desired storage bounds.

*6.2.2 Schemes Based on Parentheses.* Many representations use the concept of parentheses for succinct or compact encoding [176, 221, 324]. In some cases (usually planar graphs), they use one type of parentheses, but several schemes propose to use multiple types of parentheses if the input graph has a more complex structure. Moreover, several other representations use parentheses combined with reordering the vertices according to a special order called the *canonical order* [239]. These are all planar graphs [38, 105, 114, 202].

*6.2.3 Schemes Based on Encoding Trees.* Some representations are based on decomposing the input graph into trees constructed from a DFS graph traversal, and then encoding such trees using a selected scheme [243, 254].

*6.2.4 Others.* There are also other representations [8, 172, 218, 415, 419]. For example, Blandford et al. [54] relabel vertices based on recursive partitioning of the input graph to achieve compactness. Raman et al. [366] use succinct indexable dictionaries as a basis for ensuring succinct binary relations that can then be used to encode succinctly an arbitrary graph.

### 6.3 Other Storage Lower Bound Measures

Besides compactness and succinctness, there exist other concepts related to storage lower bounds that could be used while developing and analyzing compression schemes or storage-efficient representations. A detailed description of such concepts is outside the scope of this work. However, we briefly mention them to make this survey complete and to provide the associated links. One obvious related notion in discussing storage lower bounds is graph entropy; it was covered in several surveys [136, 211, 321, 394, 395]. Another way to describe storage bounds is Kolmogorov complexity of graphs that was covered in some works [204, 277, 317]. These works most often focus on investigating the "information content" of a given graph family, for example the notion of topological entropy [368, 413] is related to the probability of a graph having a certain partitioning structure. Chierichetti at el. [107] discuss the information content of web graphs and propose a graph model that reflects this content.

Some of these works specifically address compression [111–113, 198]. Choi and Szpankowski [111–113] propose the "Structural zip" algorithm for compressing unlabeled graphs; it compresses a given labeled $G$ into a codeword that can be decoded into a graph isomorphic to $G$. The main idea behind the algorithm is as follows. First, a vertex $v_1$ is selected and its neighbor count is stored explicitly. Then, the remaining $n - 1$ vertices are partitioned into two sets: $v_1$'s neighbors and non-neighbors.

This continues recursively by selecting a vertex $v_2$ from $v_1$'s neighbors and storing two numbers: the number of $v_2$ neighbors among each of these two sets. Next, the remaining $n-2$ vertices are partitioned into four further sets: the neighbors of both $v_1$ and $v_2$, the neighbors of $v_1$ that are non-neighbors of $v_2$, the non-neighbors of $v_1$ that are $v_2$'s neighbors, and the non-neighbors of both $v_1$ and $v_2$. This continues until all vertices are processed. During the algorithm execution, two types of encoded neighbor counts are maintained and concatenated into one of the separate binary sequences. First, the neighbor counts of length more than one bit (i.e., for subsets $|U| > 1$) are concatenated to form the first sequence. Second, the neighbor counts of length exactly one bit (i.e., for subsets $|U| = 1$) are concatenated to form the second sequence.

Moreover, Luczak et al. [295] design asymptotically optimal algorithms for compressiing unlabeled and labeled graphs constructed with the preferential attachment model. Others analyze theoretical aspects of compressing clustered graphs [2, 27] or establish a formal relationship between the polynomials with simple zeros and the storage lower bound of a given graph [286].

### 6.4 Discussions on Computational Complexity

Some papers discuss the computational complexity of deriving succinct representations, for example by showing NP-hardness of some schemes [174, 341].

## 7 GRAPH MINIMUM ARRANGEMENT FOR STORAGE REDUCTIONS

Another line of works uses Integer Linear Programming (ILP) formulations to compress graphs by reordering vertex labels such that the new labels can be compressed more effectively. For example, some schemes assign labels to decrease differences between IDs of consecutive neighbors in each neighborhood; these minimized differences are then encoded using some variable-length coding, ultimately reducing the size of each such neighborhood and thus $G$'s size. We already discussed some schemes that aim at improving such reorderings; now we focus on existing research that explicitly uses ILP formulations or improves them. This particular problem is called *Minimum Linear Gap Arrangement* (MLinGapA) because it consists in minimizing linear gaps between consecutive neighbors. There are three other related problems: *Minimum Logarithmic Gap Arrangement* (MLogGapA), *Minimum Linear Arrangement* (MLinA), and *Minimum Logarithmic Arrangement* (MLogA). More generally, this family of problems is called *Minimum Arrangement* problems and are a part of a domain called *Graph Layout* problems [144].

### 7.1 Definitions of Minimimum Arrangement Problems

Formally, a *layout* of an undirected graph $G$ is a bijective function $\phi : V \to [n] = \{1, ..., n\}$ [352] that reassigns labels of vertices so that a certain function is minimized. Now, the definition of Minimum Linear Arrangement problem (MLinA) is as follows: find a layout $\phi^*$ that minimizes the sum of differences of each pair of two vertices connected with an edge:

$$\overbrace{\sum_{v\in V}\sum_{u\in N_v} |\phi^*(v) - \phi^*(u)|}^{\phi^* \text{minimizes this expression}} = \min_{\forall\phi, \forall v\in V} \sum_{v\in V}\sum_{u\in N_v} |\phi(v) - \phi(u)| \tag{6}$$

A strongly related problem is Minimum Logarithmic Arrangement (MLogA) where ones derives a layout $\phi^*$ that minimizes the sum of logarithms of differences; incorporating logarithms takes into account the exact bit count of numbers to be encoded

$$\overbrace{\sum_{v\in V}\sum_{u\in N_v} \log|\phi^*(v) - \phi^*(u)|}^{\phi^* \text{minimizes this expression}} = \min_{\forall\phi, \forall v\in V} \sum_{v\in V}\sum_{u\in N_v} \log|\phi(v) - \phi(u)| \tag{7}$$

Next, the objective function can also minimize the sum of differences between consecutive neighbors in adjacency lists (Minimum Linear Gap Arrangement problem (MLinGapA)), which one can directly use to decrease the storage for a given graph if differences between vertex ID are stored and encoded with variable-length coding:

$$\overbrace{\sum_{v \in V} \sum_{i=0}^{|N_v|-1} |\phi^*(N_{i+1,v}) - \phi^*(N_{i,v})|}^{\phi^* \text{minimizes this expression}} = \min_{\forall \phi, \forall v \in V} \sum_{v \in V} \sum_{i=0}^{|N_v|-1} |\phi(N_{i+1,v}) - \phi(N_{i,v})| \tag{8}$$

Finally, the same problem can be (analogously to MLinA) formulated including logarithms and result in Minimum Logarithmic Gap Arrangement (MLogGapA):

$$\overbrace{\sum_{v \in V} \sum_{i=0}^{|N_v|-1} \log |\phi^*(N_{i+1,v}) - \phi^*(N_{i,v})|}^{\phi^* \text{minimizes this expression}} = \min_{\forall \phi, \forall v \in V} \sum_{v \in V} \sum_{i=0}^{|N_v|-1} \log |\phi(N_{i+1,v}) - \phi(N_{i,v})| \tag{9}$$

### 7.2 Compression Schemes Based on Minimum Arrangement Problems

There exist many works that reduce the complexity or propose heuristics for the arrangement problems in § 7.1. They are listed in existing surveys [144, 352]; we do not explicitly describe them as they do not directly relate to graph compression. Second, various works compress graphs by simply enhancing vertex labelings; we addressed many of these works in previous sections. We now only focus on works that explicitly compress graphs using minimum arrangement ILP formulations.

Safro and Temkin [379] enhance the MLogA for general graphs by incorporating link weights into the ILP formulation. They motivate it by observing that link weight can measure how often a link is used; frequently accessed links would be compressed more effectively. Their algorithm is based on a more generic strategy called the algebraic multigrid (AMG) methodology [69] for linear ordering problems [378]. In AMG, one first decomposes the original problem into several approximate ones. In the case of MLogGapA, each approximate subproblem is based on a projection of the corresponding graph Laplacian into a lower-dimensional space. Then, solutions of subproblems are used to derive the final solution. This approach has two key advantages: it has a linear complexity and can easily be parallelized and implemented with standard matrix–vector primitives. Now, Safro and Temkin first formulate the weighted MLogA problem:

$$\overbrace{\sum_{v \in V} \sum_{u \in N_v} w_{vu} \log |\phi^*(v) - \phi^*(u)|}^{\phi^* \text{minimizes this expression}} = \min_{\forall \phi, \forall v \in V} \sum_{v \in V} \sum_{u \in N_v} w_{vu} log |\phi(v) - \phi(u)| \tag{10}$$

Then, they conduct a series of steps, in each step they reduce the size of the input graph by *coarsening it*: repeatedly merging pairs of vertices that satisfy certain properties. At some point, the (much smaller) obtained graph is used to solve Eq. (10). Then, the original graph is derived by reversing the coarsening effects, with the computed solution updated at each de-coarsening step.

Chierichetti et al. [106] analyze various aspects of Minimum Arrangement problems; they target social networks but their formal analysis is generic. Specifically, they prove that MLogA is NP-hard on multi-graphs (graphs that admit multiple edges between two vertices), MLinGapA is NP-hard, and that MLogA has the time lower bound of $\Omega(m \log n)$ for expander-like graphs. Similarly, Dhulipala et al. [143] discuss arrangement problems in the context of graph compression; they offer a proof of the NP-hardness of MLogGapA and they introduce the ILP formulation of *Bipartite Minimum Logarithmic Arrangement* (BiMLogA), essentially the MLogA for bipartite graphs.

Finally, there exist various algorithms that enhance graph compression by relabeling vertices but without explicitly mentioning the ILP formulation of arrangement problems [54, 57, 60, 61]. We covered them extensively in past sections.

## 8 REMAINING SCHEMES

We also discuss schemes that fall outside other categories. Johnson et al. [227] discusses how to compress binary matrices by reordering the columns so that the whole matrix is more compression-friendly. Such matrices could be used to represent graphs using less storage. Moreover, Borici and Thomo [65] compress graphs by transforming them into corresponding *hypergraphs* and then partitioning the hypergraphs so that vertices with similar properties (e.g., degrees) are in the same partition. This makes the corresponding adjacency matrix more compression-friendly. Other works include compressing dense graphs [240] and vertex-transitive graphs [287].

### 8.1 Hierarchical Schemes

We discuss general hierarchical schemes similar to those presented in the web graph section § 4.1.5.

