Modelling of Efficient Graph-aware Data Storage using DNA

Asad Usmani[a] and Lena Wiese[b]
Institute of Informatics, Goethe University, Bockenheim Campus, Frankfurt, Germany

Keywords: Dna Storage, Big Data Archives, Data Management, Data Modelling, Data Compression, Simple Graph.

Abstract: The global demand for massive data archival with a yearly exponential growth rate is near to outpacing the capability of the conventional world storage media. Fortunately, DNA (Deoxyribonucleic acid) storage has made a substantial breakthrough for archiving such vast data for a long time. Though many scientists have made remarkable efforts to use DNA storage as a promising emergent solution for archiving raw data, not anyone has exploited it to store graph-aware encoded data. Desirably, the exploitation of graph-aware data archiving has notable advantages over raw data. That supports data portability and significantly reduces the concerned data size for DNA storage in terms of nucleotides. Hence, it benefits us in database operational cost reduction. We present a theoretical model for efficient DNA storage of simple graph-based scientific data. Furthermore, some simple graph-based datasets, particularly from the biological domain, have been used for experimental results and analysis. That revealed a compression ratio between 1.18 to 1.53.

1 INTRODUCTION

Long term scientific data archives (Doorn and Tjalsma, 2007; Buneman et al., 2004; Whitlock et al., 2010) have become a foundation of science and future research advancements. Researchers in various fields such as biology, life sciences, ecology and medicine, working exclusively on network data, need historical data archives for practical experiments. For instance, six prominent public databases have adapted to store and retrieve existing data about the protein-protein interaction (PPI) network (Lehne and Schlitt, 2009). Typically, a graph model is used to represent such network data. For convenient modelling of the PPI network, we denote proteins as nodes and interactions between them as edges within a simple undirected graph. Likewise, various other applications from social networks, biological networks, community detection etc., are also strong candidates for simple graph-based modelling. Needfully, the gigantic graph-aware data archival wants a low cost and durable storage medium. Fortunately, DNA storage has been established and functional for offering such data archival.

Since the 60s, scientists have dreamed about the data storage capabilities of DNA (Neiman, 1964), but this field has significantly evolved during the last decade. In 2012 and 2013, almost a megabyte of data was stored in DNA and then successfully recovered by two groups guided by Church (Church et al., 2012) and Goldman (Goldman et al., 2013), respectively. Prominently, (Organick et al., 2018) stored and recovered data at a large scale comprised of 35 files (over 200MB in size) using millions of DNA nucleotides. Thus, synthetic DNA storage is potentially a next-generation data storage medium (Clelland et al., 1999; Bancroft et al., 2001; Church et al., 2012; Goldman et al., 2013; Bornholt et al., 2016), which is a low cost, highly dense, immutable, durable and energy-efficient storage solution for rapidly growing archived data. The half-life of DNA is approximately 520 years (Allentoft et al., 2012), which reveals its high durability. Despite that DNA storage ensures the longevity of the data archival at a low cost, the concerned accessibility cost makes its usage slightly impractical. However, the accessibility cost is exponentially decreasing every year with the rapid advancements in the biotechnology industry (Carlson, 2014; Eid et al., 2009). Since DNA writing (synthesis) and reading (sequencing) methods are gradually upgrading, thus economical DNA storage is evolving with anticipation. Evidently, DNA storage has made great strides in preserving huge data seemingly forever hence simple graph-aware data archival is a must use-case.

The simple graph-aware data storage using DNA would be comparatively more efficient than raw data archival. We will see in a later section that it re-

[a] https://orcid.org/0000-0001-5579-6980
[b] https://orcid.org/0000-0003-3515-9209

Usmani, A. and Wiese, L.
Modelling of Efficient Graph-aware Data Storage using DNA.
DOI: 10.5220/0011355400003269
In *Proceedings of the 11th International Conference on Data Science, Technology and Applications (DATA 2022)*, pages 180-189
ISBN: 978-989-758-583-8; ISSN: 2184-285X

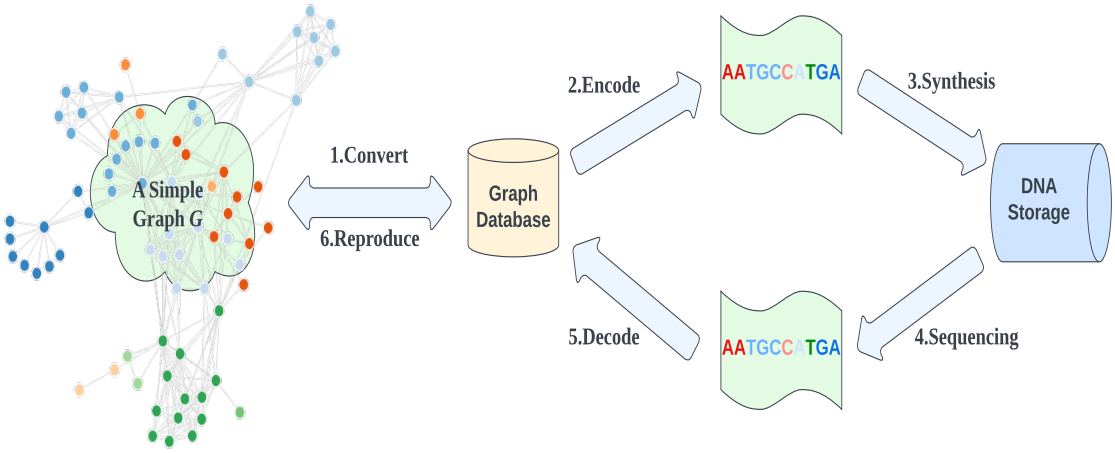


Figure 1: DNA end to end process for reading and writing a simple graph data.

duces the number of nucleotides that need to be stored in DNA. This compact data leads toward more economical DNA storage as the synthesis process costs up to almost $0.05–0.15 per nucleotide (Kosuri and Church, 2014). Additionally, it also resolves the data portability concern as a by-product. Due to this, graph-aware encoded data archival is convincingly encouraged rather than raw data archival. We start this article by explaining the DNA background and defining the simple graph and compression techniques in Section 2. Then we proceed to Section 3 to discuss our methodology, where we propose a theoretical model for simple graph-aware data storage using DNA. In section 4, the experimental results, provisioning diverse real-world datasets, illustrate the worthwhile compression ratio obtained.

2 BACKGROUND AND RELATED WORK

2.1 DNA Storage System and Database Engine

Deoxyribonucleic acid or DNA (Seeman, 2003) is a composition of four nucleotides: adenine (A), thymine (T), cytosine (C), and guanine (G) and a sequence of these nucleotides is called an oligonucleotide (oligo) or a DNA strand. Generally, the DNA molecules are the carrier of genetic information, which is functionally essential for all living organisms. Moreover, a DNA strand has two ends: $5'$ and $3'$ and its structure is composed of a double helix in nature with two ("reverse complement") strands in opposite directions. In contrast, a single DNA strand is sufficient for data storage. Primarily, DNA storage deals with nucleotides rather than bits for traditional storage media such as magnetic tape, HDD, SSD etc. Therefore, digital data must be in a sequence of quaternary (A, G, T, C) characters for DNA storage compatibility. For instance, a binary string of length n can be encoded into an equivalent oligo of size $n/2$ by mapping 00, 01, 10, 11 to A, G, T and C, respectively. Optionally, one of the suggested DNA encoding schemes (Heinis and Alnasir, 2019) can also be used as appropriate. Overall, any digital data of classic objects like PDF, JPEG or a graphical representation etc., can be encoded and then decoded to retrieve the original information using DNA. An end-to-end DNA storage process is abstractly expressed in Figure 1 to read and write a simple graph data.