*8.1.1 Grouping Cells of Adjacency Matrix.* First, we outline works that utilize hierarchy related to adjacency matrices, for example, group non-zero cells into blocks and compress such blocks separately. Lim, Kang, and Faloutsos [234, 282] propose SlashBurn: a scheme that exploits high-degree vertices (*hubs*, found often in real-world graphs) and their neighbors (*spokes*) to achieve high compression ratios. This forms a different type of community structure than the traditional "caveman" communities where vertices are clustered within certain groups ("caves") and sparsely connected to other vertex groups. They propose vertex relabeling that uses this observation and results in space-efficient representation of the adjacency matrix. The SlashBurn algorithm (1) removes high-degree vertices and assign them the lowest labels (2) finds connected components in the resulting graph and assign the vertices in these components the highest labels, in the decreasing order of the sizes of the connected components that they belong to, (3) finds the giant connected component in the resulting graph and executes step (1) on it recursively, until its size is below a certain threshold. SlashBurn was extended to distributed-memory settings. Moreover, Li and Rao compress graphs by grouping parts of the adjacency matrix and using different codes to reduce the space required to store a given group [275]. Furthermore, Li et al. [276] first cluster graph adjacency matrix via graph structure information, and then represent the clustered matrix by lists of encoded numbers. Finally, various schemes described in other parts of this survey are related to compressing adjacency matrices hierarchically. Examples are works on $k^2$-trees [78] (see § 4.1.3).

*8.1.2 Schemes Based on Supervertices.* A large portion of hierarchical schemes explicitly groups vertices with similar properties into *supervertices* (also called *supernodes*) and collapse edges between them into *superedges*. Many of them were described in § 4.1.5. Here, we mention works that are not explicitly related to web graphs. Stanley et al. [399] find clusters in a given graph and then simplify and represent it using supervertices with one vertex being formed from one cluster. Next, Toivonen, Zhou, and others [412, 441] propose a hierarchical scheme that targets weighted graphs. They group vertices with similar neighborhoods into supervertices, and group multiple weighted edges between such supervertices into superedges. Another similar work that considers algorithms for all-pairs shortest paths, bipartite matching, and edge and vertex connectivity was conducted by Feder and Motwani [157, 158]. Moreover, Brown et al. [83] use genetic algorithms to assess the similarity of vertices (where two vertices are considered similar if many of their neighbors are identical). Similar vertices are merged into supervertices and the graph size is ultimately reduced. In addition, Sun et al. [401] measure the overlap of neighbors between vertices and, if the overlap is large enough, the identical neighborhood parts are collapsed and a certain data structure is used to encode this structural change. Lamarche-Perrin et al. [269] target compressing weighted graphs with supervertices. Finally, Nourbakhsh simplifies the input graph (and thus reduces its size). He uses Szemeredi's Regularity Lemma [259] to cluster the graph and to produce a smaller graph where clusters become vertices [334].

*8.1.3 Tree Decompositions.* Some lossless compression schemes decompose a graph into several trees, encode these trees separately, and ultimately reduce the overall space requirements. Chen and Reif [102] decompose an input graph into several binary trees, and finally compress these trees with a proposed tree-compression algorithm. The key idea in compressing a single binary tree is to further decompose this tree into smaller subtrees. These subtrees are small enough that any such subtree can be found multiple times in the input tree. Thus, after the full binary tree decomposition, the authors calculate occurrence probabilities for each subtree and assign the corresponding Huffman code to it. Finally, the tree is encoded by traversing it and assigning the above codes. Now, the method to find and count respective subtrees is similar to counting words in texts. Specifically, the authors traverse the input tree with BFS and build a suffix tree in the process where each node of a suffix tree corresponds to one specific subtree. A similar approach for compressing probabilistic graphs was described by Maniu et al. [305].

*8.1.4 Others.* Approximation algorithms for finding the best virtual-node hierarchical compression were proposed by Feder et al. [156]. They also illustrated that the optimal compression of this type is NP-hard. Other works include compression used for obtaining better clustering [115, 322], using quadtrees to compress adjacency matrices [99], compressing graphs that model automata [319], partitioning an input graph and compressing each partition independently [142].

### 8.2 Compression for More Efficient Computation

Here, we outline works that specifically use compression for faster graph algorithms. These schemes are different from the ones described in the section devoted to problem-aware graph compression (§ 10.1) because they do not propose novel compression but discuss how to use existing compression schemes for faster graph algorothms. Liakos et al. [279] use various compression techniques (bit vectors and different types of coding techniques) in distributed-memory graph processing engines to reduce the pressure on the memory subsystem and thus accelerate processing. Next, Granskog and Striger analyze whether graph traversal algorithms (BFS, DFS) can be accelerated by using compression methods such as $k^2$-trees [193]. Other works use compression as one of the tools for better data mining capabilities [159], faster queries on graphs [326], accelerating subgraph matching by reducing the size of sets that contain matching candidates [361], or solving bin packing problems more efficiently [68].

*8.2.1 Compression in Graph Processing Engines.* Some works specifically discuss how to accelerate a given graph processing engine with compression. Shun et al. [392, 392] developed Ligra+, an enhancement over the Ligra graph processing engine [391] that uses parallelism to accelerate compression and decompression of graph data and thus amortize the costs of utilizing compressed graph representations while reducing the pressure on the memory subsystem. Other works that use parallelization to accelerate compression also exist [149]. Furthermore, Chen et al. [101] used compressed graphs with a generic topological OLAP framework in online graph analysis. Another paper [4] uses various vertex relabelings for more compression friendly graph layout within EmptyHeaded, an engine that outperforms standard OLAP systems. Chavan conducted an empirical study on graph compression in engines such as Pregel [100, 301].

### 8.3 Vertex Coding

Certain papers from 60s are tentatively connected to graph compression. Specifically, Breuer and Folkman [70, 71] analyzed *coding vertices*, i.e., assigning each vertex a unique binary code that is always smaller than a certain constant if two vertices are connected, and is always larger than this constant if two vertices are not connected. Using these coding schemes, the adjacency of any two nodes can be determined by the Hamming distance of their labels and may have applications in the domain of implicit graph representations (§ 9.2).

## 9 RELATED DOMAINS COVERED IN SURVEYS

We now mention works and surveys covering areas that are related to lossless graph compression. First, there are works on compressing graphs with the purpose of **more effective visualization**, for example Dwyer et al.'s [147]. They were partially covered in another survey [291]. Second, **compression of meshes** was covered extensively in several surveys [300, 346, 403]. Third, **compression of trees** is outside the scope of this work. It is partially covered in other works [242].

### 9.1 Lossless Summarization of Graphs

*Summarization of graphs* is an area where input graph data is *summarized* in order to provide a smaller graph description that may focus on some particular graph aspects [30, 249–252, 292, 371, 386]. These works were covered in a survey [291]. The most important connection to graph compression is that in many of these schemes, the process of graph summarization also leads to size reduction. For example, vertices within a cluster are grouped to form a supervertex, and edges are merged into superedges [44, 260, 261, 272, 289, 290, 343, 370, 410, 427, 431, 439, 442], similarly to many hierarchical schemes in web graphs (§ 4.1.5). Some works use or discuss bisimulation, especially in the domain of RDF graphs [89, 95]. In addition, various works use summarization to better understand the structure of graphs in domains such as biology [331, 411] or independently of a specific domain [7]. Moreover, there are works dedicated to the **summarization of dynamic graphs** [248, 362, 414].

### 9.2 Efficient and Implicit Graph Representations

Intuitively, *efficient* (in many cases also called *implicit*) graph representations provide vertex labels that encode the structure of the input graph so that no additional structure dedicated to storing edges is required. For example, Kannan and Naor in their seminal work [238] assign $O(\log n)$ bit labels to vertices such that these labels completely encode the structure of the graph. Thus, no additional data structure that determines edges is required. In addition, given the labels of any two vertices, the authors show that one can test if the vertices are adjacent in time linear in the size of the labels. Many other such schemes exist [15, 344, 382, 404]. Another thread of related work are algorithms for efficient derivation of such representations [26]. Now, such representations are covered extensively in a book by Spinrad [398]. Since the book was published, more such representations were discovered [97, 129, 177, 405].

**Terminology Clarification** We now clarify a certain terminology issue. An *implicit* graph representation as described above is a representation where vertex labels themselves *encode the information on edges between vertices*. Now, the term *implicit* is used in another context in the literature [137]. It describes a representation of an arbitrary data that is a constant additive factor away from the storage lower bound for this data. Formally, if the optimum to store some data is $\mathcal{N}$ bits, an implicit representation takes $\mathcal{N} + O(1)$ bits [137]. We do not know of any graph representations that are implicit in the second sense.

## 10 TAXONOMY AND DISCUSSION OF FEATURES

We now group and discuss graph compression schemes based on selected common *features* for better understanding of lossless graph compression.

### 10.1 Problem-Aware Graph Compression

First, we discuss schemes that, despite applying compression, still allow to obtain selected graph properties or solve selected graph problems fast. Sadri et al. [377] propose Shrink, a compression scheme that preserves distances between vertices. The compression proceeds in steps, in each step it iteratively merges vertices. During each merging, a system of linear equations is solved to define new edge weights to minimize changes in the distances. Merging continues until a specified number of vertices is reached. Moreover, Fan et al. [152, 153] develop compression strategies that preserve high performance and losslessness for two classes of graph queries: reachability and graph pattern queries via (bounded) simulation. Next, Hernandez discusses application-driven graph representations and compression [205]. Another similar work (performed for general graphs) that considers algorithms for all-pairs shortest paths, bipartite matching, and edge and vertex connectivity was conducted by Feder and Motwani [157, 158]. Finally, most of succinct and compact graph representations provide graph queries that ensure a specific time complexity, most often constant-time or logarithmic, see § 6.

### 10.2 Compression of Dynamic Graphs

We separately discuss compressing *dynamic* graphs (in the literature, they are also called *temporal*, *evolving*, or *time-evolving*). Brodal and Fagerberg [80] present a linear space graph data structure for graphs with bounded arboricity (example such graphs are planar graphs or bounded-treewidth graphs) under insertions, edge deletions, and adjacency queries. The proposed representation is the adjacency list representation, augmented with a simple scheme that maintains the structure under graph modifications. The core idea in proving the stated time bounds (constant-time adjacency queries in a graph with bounded arboricity $c$) is to reduce this problem to a simpler problem of assigning directions to edges (i.e., constructing a directed graph out of the input undirected one) so that all vertices have outdegree $O(c)$.

Iverson and Karypis [219] propose five data structures for representing dynamic sparse graphs. Their structures offer different trade-offs between size and speed of provided graph operations. The structures are: Linked-List (LL), Batch Compressed Sparse Row (BCSR), Dynamic Adjacency Array (DAA), Dynamic Intervalized Adjacency Array (DIAA), and Dynamic Compressed Adjacency Array (DCAA). LL is based on a simple set of linked lists with one list being responsible for one vertex neighborhood. BCSR is essentially an LL, but when a size of a linked list grows too large, it is resized into a static CSR. In DAA, each neighborhood is a dynamically allocated array that must be resized if there are updates. DIAA is essentially a DAA enhanced with storing contiguous vertices as intervals. Finally, DCAA leverages ideas from the WebGraph framework (§ 4.1.4).

Other works on dynamic graphs exist, for example Boldi et al. [59] analyzes how the web graph evolves and how its respective snapshots can be compressed, Caro et al. [90] design compact graph representations that enable answering queries fast, and Klitzke and Nicholson [256] mention compressing dynamic graphs as a part of their general framework for managing dynamic succinct data structures.

Finally, several dynamic schemes were described in the other sections of this survey, for example in the parts devoted to summarizing dynamic graphs [248, 362, 386, 414] (§ 9.1), compressing RDF graphs (§ 4.4), De Bruijn graphs [45] (§ 4.3.1), graph databases [265] (§ 5), succinct data structures [406] (§ 6), and others [76, 320, 383]. We list them to facilitate navigating the survey and refer the reader to these specific sections for more information.

Dynamic graphs are also considered in streaming settings; we discuss this separately in § 10.3.

*10.2.1 Viewing Graphs As Tensors.* We separately discuss works that add more dimensions to its adjacency matrix to model changes. Caro et al. [91] represent dynamic (*temporal*) graphs using 4-dimensional binary tensors. Two dimensions are used to model edges and two other dimensions model time intervals where a given edge exists. Then, they propose to compress such a representation with a generalization of $k^2$-trees [78] (see § 4.1.3) to a $d$-dimensional space, called $k^d$-tree. The key idea is similar to that of simple $k^2$-trees, namely, parts of a $d$-dimensional tensor with zeros in cells are compressed with internal nodes of a $k^d$-tree while tree leaves represent parts of the tensor that have more than one non-zero cell. Related approaches based on viewing a dynamic graph as a 4-dimensional object were discussed by Brisaboa, Bernardo, Caro, and others [75, 131].

### 10.3 Compression of Graphs in Streaming Settings

Various compression schemes are designed for streaming settings [139, 246, 247, 285, 332, 337, 385, 407, 408, 437]. For example, Nelson et al. [332] use quadtrees to compress graph streams.

## 11 CONCLUSION

Graph compression is an important area of research as it can be used to accelerate numerous modern graph workloads by reducing the amount of transferred data. Yet, it is a diverse set of fields driven by different communities, with a plethora of techniques, algorithms, domains, and approaches. We present the first survey that analyzes the rich world of lossless graph compression. We do not only list and categorize the existing work, but also provide key ideas, insights, and discuss formal underpinning of selected works. Our work can be used by architects and developers willing to select the best compression scheme in a given setting, graph theoreticians aiming to understand the high-level view of lossless graph compression, and anyone who wants to deepen their knowledge of this fascinating field.

## REFERENCES

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*, pages 411–422. VLDB Endowment, 2007.

[2] E. Abbe. Graph compression: The effect of clusters. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pages 1–8. IEEE, 2016.

[3] D. J. Abel. A b+-tree structure for large quadtrees. *Computer Vision, Graphics, and Image Processing*, 27(1):19–31, 1984.

[4] C. R. Aberger, A. Nötzli, K. Olukotun, and C. Ré. Emptyheaded: boolean algebra based graph processing. *ArXiv e-prints*, 2015.

[5] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference, 2001. Proceedings. DCC 2001.*, pages 203–212. IEEE, 2001.

[6] R. Agarwal, A. Khandelwal, and I. Stoica. Succinct: Enabling queries on compressed data. In *NSDI*, pages 337–350, 2015.

[7] S. E. Ahnert. Generalised power graph compression reveals dominant relationship patterns in complex networks. *Scientific reports*, 4:4385, 2014.

[8] L. C. Aleardi, O. Devillers, and A. Mebarki. 2d triangulation representation using stable catalogs. In *Proc. of CCCG*, pages 71–74, 2006.

[9] L. C. Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *Workshop on Algorithms and Data Structures*, pages 134–145. Springer, 2005.

[10] L. C. Aleardi, O. Devillers, and G. Schaeffer. *Compact representation of triangulations*. PhD thesis, INRIA, 2006.

[11] L. C. Aleardi, O. Devillers, and G. Schaeffer. Succinct representations of planar maps. *Theoretical Computer Science*, 408(2):174–187, 2008.

[12] F. Almodaresi, P. Pandey, and R. Patro. Rainbowfish: A succinct colored de bruijn graph representation. *bioRxiv*, page 138016, 2017.

[13] F. Almodaresi, H. Sarkar, and R. Patro. A space and time-efficient index for the compacted colored de bruijn graph. *bioRxiv*, page 191874, 2017.

[14] M. Almutairy, J. Fish, and C. T. Brown. Space-efficient read indexing and retrieval based on compressed de bruijn graphs. In *Computational Advances in Bio and Medical Sciences (ICCABS), 2013 IEEE 3rd International Conference on*, pages 1–1. IEEE, 2013.

[15] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 53–62. IEEE, 2002.

[16] S. Álvarez, N. R. Brisaboa, S. Ladra, and Ó. Pedreira. A compact representation of graph databases. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 18–25. ACM, 2010.

[17] S. Álvarez García. Compact and efficient representations of graphs. 2014.

[18] S. Álvarez-García, N. Brisaboa, J. D. Fernández, M. A. Martínez-Prieto, and G. Navarro. Compressed vertical partitioning for efficient rdf management. *Knowledge and Information Systems*, 44(2):439–474, 2015.

[19] S. Álvarez-García, N. R. Brisaboa, J. D. Fernández, and M. A. Martínez-Prieto. Compressed k2-triples for full-in-memory rdf engines. *arXiv preprint arXiv:1105.4004*, 2011.

[20] S. Álvarez-García, N. R. Brisaboa, J. D. Fernández, M. A. Martínez-Prieto, and G. Navarro. Compressed vertical partitioning for full-in-memory rdf management. *arXiv preprint arXiv:1310.4954*, 2013.

[21] S. Álvarez-García, N. R. Brisaboa, C. Gómez-Pantoja, and M. Marin. Distributed query processing on compressed graphs using k2-trees. In *International Symposium on String Processing and Information Retrieval*, pages 298–310. Springer, 2013.

[22] S. Alvarez-Garcia, G. de Bernardo, N. R. Brisaboa, and G. Navarro. A succinct data structure for self-indexing ternary relations. *Journal of Discrete Algorithms*, 43:38–53, 2017.

[23] E. Angelino. Compressing graphs with semantic structure.

[24] V. N. Anh and A. Moffat. Local modeling for webgraph compression. In *Data Compression Conference (DCC), 2010*, pages 519–519. IEEE, 2010.

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

[26] S. R. Arikati, A. Maheshwari, and C. D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discrete Applied Mathematics*, 78(1-3):1–16, 1997.

[27] A. R. Asadi, E. Abbe, and S. Verdú. Compressing data on graphs with clusters. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 1583–1587. IEEE, 2017.

[28] Y. Asano, T. Ito, H. Imai, M. Toyoda, and M. Kitsuregawa. Compact encoding of the web graph exploiting various power laws. In *International Conference on Web-Age Information Management*, pages 37–46. Springer, 2003.

[29] Y. Asano, Y. Miyawaki, and T. Nishizeki. Efficient compression of web graphs. In *International Computing and Combinatorics Conference*, pages 1–11. Springer, 2008.

[30] N. Ashrafi Payaman and M. Kangavari. Graph hybrid summarization. *Journal of AI and Data Mining*, 2017.

[31] M. Atre. *Bit-by-bit: Indexing and querying RDF data using compressed bit-vectors*. PhD thesis, Rensselaer Polytechnic Institute, 2011.

[32] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix bit loaded: a scalable lightweight join query processor for rdf data. In *Proceedings of the 19th international conference on World wide web*, pages 41–50. ACM, 2010.

[33] B. T. Ávila and R. D. Lins. W-tree: A compact external memory representation for webgraphs. *ACM Transactions on the Web (TWEB)*, 10(1):6, 2016.

[34] F. Ay, M. Dang, and T. Kahveci. Metabolic network alignment in large scale by network compression. *BMC bioinformatics*, 13(3):S2, 2012.

[35] U. Baier, T. Beller, and E. Ohlebusch. Graphical pan-genome analysis with compressed suffix trees and the burrows–wheeler transform. *Bioinformatics*, 32(4):497–504, 2015.

[36] A. T. Balaban. Applications of graph theory in chemistry. *Journal of chemical information and computer sciences*, 25(3):334–343, 1985.

[37] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

[38] J. Barbay, L. C. Aleardi, M. He, and J. I. Munro. Succinct representation of labeled graphs. *Algorithmica*, 62(1-2):224–257, 2012.

[39] J. Barbay, L. Castelli Aleardi, M. He, and J. Munro. Succinct representation of labeled graphs. In T. Tokuyama, editor, *Algorithms and Computation*, volume 4835 of *Lecture Notes in Computer Science*, pages 316–328. Springer Berlin Heidelberg, 2007.

[40] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theoretical Computer Science*, 387(3):284–297, 2007.

[41] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 680–689. Society for Industrial and Applied Mathematics, 2007.

[42] H. R. Bazoobandi, S. de Rooij, J. Urbani, A. ten Teije, F. van Harmelen, and H. Bal. A compact in-memory dictionary for rdf data. In *European Semantic Web Conference*, pages 205–220. Springer, 2015.

[43] D. Beckett, T. Berners-Lee, and E. Prud'hommeaux. Turtle-terse rdf triple language. *W3C Team Submission*, 14(7), 2008.

[44] Y. Bei, Z. Lin, and D. Chen. Summarizing scale-free networks based on virtual and real links. *Physica A: Statistical Mechanics and its Applications*, 444:360–372, 2016.

[45] D. Belazzougui, T. Gagie, V. Mäkinen, and M. Previtali. Fully dynamic de bruijn graphs. In *International Symposium on String Processing and Information Retrieval*, pages 145–152. Springer, 2016.

[46] D. Belazzougui, T. Gagie, V. Mäkinen, M. Previtali, and S. J. Puglisi. Bidirectional variable-order de bruijn graphs. In *Latin American Symposium on Theoretical Informatics*, pages 164–178. Springer, 2016.

[47] K. Belk, C. Boucher, A. Bowe, T. Gagie, P. Morley, M. D. Muggli, N. R. Noyes, S. J. Puglisi, and R. Raymond. Succinct colored de bruijn graphs. *bioRxiv*, page 040071, 2016.

[48] T. Beller and E. Ohlebusch. Efficient construction of a compressed de bruijn graph for pan-genome analysis. In *Annual Symposium on Combinatorial Pattern Matching*, pages 40–51. Springer, 2015.

[49] T. Beller and E. Ohlebusch. A representation of a compressed de bruijn graph for pan-genome analysis that enables search. *Algorithms for Molecular Biology*, 11(1):20, 2016.

[50] G. Benoit, C. Lemaitre, D. Lavenier, E. Drezen, T. Dayris, R. Uricaru, and G. Rizk. Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph. *BMC bioinformatics*, 16(1):288, 2015.