(Appuswamy et al., 2019) proposed a DNA storage system architecture for a relational database as *OligoArchive*. They replaced traditional data storage tape with a DNA storage device as oligos had to be stored instead of binary data. The presented DNA storage system has three components: a synthesizer, a sequencer, and a storage container. The DNA synthesizer encodes the digital data and then stores them in DNA as oligos. Conversely, the DNA sequencer reads oligos and converts them into original digital data. Both synthesis and sequencing methods in the DNA storage system support PUT and GET operations of the database engine, respectively. The storage container can be conceived as an object for the DNA storage device. This object is also called DNA Storage Library: a collection of "DNA pools". Knowingly, an oligo is the basic unit of DNA storage, which can roughly store at most 100-200 nucleotides. Therefore, many DNA strands are needed to map a classic data object in partitions. The limitation of one-on-one mapping of oligos and DNA pools leads to reserving several DNA pools to contain single object

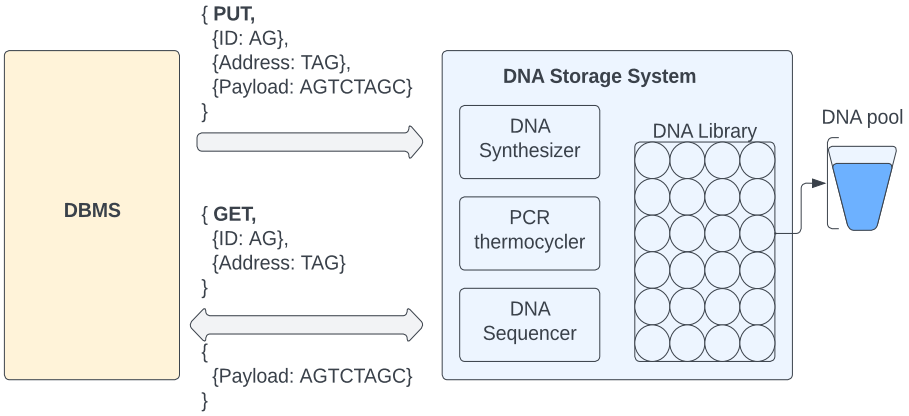


Figure 2: Demonstrates an abstraction of the database engine and DNA storage system interfacing proposed in the *OligoArchive* architecture. Both GET and PUT methods are used to read and write encoded relational data, respectively.

data. Unlikely, a "DNA pool" does not allow an organized address for indexing as a traditional storage medium. So, a unique address itself must be embedded in every DNA strand with payload data. A DNA sequencing primer as a unique object identifier is responsible for retrieving all the corresponding oligos from the DNA Storage Library for a particular object. The PCR (polymerase chain reaction) thermocycler facilitates retrieving data from the DNA Storage Library when a template DNA strand provides input in the sequencing process. This template DNA strand includes the sequencing primers to locate the exact oligo from where the data are to be retrieved and populates itself with output data. In conclusion, both GET and PUT methods can be utilised in reading and writing oligo(s) from/to DNA storage system using the same database engine for the relational data as shown in Figure 2.

2.2 Simple Graph Model and Representation

To gain efficiency for simple graph-aware DNA data storage in terms of nucleotides, the impact of different representations for a simple graph model needs to evaluate quantitatively. (Besta et al., 2019) comprehensively described a few graph models from the perspective of graph databases. In this regard, we are mentioning only a simple graph underneath.

Definition: *A (simple) graph $\mathcal{G}$ is modeled using a set of two objects $\mathcal{V}$ and $\mathcal{E}$, where $\mathcal{V}$ is a set of vertices (or nodes) and $\mathcal{E}$ denotes a set of edges (or links). The set $\mathcal{V}$ consists of a non-empty finite number of n vertices, e.g. $\mathcal{V} = \{v_1, v_2, v_3, \ldots, v_n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of m paired vertices, e.g. $\mathcal{E} = \{e_1, e_2, e_3, \ldots, e_m\} \mid \forall\, e_k \in \mathcal{E}\ (1 \geq k \leq m) = \{(x, y) — x, y \in \mathcal{V}$ and $x \neq y\}$. For an undirected $\mathcal{G}$, an edge $e_k = (x,y) \in \mathcal{E}$ is a set of two nodes while e_k represents a tuple (or directed edge) of two nodes from x towards y in case of directed graph $\mathcal{G}$, where x stands for the out-vertex (or source node) and y stands for the in-vertex (or dest node).*

The selection of a simple graph representation technique must be optimal for efficient DNA storage. Primarily, one of the four techniques (Davoudian et al., 2018) can be used for a simple graph representation: adjacency matrix (AM), adjacency list (AL), edge list (EL) and compressed sparse row (CSR). We briefly describe only AM and AL as followings:

Adjacency Matrix: Usually, AM is used to represent $\mathcal{G}$ in memory. In this representation, a matrix $\mathcal{M} \in \{0,1\}^{n,n}$ shows the presence of all the edges in $\mathcal{G}$ such that $\mathcal{M}_{x,y} = 1 \Leftrightarrow (x,y) \in \mathcal{E}$ if (x,y) is directed, otherwise $\mathcal{M}_{x,y} = 1$ and $\mathcal{M}_{y,x} = 1$. The AM representation is not recommended for large scale graphs.

Adjacency List: For the AL representation of $\mathcal{G}$, we maintain an array A for n vertices such that $|A| = n$. Each index of the array A corresponds to a vertex's ID. In addition, for each vertex x, a list A_x is associated with it, which only contains IDs of other vertices such that $y \in A_x \Leftrightarrow (x,y) \in \mathcal{E}$ if (x,y) is directed, otherwise $y \in A_x$ and $x \in A_y$. The AL format minimizes the storage overhead for sparse graphs.

2.3 Compression Techniques

Researchers have developed various compression techniques (Boldi and Vigna, 2004; Apostolico and Drovandi, 2009) to optimize a graph representation by exploiting compression for AM, AL and EL etc. Despite that, (Simecek, 2009) used the quadtree compression strategy for simple graph storage; a more compact representation of AM presented by (Álvarez et al., 2010) is called k^2-tree.

2.3.1 k^2-tree Compression Technique

For k^2-tree construction (Álvarez et al., 2010), an AM $\mathcal{M}$ of size $n \times n$ is recursively partitioned based on an input value k such that n is in k's power. For each partition, a node symbolizing a parent contains one bit and k^2 pointers to its children (sub-partitions) in the tree. At the first level, the $\mathcal{M}$ is split equally into k^2 submatrices. The root node holds these k^2 submatrices using its k^2 pointers. For each node of k^2 children, its bit is enabled to 0 if the corresponding partition contains only 0s, or 1 otherwise. A node with a bit value 0 reveals that its affiliated submatrix is empty. Hence, there is no need to make a recursive call for that particular partition anymore. Otherwise, the submatrix further breaks continually until the base case finds. Concisely, the k^2-tree technique generates a binary string as an output. That optimally represents a simple graph compared to AM. The k^2-tree takes advantage of a matrix's sparsity, especially when a sparse matrix would have multiple large regions containing only 0s. Where applicable, this technique is valuable in getting significant data compression.

2.3.2 Re-Pair Algorithm

Furthermore, (Claude and Navarro, 2010) adapted a grammar-based compression technique for the AL compression of a simple graph using Re-Pair algorithm (RA). Using this technique, a single array list $L(G)$ is composed by including all the adjacency lists of a graph G: $L(G) = \bar{v_1}v_{1,1}v_{1,2}...v_{1,m_1}\ \bar{v_2}v_{2,1}v_{2,2}...v_{2,m_2}\ ...\ \bar{v_n}v_{n,1}v_{n,2}...v_{n,m_n}$ so that $v_{j,k} < v_{j,k+1}$ where $1 \leq j \leq n$ and $1 \leq k \leq m_j$. There, each delimiter $\bar{v_j} = -v_j$ is essentially incorporated in the list $L(G)$ to distinguish the adjacency list of the node v_j from the previous one.

Without going into details for the algorithmic explanation, the Re-Pair algorithm runs various steps on input list $L(G)$ and then produces three objects as output: a remaining adjacency list $C(G)$, a dictionary T for grammar rules and two bitmaps B_1 and B_2. The list $C(G)$ contains a subset of $L(G)$. The dictionary T occupies several dictionary rules. For tracing the beginning and size of each adjacency list in the $C(G)$, two bitmaps $B_1[1...n]$ and $B_2[1...|C(G)|]$ can be advantageously used rather than keeping pointers to every adjacency list especially for sparse graphs. If the adjacency list of node V_i is empty then $B_1[i] = 0$ otherwise $B_1[i] = 1$. The bitmap B_2 marks the starting place of every adjacency list in C(G). Where the function $rank(B_1, i)$ returns the number of $1s$ until index i inclusively in B_1. Given that if $B_1[i] = 1$ then function $select(B_2, rank(B_1, i))$ returns the starting index of i^{th} node's adjacency list otherwise it is empty.