[51] G. Benoit, C. Lemaitre, D. Lavenier, and G. Rizk. Compression of high throughput sequencing data with probabilistic de bruijn graph. *arXiv preprint arXiv:1412.5932*, 2014.

[52] F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[53] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the web. *Computer networks and ISDN Systems*, 30(1-7):469–477, 1998.

[54] D. K. Blandford, G. E. Blelloch, and I. A. Kash. Compact representations of separable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 679–688, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[55] G. E. Blelloch and A. Farzan. Succinct representations of separable graphs. In *Annual Symposium on Combinatorial Pattern Matching*, pages 138–150. Springer, 2010.

[56] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[57] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World wide web*, pages 587–596. ACM, 2011.
[58] P. Boldi and M. Santini. Compressing social networks by community detection. Technical report, Technical Report RI-DSI 330-10. Dipartimento di Scienze dellInformazione, 2010.
[59] P. Boldi, M. Santini, and S. Vigna. A large time-aware web graph. In *ACM SIGIR Forum*, volume 42, pages 33–38. ACM, 2008.
[60] P. Boldi, M. Santini, and S. Vigna. Permuting web and social graphs. *Internet Mathematics*, 6(3):257–283, 2009.
[61] P. Boldi, M. Santini, and S. Vigna. Permuting web graphs. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 116–126. Springer, 2009.
[62] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004.
[63] P. Boldi and S. Vigna. Webgraph: Things you thought you could not do with java™. In *Proceedings of the 3rd international symposium on Principles and practice of programming in Java*, pages 1–8. Trinity College Dublin, 2004.
[64] P. Boldi and S. Vigna. Codes for the world wide web. *Internet mathematics*, 2(4):407–429, 2005.
[65] A. Borici and A. Thomo. Semantic graph compression with hypergraphs. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 1097–1104. IEEE, 2014.
[66] C. Boucher, A. Bowe, T. Gagie, S. J. Puglisi, and K. Sadakane. Variable-order de bruijn graphs. In *Data Compression Conference (DCC), 2015*, pages 383–392. IEEE, 2015.
[67] A. Bowe, T. Onodera, K. Sadakane, and T. Shibuya. Succinct de bruijn graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer, 2012.
[68] F. Brandao and J. P. Pedroso. Bin packing and related problems: general arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.
[69] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (amg) for automatic multigrid solutions with application to geodetic computations. *Report, Inst. for computational Studies, Fort collins, colo*, 1982.
[70] M. Breuer. Coding the vertexes of a graph. *IEEE transactions on Information Theory*, 12(2):148–153, 1966.
[71] M. A. Breuer and J. Folkman. An unexpected result in coding the vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20(3):583–600, 1967.
[72] L. Breyer. Web graph compression in markovpr 1.1. 2002.
[73] S. Brin and L. Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.
[74] N. R. Brisaboa, R. Cánovas, F. Claude, M. A. Martínez-Prieto, and G. Navarro. Compressed string dictionaries. In *International Symposium on Experimental Algorithms*, pages 136–147. Springer, 2011.
[75] N. R. Brisaboa, D. Caro, A. Fariña, and M. A. Rodríguez. A compressed suffix-array strategy for temporal-graph indexing. In *International Symposium on String Processing and Information Retrieval*, pages 77–88. Springer, 2014.
[76] N. R. Brisaboa, A. Cerdeira-Pena, G. de Bernardo, and G. Navarro. Compressed representation of dynamic binary relations with applications. *Information Systems*, 69:106–123, 2017.
[77] N. R. Brisaboa, A. Cerdeira-Pena, A. Farina, and G. Navarro. A compact rdf store using suffix arrays. In *International Symposium on String Processing and Information Retrieval*, pages 103–115. Springer, 2015.
[78] N. R. Brisaboa, S. Ladra, and G. Navarro. k2-trees for compact web graph representation. In *International Symposium on String Processing and Information Retrieval*, pages 18–30. Springer, 2009.
[79] N. R. Brisaboa, S. Ladra, and G. Navarro. Compact representation of web graphs with extended functionality. *Information Systems*, 39:152–174, 2014.
[80] G. Brodal and R. Fagerberg. Dynamic representations of sparse graphs. *Algorithms and Data Structures*, pages 773–773, 1999.

[81] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[82] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *International semantic web conference*, pages 54–68. Springer, 2002.

[83] J. A. Brown, S. Houghtent, T. K. Collins, and Q. Qu. Evolving graph compression using similarity measures for bioinformatics applications. In *CIBCB*, pages 1–6, 2016.

[84] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 95–106. ACM, 2008.

[85] M. Burger. *Memory-Efficient and Parallel Simulation of Super Carbon Nanotubes*. PhD thesis, Technische Universität, 2017.

[86] M. Burger, C. Bischof, and J. Wackerfuß. Compressed symmetric graphs for the simulation of super carbon nanotubes. In *High Performance Computing & Simulation (HPCS), 2016 International Conference on*, pages 286–293. IEEE, 2016.

[87] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. 1994.

[88] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe. Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome research*, 18(5):810–820, 2008.

[89] S. Campinas. *Graph summarisation of web data: data-driven generation of structured representations*. PhD thesis, 2016.

[90] D. Caro, M. A. Rodríguez, and N. R. Brisaboa. Data structures for temporal graphs based on compact sequence representations. *Information Systems*, 51:1–26, 2015.

[91] D. Caro, M. A. Rodríguez, N. R. Brisaboa, and A. Fariña. Compressed kd-tree for temporal graphs. *Knowledge and Information Systems*, 49(2):553–595, 2016.

[92] B. Cazaux, T. Lecroq, and E. Rivals. From indexing data structures to de bruijn graphs. In *CPM*, pages 89–99, 2014.

[93] B. Cazaux, T. Lecroq, and E. Rivals. Linking indexing data structures to de bruijn graphs: Construction and update. *Journal of Computer and System Sciences*, 2016.

[94] B. Cazaux, G. Sacomoto, and E. Rivals. Superstring graph: a new approach for genome assembly. In *International Conference on Algorithmic Applications in Management*, pages 39–52. Springer, 2016.

[95] Š. Čebirić, F. Goasdoué, and I. Manolescu. Query-oriented summarization of rdf graphs. *Proceedings of the VLDB Endowment*, 8(12):2012–2015, 2015.

[96] M. J. Chaisson and P. A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome research*, 18(2):324–330, 2008.

[97] M. Chandoo. On the implicit graph conjecture. *arXiv preprint arXiv:1603.01977*, 2016.

[98] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. Von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *Acm Sigplan Notices*, volume 40, pages 519–538. ACM, 2005.

[99] A. Chatterjee, M. Levan, C. Lanham, M. Zerrudo, M. Nelson, and S. Radhakrishnan. Exploiting topological structures for graph compression based on quadtrees. In *Research in Computational Intelligence and Communication Networks (ICRCICN), 2016 Second International Conference on*, pages 192–197. IEEE, 2016.

[100] A. CHAVAN. An introduction to graph compression techniques for in-memory graph computation.

[101] C. Chen, X. Yan, F. Zhu, J. Han, and S. Y. Philip. Graph olap: Towards online analytical processing on graphs. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 103–112. IEEE, 2008.

[102] S. Chen and J. H. Reif. Efficient lossless compression of trees and graphs. In *Proceedings of the Conference on Data Compression*, DCC '96, pages 428–, Washington, DC, USA, 1996. IEEE Computer Society.

[103] A. Cheng and P. Dickinson. Exploiting graph compression techniques for coding and monitoring of networks. In *Communications (ICC), 2014 IEEE International Conference on*, pages 1173–1178. IEEE, 2014.

[104] L. Cheng, A. Malik, S. Kotoulas, T. E. Ward, and G. Theodoropoulos. Efficient parallel dictionary encoding for rdf data. In *in Proc. 17th Int. Workshop on the Web and Databases*. Citeseer, 2014.

[105] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 506–515, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[106] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228. ACM, 2009.

[107] F. Chierichetti, R. Kumar, S. Lattanzi, A. Panconesi, and P. Raghavan. Models for the compressible web. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 331–340. IEEE, 2009.

[108] R. Chikhi, A. Limasset, S. Jackman, J. T. Simpson, and P. Medvedev. On the representation of de bruijn graphs. In *RECOMB*, volume 8394, pages 35–55. Springer, 2014.

[109] R. Chikhi, A. Limasset, and P. Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

[110] R. Chikhi and G. Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):22, 2013.

[111] Y. Choi. Fast algorithm for optimal compression of graphs. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 34–46. SIAM, 2010.

[112] Y. Choi and W. Szpankowski. Compression of graphical structures. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 364–368. IEEE, 2009.

[113] Y. Choi and W. Szpankowski. Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Transactions on Information Theory*, 58(2):620–638, 2012.

[114] R. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *Automata, Languages and Programming*, pages 118–129, 1998.

[115] R. Cilibrasi and P. M. Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[116] F. Claude and S. Ladra. Practical representations for web and social graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1185–1190. ACM, 2011.

[117] F. Claude and G. Navarro. A fast and compact web graph representation. In *International Symposium on String Processing and Information Retrieval*, pages 118–129. Springer, 2007.

[118] F. Claude and G. Navarro. Extended compact web graph representations. In *Algorithms and Applications*, pages 77–91. Springer, 2010.

[119] C. Clos. A study of non-blocking switching networks. *Bell Labs Technical Journal*, 32(2):406–424, 1953.

[120] R. F. Cohen, G. Di Battista, A. Kanevsky, and R. Tamassia. Reinventing the wheel: An optimal data structure for connectivity queries. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 194–200, New York, NY, USA, 1993. ACM.

[121] S. Cohen. Data management for social networking. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 165–177. ACM, 2016.

[122] T. K. Collins, A. Zakirov, J. A. Brown, and S. Houghten. Single-objective and multi-objective genetic algorithms for compression of biological networks. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2017 IEEE Conference on*, pages 1–8. IEEE, 2017.

[123] E. S. Committee et al. Edif electronic design interchange format version 2 0 0. *Electronic Industries Association*, 1987.

[124] P. E. Compeau, P. A. Pevzner, and G. Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

[125] T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 2011.