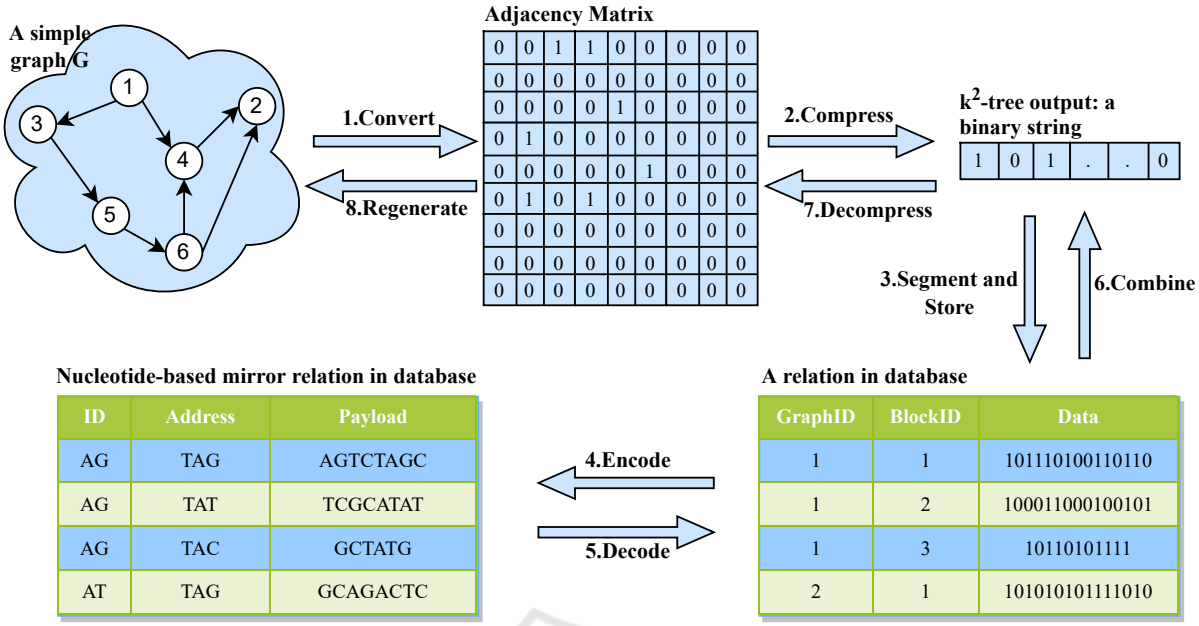
3 PROPOSED GRAPH-AWARE DATA STORAGE MODELLING FOR DNA

A model is an abstract but informative representation of a system. That reveals the entities or objects and relationships among those present within that system. Furthermore, we can gain a comprehensive understanding of the overall workflow of the entire process while using a model (Epstein, 2008).

3.1 Model using k^2-tree Technique

Here we suggest an efficient theoretical model for storing a simple directed graph into DNA. For that, four recommended steps are illustrated in Figure 3. Firstly, a simple directed graph is represented in AM format and subsequently compressed in a binary string using the k^2-tree technique. The size of that compressed binary string depends on the input graph size and its compression ratio. Due to the limited capacity, accommodation of an entire binary sequence into one DNA strand is not permissible. So, in the third step, the single binary object is divided into multiple fixed-length segments. The size of a DNA strand determines the number of blocks of the input binary string. For instance, a binary string with a length of 900 bits will be partitioned into three chunks only if a DNA strand consists of 150 nucleotides in addition to primers and considering the encoding method presented by Church (Lee et al., 2018), which maps 2 bits to one nucleotide. Accordingly, a relation with three attributes: *GraphID*, *BlockID* and *Data*, is created within a database system to store every graph's equivalent binary configuration. The attribute *GraphID* denotes a graph number in a relation and identifies all the chunks associated with this graph. The attributes *BlockID* and *Data* specify the order of each block and its corresponding binary data of a graph, respectively. Any relational database may contain such fixed-length data of all given attributes within a relation. Finally, that relation replicates into another nucleotides-based shadow relation such that each tuple containing binary data maps to an equivalent nucleotides-based tuple by encoding. Hence, each nucleotides-based tuple will fit into an independent DNA strand for systematic DNA storage. Conclusively, more compacted graph data will demand fewer DNA strands. It, in turn, benefits in overall cost reduction of synthesis and sequencing processes.