[126] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[127] M. R. Crusoe, H. F. Alameldin, S. Awad, E. Boucher, A. Caldwell, R. Cartwright, A. Charbonneau, B. Constantinides, G. Edvenson, S. Fay, et al. The khmer software package: enabling efficient nucleotide

sequence analysis. *F1000Research*, 4, 2015.
[128] O. Curé, G. Blin, D. Revuz, and D. C. Faye. Waterfowl: A compact, self-indexed and inference-enabled immutable rdf store. In *European Semantic Web Conference*, pages 302–316. Springer, 2014.
[129] A. R. Curtis, C. Izurieta, B. Joeris, S. Lundberg, and R. M. McConnell. An implicit representation of chordal comparability graphs in linear time. *Discrete Applied Mathematics*, 158(8):869–875, 2010.
[130] O. Dawelbeit and R. McCrindle. Efficient dictionary compression for processing rdf big data using google bigquery. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.
[131] G. De Bernardo, N. R. Brisaboa, D. Caro, and M. A. Rodríguez. Compact data structures for temporal graphs. In *Data Compression Conference (DCC), 2013*, pages 477–477. IEEE, 2013.
[132] G. de Bernardo Roca. *New data structures and algorithms for the efficient management of large spatial datasets*. PhD thesis, PhD thesis, Universidade da Coruña, 2014.
[133] F. de Bruijn. A combinatorial problem. 1946.
[134] R. De La Briandais. File searching using variable length keys. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 295–298. ACM, 1959.
[135] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[136] M. Dehmer and A. Mowshowitz. A history of graph entropy measures. *Information Sciences*, 181(1):57–78, 2011.
[137] E. Demaine. Advanced Data Structures, 2012. Lecture Notes.
[138] N. B. Déme, A. F. Dia, A. Boly, Z. Kazi-Aoul, and R. Chiky. An efficient approach for real-time processing of rdsz-based compressed rdf streams. In *Software Engineering Research, Management and Applications*, pages 147–166. Springer, 2018.
[139] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. *ACM Transactions on Algorithms (TALG)*, 6(1):6, 2009.
[140] N. Deo and B. Litow. A structural approach to graph compression. In *Proc. of the 23th MFCS Workshop on Communications*, pages 91–101. Citeseer, 1998.
[141] L. P. Deutsch. Gzip file format specification version 4.3. 1996.
[142] M. Dhabu, P. Deshpande, and S. Vishwakarma. Partition based graph compression. *Editorial Preface*, 4(9), 2013.
[143] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. *arXiv preprint arXiv:1602.08820*, 2016.
[144] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.
[145] K. Dinkla, M. A. Westenberg, and J. J. van Wijk. Compressed adjacency matrices: Untangling gene regulatory networks. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2457–2466, 2012.
[146] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1-3):221–256, 2004.
[147] T. Dwyer, N. H. Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE transactions on visualization and computer graphics*, 19(12):2596–2605, 2013.
[148] P. Elias. Universal codeword sets and representations of the integers. *IEEE transactions on information theory*, 21(2):194–203, 1975.
[149] E. En, A. Alam, K. U. Khan, and Y.-K. Lee. Parallel compression of weighted graphs. In *Proceedings of the 7th International Conference on Emerging Databases*, pages 68–77. Springer, 2018.
[150] P. Erdos and A. Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4):343–347, 1961.
[151] F. Esposito, L. Iannone, I. Palmisano, D. Redavid, and G. Semeraro. Redd: An algorithm for redundancy detection in rdf models. In *ESWC*, pages 138–152. Springer, 2005.
[152] W. Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21. ACM, 2012.
[153] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 157–168. ACM, 2012.

[154] A. Farzan and J. I. Munro. Succinct representations of arbitrary graphs. In *European Symposium on Algorithms*, pages 393–404. Springer, 2008.

[155] A. Farzan and J. I. Munro. Succinct Encoding of Arbitrary Graphs. *Theor. Comput. Sci.*, 513:38–52, Nov. 2013.

[156] T. Feder, A. Meyerson, R. Motwani, L. O'Callaghan, and R. Panigrahy. Representing graph metrics with fewest edges. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 355–366. Springer, 2003.

[157] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 123–133. ACM, 1991.

[158] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, 1995.

[159] J. Feng, X. He, N. Hubig, C. Bohm, and C. Plant. Compression-based graph mining exploiting structure primitives. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 181–190. IEEE, 2013.

[160] J. D. Fernández. *Compact RDF Representations for Publishing and Exchanging in the Web of Data*. PhD thesis, University of Chile, Chile, 2011.

[161] J. D. Fernández, C. Gutierrez, and M. A. Martínez-Prieto. Rdf compression: basic approaches. In *Proceedings of the 19th international conference on World wide web*, pages 1091–1092. ACM, 2010.

[162] J. D. Fernández, A. Llaves, and O. Corcho. Efficient rdf interchange (eri) format for rdf data streams. In *International Semantic Web Conference*, pages 244–259. Springer, 2014.

[163] J. D. Fernández, M. A. Martínez-Prieto, M. Arias, C. Gutierrez, S. Álvarez-García, and N. R. Brisaboa. Lightweighting the web of data through compact rdf/hdt. In *CAEPIA*, pages 483–493. Springer, 2011.

[164] J. D. Fernández, M. A. Martínez-Prieto, and C. Gutierrez. Compact representation of large rdf data sets for publishing and exchange. In *International Semantic Web Conference*, pages 193–208. Springer, 2010.

[165] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.

[166] N. Fernández, J. Arias, L. Sánchez, D. Fuentes-Lorenzo, and Ó. Corcho. Rdsz: an approach for lossless rdf stream compression. In *European Semantic Web Conference*, pages 52–67. Springer, 2014.

[167] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM (JACM)*, 57(1):4, 2009.

[168] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[169] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):20, 2007.

[170] P. Ferragina, F. Piccinno, and R. Venturini. Compressed indexes for string searching in labeled graphs. In *Proceedings of the 24th International Conference on World Wide Web*, pages 322–332. International World Wide Web Conferences Steering Committee, 2015.

[171] P. Flick, C. Jain, T. Pan, and S. Aluru. A parallel connectivity algorithm for de bruijn graphs in metagenomic applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 15. ACM, 2015.

[172] E. Fusy, D. Poulalhon, and G. Schaeffer. Dissections and trees, with applications to optimal mesh encoding and to random sampling. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 690–699. Society for Industrial and Applied Mathematics, 2005.

[173] M. A. Gallego, O. Corcho, J. D. Fernández, M. A. Martínez-Prieto, and M. C. Suárez-Figueroa. Compressing semantic metadata for efficient multimedia retrieval. In *Conference of the Spanish Association for Artificial Intelligence*, pages 12–21. Springer, 2013.

[174] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.

[175] S. A. Garcia, N. R. Brisaboa, G. de Bernardo, and G. Navarro. Interleaved k2-tree: Indexing and navigating ternary relations. In *Data Compression Conference (DCC), 2014*, pages 342–351. IEEE, 2014.
[176] C. Gavoille and N. Hanusse. On compact encoding of pagenumber $k$ graphs. *Discrete Mathematics and Theoretical Computer Science*, 10(3):23–34, 2008.
[177] C. Gavoille and A. Labourel. Shorter implicit representation for planar graphs and bounded treewidth graphs. In *European Symposium on Algorithms*, pages 582–593. Springer, 2007.
[178] V. Gayathri and P. S. Kumar. Horn-rule based compression technique for rdf data. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 396–401. ACM, 2015.
[179] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick. Parallel de bruijn graph construction and traversal for de novo genome assembly. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 437–448. IEEE Press, 2014.
[180] P.-L. George and H. Borouchaki. Delaunay triangulation and meshing. 1998.
[181] M. GIATSOGLOU. *Managing Massive Graphs for the Web and Web 2.0*. PhD thesis, ARISTOTLE UNIVERSITY OF THESSALONIKI, 2010.
[182] S. Gieße. Pan-genomes and de bruijn graphs.
[183] A. C. Gilbert and K. Levchenko. Compressing network graphs. In *Proceedings of the LinkKDD workshop at the 10th ACM Conference on KDD*, volume 124, 2004.
[184] J. M. Giménez García et al. Scalable rdf compression with mapreduce and hdt. 2015.
[185] J. M. Giménez-García, J. D. Fernández, and M. A. Martínez-Prieto. Hdt-mr: A scalable solution for rdf compression with hdt and mapreduce. In *European Semantic Web Conference*, pages 253–268. Springer, 2015.
[186] S. Goldstein, A. Briska, S. Zhou, and D. Schwartz. Sequences, maps, genomes and graphs: Graph compression algorithms for efficiently comparing genomes. *UW Biostatistics and Medical Informatics Technical Report*, 181:1–11, 2004.
[187] S. Golomb. Run-length encodings (corresp.). *IEEE transactions on information theory*, 12(3):399–401, 1966.
[188] S. L. González. *Algorithms and compressed data structures for information retrieval*. PhD thesis, Universidade da Coruña, 2011.
[189] I. J. Good. Normal recurring decimals. *Journal of the London Mathematical Society*, 1(3):167–169, 1946.
[190] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, et al. Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, 29(7):644–652, 2011.
[191] S. Grabowski and W. Bieniecki. Tight and simple web graph compression. *arXiv preprint arXiv:1006.0809*, 2010.
[192] S. Grabowski and W. Bieniecki. Merging adjacency lists for efficient web graph compression. In *Man-Machine Interactions 2*, pages 385–392. Springer, 2011.
[193] T. Granskog and A. Strigér. A comparison of search times oncompressed and uncompressedgraphs, 2015.
[194] F. Gray. Pulse code communication, Mar. 17 1953. US Patent 2,632,058.
[195] S. Grimm and J. Wissmann. Elimination of redundancy in ontologies. In *Extended Semantic Web Conference*, pages 260–274. Springer, 2011.
[196] T. Guang, J. Gu, and L. Huang. Detect redundant rdf data by rules. In *International Conference on Database Systems for Advanced Applications*, pages 362–368. Springer, 2016.
[197] J.-L. Guillaume, M. Latapy, and L. Viennot. Efficient and simple encodings for the web graph. In *International Conference on Web-Age Information Management*, pages 328–337. Springer, 2002.
[198] B. Guler, A. Yener, P. Basu, C. Andersen, and A. Swami. A study on compressing graphical structures. In *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*, pages 823–827. IEEE, 2014.
[199] D. Hannah, C. Macdonald, and I. Ounis. Analysis of link graph compression techniques. In *European Conference on Information Retrieval*, pages 596–601. Springer, 2008.
[200] S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. 2003.

[201] M. Hayashida and T. Akutsu. Comparing biological networks via graph compression. *BMC systems biology*, 4(2):S13, 2010.

[202] X. He, M.-Y. Kao, and H.-I. Lu. Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM Journal on Discrete Mathematics*, 12(3):317–325, 1999.

[203] X. He, M.-Y. Kao, and H.-I. Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3):838–846, 2000.

[204] J. Hearn. Applications of kolmogorov complexity to graphs. 2006.

[205] C. Hernández and M. Marın. *Managing Massive Graphs*. PhD thesis, PhD thesis, universidad de chile, 2014. http://users. dcc. uchile. cl/˜ gnavarro/algoritmos/tesisCecilia. pdf, 2009.

[206] C. Hernández and G. Navarro. Compression of web and social graphs supporting neighbor and community queries. In *Proc. 5th ACM Workshop on Social Network Mining and Analysis (SNA-KDD). ACM*, 2011.

[207] C. Hernández and G. Navarro. Compressed representation of web and social networks via dense subgraphs. In *International Symposium on String Processing and Information Retrieval*, pages 264–276. Springer, 2012.

[208] C. Hernández and G. Navarro. Compressed representations for web and social graphs. *Knowledge and information systems*, 40(2):279–313, 2014.

[209] A. Hernández-Illera, M. A. Martínez-Prieto, and J. D. Fernández. Serializing rdf in compressed space. In *Data Compression Conference (DCC), 2015*, pages 363–372. IEEE, 2015.

[210] G. Holley, R. Wittler, and J. Stoye. Bloom filter trie–a data structure for pan-genome storage. In *International Workshop on Algorithms in Bioinformatics*, pages 217–230. Springer, 2015.

[211] A. Holzinger, B. Ofner, C. Stocker, A. C. Valdez, A. K. Schaar, M. Ziefle, and M. Dehmer. On graph entropy measures for knowledge discovery from publication network data. In *International Conference on Availability, Reliability, and Security*, pages 354–362. Springer, 2013.

[212] A. Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[213] M. Hosseini, D. Pratas, and A. J. Pinho. A survey on data compression methods for biological sequences. *Information*, 7(4):56, 2016.

[214] J. Huang, D. J. Abadi, and K. Ren. Scalable sparql querying of large rdf graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[215] L. Iannone, I. Palmisano, and D. Redavid. Optimizing rdf storage removing redundancies: An algorithm. *Innovations in Applied Artificial Intelligence*, pages 732–742, 2005.

[216] R. M. Idury and M. S. Waterman. A new algorithm for dna sequence assembly. *Journal of computational biology*, 2(2):291–306, 1995.

[217] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.

[218] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17(2):215–219, 1982.

[219] J. Iverson and G. Karypis. Storing dynamic graphs: Speed vs. storage trade-offs. 2014.

[220] S. D. Jackman, B. P. Vandervalk, H. Mohamadi, J. Chu, S. Yeo, S. A. Hammond, G. Jahesh, H. Khan, L. Coombe, R. L. Warren, et al. Abyss 2.0: resource-efficient assembly of large genomes using a bloom filter. *Genome research*, 27(5):768–777, 2017.

[221] G. Jacobson. Space-efficient static trees and graphs. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 549–554. IEEE, 1989.

[222] G. J. Jacobson. *Succinct Static Data Structures*. PhD thesis, Pittsburgh, PA, USA, 1988. AAI8918056.

[223] A. G. Jagalpure. *RGIS: Efficient Representation, Indexing and Querying of Large RDF Graphs*. PhD thesis, University of Georgia, 2012.

[224] M. Jaillard, M. Tournoud, L. Lima, V. Lacroix, J.-B. Veyrieras, and L. Jacob. Representing genetic determinants in bacterial gwas with compacted de bruijn graphs. *bioRxiv*, page 113563, 2017.

[225] X. Jiang, X. Zhang, F. Gao, C. Pu, and P. Wang. Graph compression strategies for instance-focused semantic mining. In *CSWS*, pages 50–61, 2013.

[226] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 595–608. ACM, 2008.

[227] D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 13–23. VLDB Endowment, 2004.

[228] A. K. Joshi. *Exploiting Alignments in Linked Data for Compression and Query Answering*. PhD thesis, Wright State University, 2017.

[229] A. K. Joshi, P. Hitzler, and G. Dong. Towards logical linked data compression. In *Proceedings of the Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web, LHD+ SemQuant2012, at the 11th International Semantic Web Conference, ISWC2012*, 2012.

[230] A. K. Joshi, P. Hitzler, and G. Dong. Logical linked data compression. In *Extended Semantic Web Conference*, pages 170–184. Springer, 2013.

[231] A. K. Joshi, P. Hitzler, and G. Dong. Alignment aware linked data compression. In *Joint International Semantic Technology Conference*, pages 73–81. Springer, 2015.

[232] J. Jusko, M. Rehak, and T. Pevny. A memory efficient privacy preserving representation of connection graphs. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, page 4. ACM, 2014.

[233] M. Kamal and M. I. Khan. Memory optimization for global protein network alignment using pushdown automata and de bruijn graph based bloom filter. *Journal of Software*, 9(10):2622–2628, 2014.

[234] U. Kang and C. Faloutsos. Beyond'caveman communities': Hubs and spokes for graph compression and mining. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 300–309. IEEE, 2011.

[235] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos. Gbase: a scalable and general graph management system. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1091–1099. ACM, 2011.

[236] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos. Gbase: an efficient analysis platform for large graphs. *The VLDB Journal—The International Journal on Very Large Data Bases*, 21(5):637–650, 2012.

[237] R. Kannan. Unraveling k-page graphs. *Information and control*, 66(1-2):1–5, 1985.

[238] S. Kannan, M. Naor, and S. Rudich. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

[239] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[240] M.-Y. Kao, N. Occhiogrosso, and S.-H. Teng. Simple and efficient graph compression schemes for dense and complement graphs. *Journal of Combinatorial Optimization*, 2(4):351–359, 1998.

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

[242] J. Katajainen and E. Mäkinen. Tree compression and optimization with applications. *International Journal of Foundations of Computer Science*, 1(04):425–447, 1990.

[243] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995.

[244] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, and P. Castagna. Jena-hbase: A distributed, scalable and efficient rdf triple store. In *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*, pages 85–88. CEUR-WS. org, 2012.

[245] H. Khalili, A. Yahyavi, and F. Oroumchian. Web-graph pre-compression for similarity based algorithms. 2009.

[246] A. Khan and C. Aggarwal. Query-friendly compression of graph streams. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 130–137. IEEE, 2016.

[247] A. Khan and C. Aggarwal. Toward query-friendly compression of rapid graph streams. *Social Network Analysis and Mining*, 7(1):23, 2017.

[248] A. Khan, S. S. Bhowmick, and F. Bonchi. Summarizing static and dynamic big graphs. *Proceedings of the VLDB Endowment*, 10(12):1981–1984, 2017.

[249] K. U. Khan, B. Dolgorsuren, T. N. Anh, W. Nawaz, and Y.-K. Lee. Faster compression methods for a weighted graph using locality sensitive hashing. *Information Sciences*, 2017.

[250] K. U. Khan, W. Nawaz, and Y.-K. Lee. Set-based unified approach for attributed graph summarization. In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 378–385. IEEE, 2014.

[251] K. U. Khan, W. Nawaz, and Y.-K. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[252] K. U. Khan, W. Nawaz, and Y.-K. Lee. Set-based unified approach for summarization of a multi-attributed graph. *World Wide Web*, 20(3):543–570, 2017.

[253] A. Khandelwal, Z. Yang, E. Ye, R. Agarwal, and I. Stoica. Zipg: A memory-efficient graph store for interactive queries. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1149–1164. ACM, 2017.

[254] D. King and J. R. Rossignac. Guaranteed 3.67 v bit encoding of planar triangle graphs. Technical report, Georgia Institute of Technology, 1999.

[255] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[256] P. Klitzke and P. K. Nicholson. A general framework for dynamic succinct and compressed data structures. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 160–173. SIAM, 2016.

[257] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[258] D. E. Knuth. The art of computer programming, volume 4, fascicle 2: Generating all tuples and permutations, 2005.

[259] J. Komlós and M. Simonovits. Szemerédi's regularity lemma and its applications in graph theory. 1996.

[260] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM, 2014.

[261] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Summarizing and understanding large graphs. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(3):183–202, 2015.

[262] L. G. Kraft. *A device for quantizing, grouping, and coding amplitude-modulated pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

[263] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[264] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Computer networks*, 31(11):1481–1493, 1999.

[265] A. G. Labouseur, J. Birnbaum, P. W. Olsen, S. R. Spillane, J. Vijayan, J.-H. Hwang, and W.-S. Han. The g* graph database: efficiently managing large distributed dynamic graphs. *Distributed and Parallel Databases*, 33(4):479–514, 2015.

[266] S. Ladra, J. R. Paramá, and F. Silva-Coira. Compact and queryable representation of raster datasets. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 15. ACM, 2016.

[267] S. Ladra, J. R. Paramá, and F. Silva-Coira. Scalable and queryable compressed storage structure for raster data. *Information Systems*, 2017.

[268] S. Ladra González. Algorithms and compressed data structures for information retrieval. 2011.

[269] R. Lamarche-Perrin, L. Tabourier, and F. Tarissan. Information-theoretic compression of weighted graphs. In *Poster session of the MSR-INRIA Join Center Workshop on Networks: Learning, Information and Complexity*, 2016.

[270] N. J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[271] N. L. Lashtaghani and S. Dehghanian. Large graphs compression using a combined algorithm. 2014.

[272] K. LeFevre and E. Terzi. Grass: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM, 2010.

[273] N. Lehmann and J. Pérez. Implementing graph query languages over compressed data structures: A progress report. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, page 96, 2015.

[274] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1):387–401, 2000.

[275] G. Li and W. Rao. Compression-aware graph computation. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1295–1302. ACM, 2016.

[276] G. Li, W. Rao, and Z. Jin. Efficient compression on real world directed graphs. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 116–131. Springer, 2017.

[277] M. Li and M. Paul. P. vit anyi, an introduction to kolmogorov complexity and its applications, 2008.

[278] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–272, 2010.

[279] P. Liakos, K. Papakonstantinopoulou, and A. Delis. Memory-optimized distributed graph processing through novel compression techniques. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2317–2322. ACM, 2016.

[280] P. Liakos, K. Papakonstantinopoulou, and M. Sioutis. On the effect of locality in compressing social networks. In *ECIR*, pages 650–655. Springer, 2014.

[281] P. Liakos, K. Papakonstantinopoulou, and M. Sioutis. Pushing the envelope in graph compression. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1549–1558. ACM, 2014.

[282] Y. Lim, U. Kang, and C. Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3077–3089, 2014.

[283] M. R. Limon, R. Sharker, S. Biswas, and M. S. Rahman. Efficient de bruijn graph construction for genome assembly using a hash table and auxiliary vector data structures. In *Computer and Information Technology (ICCIT), 2014 17th International Conference on*, pages 121–126. IEEE, 2014.

[284] Y. Lin and P. A. Pevzner. Manifold de bruijn graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 296–310. Springer, 2014.

[285] Y.-R. Lin, K. S. Candan, H. Sundaram, and L. Xie. Scent: Scalable compressed monitoring of evolving multirelational social networks. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 7(1):29, 2011.

[286] B. Litow and N. Deo. Graph compression and the zeros of polynomials. *Information processing letters*, 92(1):39–44, 2004.

[287] B. Litow, N. Deo, and A. Cami. Compression of vertex transitive graphs. *Congressus Numerantium*, 167:161, 2004.

[288] B. Liu, D. Zhu, and Y. Wang. debwt: parallel construction of burrows–wheeler transform for large collection of genomes with de bruijn-branch encoding. *Bioinformatics*, 32(12):i174–i182, 2016.

[289] S. Liu, Q. Zhao, J. Li, and W. Rao. Graph summarization based on attribute-connected network. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 161–171. Springer, 2017.

[290] X. Liu, Y. Tian, Q. He, W.-C. Lee, and J. McPherson. Distributed graph summarization. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 799–808. ACM, 2014.

[291] Y. Liu, A. Dighe, T. Safavi, and D. Koutra. A graph summarization: A survey. *arXiv preprint arXiv:1612.04883*, 2016.

[292] Y. Liu, T. Safavi, N. Shah, and D. Koutra. Reducing million-node graphs to a few structural patterns: A unified approach.

[293] H.-I. Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 223–224. Society for Industrial and Applied Mathematics, 2002.

[294] H.-I. Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. *SIAM Journal on Computing*, 43(2):477–496, 2014.
[295] T. Luczak, A. Magner, and W. Szpankowski. Structural information and compression of scale-free graphs. *Urbana*, 51:618015, 2017.
[296] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry. Challenges in Parallel Graph Processing. *Par. Proc. Let.*, 17(1):5–20, 2007.
[297] K. Lyko, M. Nitzschke, and A.-C. N. Ngomo. Scaro: Scalable rdf compression using rule subsumption hierarchies.
[298] A. Maccioni and D. J. Abadi. On compressing graph databases.
[299] A. Maccioni and D. J. Abadi. Scalable pattern matching over compressed graphs via dedensification. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1755–1764. ACM, 2016.
[300] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys (CSUR)*, 47(3):44, 2015.
[301] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. of the ACM SIGMOD Intl. Conf. on Manag. of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.
[302] S. Maneth and F. Peternek. A survey on methods and systems for graph compression. *arXiv preprint arXiv:1504.00616*, 2015.
[303] S. Maneth and F. Peternek. Compressing graphs by grammars. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 109–120. IEEE, 2016.
[304] S. Maneth and F. Peternek. Grammar-based graph compression. *arXiv preprint arXiv:1704.05254*, 2017.
[305] S. Maniu, R. Cheng, and P. Senellart. Compression of probabilistic graphs using tree decompositions. 2013.
[306] S. Marcus, H. Lee, and M. C. Schatz. Splitmem: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics*, 30(24):3476–3483, 2014.
[307] N. Martínez-Bazan, M. Á. Águila-Lorente, V. Muntés-Mulero, D. Dominguez-Sal, S. Gómez-Villamor, and J.-L. Larriba-Pey. Efficient graph management based on bitmap indices. In *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, pages 110–119. ACM, 2012.
[308] N. Martinez-Bazan, S. Gomez-Villamor, and F. Escale-Claveras. Dex: A high-performance graph database management system. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*, pages 124–127. IEEE, 2011.
[309] M. A. Martínez-Prieto, J. D. Fernández, and R. Cánovas. Compression of rdf dictionaries. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 340–347. ACM, 2012.
[310] M. A. Martínez-Prieto, J. D. Fernández, and R. Cánovas. Querying rdf dictionaries in compressed space. *ACM SIGAPP Applied Computing Review*, 12(2):64–77, 2012.
[311] H. Maserrat. *Compression of social networks*. PhD thesis, Applied Science: School of Computing Science, 2012.
[312] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 533–542. ACM, 2010.
[313] B. McBride. Jena: Implementing the rdf model and syntax specification. In *Proceedings of the Second International Conference on Semantic Web-Volume 40*, pages 23–28. CEUR-WS. org, 2001.
[314] M. Meier. Towards rule-based minimization of rdf graphs under constraints. In *International Conference on Web Reasoning and Rule Systems*, pages 89–103. Springer, 2008.
[315] M. Meilă. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, pages 577–584. ACM, 2005.
[316] C. Miao. *AN EXPERIMENTAL STUDY ON COMPRESSED REPRESENTATIONS OF WEB GRAPHS AND SOCIAL NETWORKS BASED ON DENSE SUBGRAPH EXTRACTION*. PhD thesis, 2016.
[317] L. Ming and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Heidelberg, 1997.

[318] I. Minkin, S. Pham, and P. Medvedev. Twopaco: An efficient algorithm to build the compacted de bruijn graph from many complete genomes. *Bioinformatics*, page btw609, 2016.
[319] M. Mohri, M. Riley, and A. T. Suresh. Automata and graph compression. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 2989–2993. IEEE, 2015.
[320] J. Mondal. *Real-time analytics on large dynamic graphs.* PhD thesis, University of Maryland, College Park, 2015.
[321] A. Mowshowitz and M. Dehmer. Entropy and the complexity of graphs revisited. *Entropy*, 14(3):559–570, 2012.
[322] N. S. Mueller, K. Haegler, J. Shao, C. Plant, and C. Böhm. Weighted graph compression for parameter-free clustering with pacco. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 932–943. SIAM, 2011.
[323] J. I. Munro. Succinct data structures. *Electr. Notes Theor. Comput. Sci.*, 91:3, 2004.
[324] J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 118–126. IEEE, 1997.
[325] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.
[326] C. Nabti and H. Seba. Querying massive graph data: A compress and search approach. *Future Generation Computer Systems*, 74:63–75, 2017.
[327] M. Naor. Succinct representation of general unlabeled graphs. *Discrete Applied Mathematics*, 28(3):303–307, 1990.
[328] G. Navarro. Compressing web graphs like texts. Technical report, Technical Report TR/DCC-2007-2, Dept. of Computer Science, University of Chile, 2007.
[329] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys (CSUR)*, 39(1):2, 2007.
[330] G. Navarro and K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms (TALG)*, 10(3):16, 2014.
[331] S. Navlakha, M. C. Schatz, and C. Kingsford. Revealing biological modules via graph summarization. *Journal of Computational Biology*, 16(2):253–264, 2009.
[332] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan. On compressing massive streaming graphs with quadtrees. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2409–2417. IEEE, 2015.
[333] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal—The International Journal on Very Large Data Bases*, 19(1):91–113, 2010.
[334] F. Nourbakhsh. Algorithms for graph compression: theory and experiments. 2015.
[335] F. Nourbakhsh, S. R. Bulò, and M. Pelillo. A matrix factorization approach to graph compression. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 76–81. IEEE, 2014.
[336] D. Okanohara and K. Sadakane. Practical Entropy-Compressed Rank/Select Dictionary. In *Proc. of ALENEX'07, ACM.* Press, 2007.
[337] C. A. Packer and L. B. Holder. Graphzip: Dictionary-based compression for mining graph streams. *arXiv preprint arXiv:1703.08614*, 2017.
[338] J. Z. Pan, J. Gómez-Pérez, Y. Ren, H. Wu, and M. Zhu. Ssp: Compressing rdf data by summarisation, serialisation and predictive encoding. Technical report, Technical report, 07 2014. Available as http://www. kdrive-project. eu/resources, 2014.
[339] J. Z. Pan, J. M. G. Pérez, Y. Ren, H. Wu, H. Wang, and M. Zhu. Graph pattern based rdf data compression. In *Joint International Semantic Technology Conference*, pages 239–256. Springer, 2014.
[340] P. Pandey, M. A. Bender, R. Johnson, and R. Patro. debgr: an efficient and near-exact representation of the weighted de bruijn graph. *Bioinformatics*, 33(14):i133–i141, 2017.
[341] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.
[342] I. Pavlov. 7zip file archive application. 2007.

[343] N. A. Payaman and M. Kangavari. Gssc: Graph summarization based on both structure and concepts. *International Journal of Information & Communication Technology Research*, 9(1):33–44, 2017.

[344] D. Peleg. Informative labeling schemes for graphs. In *International Symposium on Mathematical Foundations of Computer Science*, pages 579–588. Springer, 2000.

[345] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje, and C. T. Brown. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*, 109(33):13272–13277, 2012.

[346] J. Peng, C.-S. Kim, and C.-C. J. Kuo. Technologies for 3d mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6):688–733, 2005.

[347] Y. Peng, H. C. Leung, S.-M. Yiu, and F. Y. Chin. Idba–a practical iterative de bruijn graph de novo assembler. In *Annual international conference on research in computational molecular biology*, pages 426–440. Springer, 2010.

[348] Y. Peng, H. C. Leung, S.-M. Yiu, and F. Y. Chin. Meta-idba: a de novo assembler for metagenomic data. *Bioinformatics*, 27(13):i94–i101, 2011.

[349] Y. Peng, H. C. Leung, S.-M. Yiu, and F. Y. Chin. T-idba: a de novo iterative de bruijn graph assembler for transcriptome. In *International Conference on Research in Computational Molecular Biology*, pages 337–338. Springer, 2011.

[350] L. Peshkin. Structure induction by lossless graph compression. *arXiv preprint cs/0703132*, 2007.

[351] P. Peterlongo, N. Schnel, N. Pisanti, M.-F. Sagot, and V. Lacroix. Identifying snps without a reference genome by comparing raw reads. In *String Processing and Information Retrieval*, pages 147–158. Springer, 2010.

[352] J. Petit. Addenda to the survey of layout problems. *Bulletin of EATCS*, 3(105), 2013.

[353] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[354] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. Minimising rdf graphs under rules and constraints revisited. In *AMW*, 2010.

[355] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. Redundancy elimination on rdf graphs in the presence of rules, constraints, and queries. In *International Conference on Web Reasoning and Rule Systems*, pages 133–148. Springer, 2010.

[356] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. Complexity of redundancy detection on rdf graphs in the presence of rules, constraints, and queries. *Semantic Web*, 4(4):351–393, 2013.

[357] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. In *ICALP*, pages 1080–1094. Springer, 2003.

[358] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46(3):505–527, 2006.

[359] D. Pountain. Run-length encoding. *Byte*, 12(6):317–319, 1987.

[360] B. Pradhan, K. Sandeep, S. Mansor, A. Rahman Ramli, and A. R. B. M. Sharif. Second generation wavelets based gis terrain data compression using delaunay triangulation. *Engineering Computations*, 24(2):200–213, 2007.

[361] M. Qiao, H. Zhang, and H. Cheng. Subgraph matching: on compression and computation. *Proceedings of the VLDB Endowment*, 11(2), 2017.

[362] Q. Qu, S. Liu, F. Zhu, and C. S. Jensen. Efficient online summarization of large-scale dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3231–3245, 2016.

[363] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 405–416. IEEE, 2003.

[364] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[365] N. Rahman, R. Raman, et al. Engineering the louds succinct tree representation. In *International Workshop on Experimental and Efficient Algorithms*, pages 134–145. Springer, 2006.

[366] R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4):43, 2007.

[367] K. H. Randall, R. Stata, R. G. Wickremesinghe, and J. L. Wiener. The link database: Fast access to graphs of the web. In *Data Compression Conference, 2002. Proceedings. DCC 2002*, pages 122–131. IEEE, 2002.

[368] N. Rashevsky. Life, information theory, and topology. *Bulletin of Mathematical Biology*, 17(3):229–235, 1955.

[369] V. Rengasamy, P. Medvedev, and K. Madduri. Parallel and memory-efficient preprocessing for metagenome assembly. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pages 283–292. IEEE, 2017.

[370] M. Riondato, D. García-Soriano, and F. Bonchi. Graph summarization with quality guarantees. *Data Mining and Knowledge Discovery*, 31(2):314–349, 2017.

[371] C. P. H. RIVAS, G. N. BADINO, M. M. CAIHUAN, B. B. CÁRDENAS, J. P. ROJAS, and S. VIGNA. Managing massive graphs. 2014.

[372] E. A. Rødland. Compact representation of k-mer de bruijn graphs for genome read assembly. *BMC bioinformatics*, 14(1):313, 2013.

[373] P. Ronhovde and Z. Nussinov. Local resolution-limit-free potts model for community detection. *Physical Review E*, 81(4):046114, 2010.

[374] R. Rozov, G. Goldshlager, E. Halperin, and R. Shamir. Faucet: streaming de novo assembly graph construction. *bioRxiv*, page 125658, 2017.

[375] G. A. Sacomoto, J. Kielbassa, R. Chikhi, R. Uricaru, P. Antoniou, M.-F. Sagot, P. Peterlongo, and V. Lacroix. K is s plice: de-novo calling alternative splicing events from rna-seq data. *BMC bioinformatics*, 13(6):S5, 2012.

[376] K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48(2):294–313, 2003.

[377] A. Sadri, F. D. Salim, Y. Ren, M. Zameni, J. Chan, and T. Sellis. Shrink: Distance preserving graph compression. *Information Systems*, 2017.

[378] I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *Journal of Experimental Algorithmics (JEA)*, 13:4, 2009.

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

[380] C. F. Sainte-Marie. Solution to question nr. 48. *L'intermédiaire des Mathématiciens*, 1:107–110, 1894.

[381] K. Salikhov, G. Sacomoto, and G. Kucherov. Using cascading bloom filters to improve the memory usage for de brujin graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 364–376. Springer, 2013.

[382] E. R. Scheinerman. Local representations using very short labels. *Discrete mathematics*, 203(1):287–290, 1999.

[383] B. Schiller, J. Castrillon, and T. Strufe. Efficient data structures for dynamic graph analysis. In *2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 497–504. IEEE, 2015.

[384] C. Schröppel and J. Wackerfuß. Meshing highly regular structures: The case of super carbon nanotubes of arbitrary order. *Journal of Nanomaterials*, 16(1):441, 2015.

[385] C. N. Sekharan, S. Radhakrishnan, B. Nelson, and A. Chatterjee. Queryable compression for massively streaming social networks. 2017.

[386] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064. ACM, 2015.

[387] C. E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois press, 1998.

[388] S. Sheikhizadeh, M. E. Schranz, M. Akdel, D. de Ridder, and S. Smit. Pantools: representation, storage and exploration of pan-genomic data. *Bioinformatics*, 32(17):i487–i493, 2016.

[389] L. Shi, Q. Liao, X. Sun, Y. Chen, and C. Lin. Scalable network traffic visualization using compressed graphs. In *Big Data, 2013 IEEE International Conference on*, pages 606–612. IEEE, 2013.

[390] Q. Shi, Y. Xiao, N. Bessis, Y. Lu, Y. Chen, and R. Hill. Optimizing k2 trees: A case for validating the maturity of network of practices. *Computers & Mathematics with Applications*, 63(2):427–436, 2012.

[391] J. Shun and G. E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *ACM Sigplan Notices*, volume 48, pages 135–146. ACM, 2013.

[392] J. Shun, L. Dhulipala, and G. E. Blelloch. Smaller and faster: Parallel processing of compressed graphs with ligra+. In *Data Compression Conference (DCC), 2015*, pages 403–412. IEEE, 2015.

[393] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, pages 6–12. ACM, 1999.

[394] G. Simonyi. Graph entropy: A survey. *Combinatorial Optimization*, 20:399–441, 1995.

[395] G. Simonyi. *Entropies, capacities and colorings of graphs.* PhD thesis, MTA Rényi Alfréd Matematikai Kutatóintézet, 2006.

[396] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

[397] J.-i. Sohn and J.-W. Nam. The present and future of de novo whole-genome assembly. *Briefings in bioinformatics*, page bbw096, 2016.

[398] J. P. Spinrad. *Efficient graph representations.* American mathematical society, 2003.

[399] N. Stanley, R. Kwitt, M. Niethammer, and P. J. Mucha. Compressing networks with super nodes. *arXiv preprint arXiv:1706.04110*, 2017.

[400] T. Suel and J. Yuan. Compressing the graph structure of the web. In *Data Compression Conference, 2001. Proceedings. DCC 2001.*, pages 213–222. IEEE, 2001.

[401] J. Sun, E. M. Bollt, and D. Ben-Avraham. Graph compression—save information by exploiting redundancy. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(06):P06001, 2008.

[402] J. Swacha and S. Grabowski. Ofr: An efficient representation of rdf datasets. In *International Symposium on Languages, Applications and Technologies*, pages 224–235. Springer, 2015.

[403] A. Szymczak, J. Rossignac, and D. King. Piecewise regular meshes: Construction and compression. *Graphical Models*, 64(3-4):183–198, 2002.

[404] M. Talamo and P. Vocca. Compact implicit representation of graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 164–176. Springer, 1998.

[405] M. Talamo and P. Vocca. Representing graphs implicitly using almost optimal space. *Discrete Applied Mathematics*, 108(1):193–210, 2001.

[406] R. Tamassia. A dynamic data structure for planar graph embedding. *Automata, Languages and Programming*, pages 576–590, 1988.

[407] N. Tang, Q. Chen, and P. Mitra. On summarizing graph streams. *arXiv preprint arXiv:1510.02219*, 2015.

[408] N. Tang, Q. Chen, and P. Mitra. Graph stream summarization: From big bang to big crunch. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1481–1496. ACM, 2016.

[409] C. Thomassen. The graph genus problem is np-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[410] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[411] H. Toivonen, S. Mahler, and F. Zhou. A framework for path-oriented network simplification. In *International Symposium on Intelligent Data Analysis*, pages 220–231. Springer, 2010.

[412] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973. ACM, 2011.

[413] E. Trucco. A note on the information content of graphs. *Bulletin of Mathematical Biology*, 18(2):129–135, 1956.

[414] I. Tsalouchidou, G. D. F. Morales, F. Bonchi, and R. Baeza-Yates. Scalable dynamic graph summarization. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 1032–1039. IEEE, 2016.

[415] G. Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

[416] J. Urbani, J. Maassen, and H. Bal. Massive semantic web data compression with mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 795–802. ACM, 2010.

[417] J. Urbani, J. Maassen, N. Drost, F. Seinstra, and H. Bal. Scalable rdf data compression with mapreduce. *Concurrency and Computation: Practice and Experience*, 25(1):24–39, 2013.

[418] B. P. Vandervalk, S. D. Jackman, A. Raymond, H. Mohamadi, C. Yang, D. A. Attali, J. Chu, R. L. Warren, and I. Birol. Konnector: Connecting paired-end reads using a bloom filter de bruijn graph. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pages 51–58. IEEE, 2014.

[419] R. Viaña. Quick encoding of plane graphs in log214 bits per edge. *Information Processing Letters*, 108(3):150–154, 2008.

[420] J. Wang, Y. Huang, F.-X. Wu, and Y. Pan. Symmetry compression method for discovering network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(6):1776–1789, 2012.

[421] J. Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 217–218. ACM, 2012.

[422] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

[423] R. Wickremesinghe, R. Stata, and J. Wiener. Link compression in the connectivity server. Technical report, Technical report, Compaq systems research center, 2000.

[424] K. Wilkinson and K. Wilkinson. Jena property table implementation, 2006.

[425] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[426] H. Wu, B. Villazon-Terrazas, J. Z. Pan, and J. M. Gomez-Perez. How redundant is it?-an empirical analysis on linked datasets. In *Proceedings of the 5th International Conference on Consuming Linked Data-Volume 1264*, pages 97–108. CEUR-WS. org, 2014.

[427] Y. Wu, Z. Zhong, W. Xiong, and N. Jing. Graph summarization for attributed graphs. In *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, volume 1, pages 503–507. IEEE, 2014.

[428] K. Yamanaka and S.-I. Nakano. A compact encoding of plane triangulations with efficient query supports. In *Proceedings of the 2Nd International Conference on Algorithms and Computation*, WALCOM'08, pages 120–131, Berlin, Heidelberg, 2008. Springer-Verlag.

[429] J. Yang, S. A. Savari, and O. Mencer. An approach to graph and netlist compression. In *Data Compression Conference, 2008. DCC 2008*, pages 33–42. IEEE, 2008.

[430] C. Ye, Z. S. Ma, C. H. Cannon, M. Pop, and W. Y. Douglas. Exploiting sparseness in de novo genome assembly. *BMC bioinformatics*, 13(6):S1, 2012.

[431] J. You, Q. Pan, W. Shi, Z. Zhang, and J. Hu. Towards graph summary and aggregation: A survey. In *Social Media Retrieval and Mining*, pages 3–12. Springer, 2013.

[432] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu. Triplebit: a fast and compact system for large scale rdf data. *Proceedings of the VLDB Endowment*, 6(7):517–528, 2013.

[433] A. Zakirov and J. Brown. Nsga-ii for biological graph compression. *Advanced Studies in Biology*, 9(1):1–7, 2017.

[434] D. R. Zerbino. *Genome assembly and comparison using de Bruijn graphs*. PhD thesis, University of Cambridge, 2009.

[435] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[436] H. Zhang, Y. Duan, X. Yuan, and Y. Zhang. Assg: adaptive structural summary for rdf graph data. In *Proceedings of the 2014 International Conference on Posters & Demonstrations Track-Volume 1272*, pages 233–236. CEUR-WS. org, 2014.

[437] L. Zhang, M. Gao, W. Qian, and A. Zhou. Compressing streaming graph data based on triangulation. In *Asia-Pacific Web Conference*, pages 164–175. Springer, 2016.

[438] L. Zhang, C. Xu, W. Qian, and A. Zhou. Common neighbor query-friendly triangulation-based large-scale graph compression. In *International Conference on Web Information Systems Engineering*, pages 234–243. Springer, 2014.

[439] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 880–891. IEEE, 2010.

[440] F. Zhou. Graph compression. *Department of Computer Science and Helsinki Institute for Information Technology HIIT*, pages 1–12, 2015.

[441] F. Zhou et al. Methods for network abstraction. 2012.

[442] F. Zhou, Q. Qu, and H. Toivonen. Summarisation of weighted networks. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–30, 2017.

[443] M. Zneika, C. Lucchese, D. Vodislav, and D. Kotzinos. Rdf graph summarization based on approximate patterns. In *International Workshop on Information Search, Integration, and Personalization*, pages 69–87. Springer, 2015.