In general, how much compression ratio can be obtained using the k^2-tree format? To answer it, let us assume that we exactly find as many 0s as 1s at each level of the tree. Thus, if we have n children on



ID	Address	Payload
AG	TAG	AGTCTAGC
AG	TAT	TCGCATAT
AG	TAC	GCTATG
AT	TAG	GCAGACTC

GraphID	BlockID	Data
1	1	101110100110110
1	2	100011000100101
1	3	10110101111
2	1	101010101111010

Figure 3: Illustrates the step by step database storage process of a simple graph by using the k^2-tree compression technique.

i^{th} level then there will be $\frac{n}{2} \times k^2$ children on $i+1^{th}$ level. A formula $T(i) = \frac{(k^2)^{(i-1)}}{2^{i-2}} | i \geq 3$ to find the number of child nodes at any i^{th} level can easily be determined using this recurrence $T(i) = \frac{T(i-1)}{2} \times k^2$, where $T(2) = k^2$ and $T(1) = 1$. If $d = \log_k n$ denotes the last level of k^2-tree then total bits required to represent an AM into k^2-tree format will be equal to $S = (\sum_{i=3}^{d} \frac{(k^2)^{(i-1)}}{2^{i-2}}) + k^2 + 1$. Finally, the compression ratio calculates using this $\frac{n^2}{S}$ formula as AM needs n^2 bits to represent a graph. For $n = 1024$ and $k = 4$, it produces a compression ratio almost 64 times better than AM. Theoretically, we acknowledge a considerable reduction in the nucleotide counts needed for DNA storage. However, in the next section, we will evaluate the experimental results for better comparison. In conclusion, we have not obtained any anticipated graph-aware data exploitation for efficient DNA storage using this technique rather than compression only.

3.2 Model using Re-Pair Algorithm

The theoretical model for DNA storage of graph-based data using the RA is presented below in Figure 4. Firstly, a graph G representation transforms into an AL instead of to AM in Figure 3. A composite edge list: a concatenation of all edges-list into a single edge list; is then provided with an input to the RA during the second step. For data compression, we already know that the Re-Pair algorithm generates a triplet of objects as output: 1) a dictionary of rules T, 2) a remaining edges list $C(G)$, and 3) two bitmaps B_1 and B_2. This triplet is a core to reconstruct the original graph again in AL representation after its perfect restoration from DNA storage. Ideally, this triplet requires fewer bits than direct storage of a single identical edges list into DNA. A mechanism must be established for the careful handling of such triplets in DNA storage. Specifically, the parts of a triplet must be concatenated to obtain a binary string and then restored again correctly when needed. Thirdly, each triplet's elements are successively integrated in addition to their cardinalities as prefixes to make a composite string. Indeed, each triplet's factor will be split again from its composite object with the help of its prefix. Afterwards, the binary sequence is partitioned into multiple fixed-length chunks and stored in a database table. Furthermore, a nucleotides-based shadow relation is synchronized from binary to nucleotides format after encoding data similar to the previous process.

We consider and evaluate the AL representation of a simple directed graph G for DNA storage. Supposedly, we store it as a concatenated sequence of $n+m+1$ integers with p bits each such that $|V||\|_{i=1}^{n} V_i = |V_i||e_{i,1}\|e_{i,2}\|\ldots\|e_{i,|V_i|}$, where $n = |V|$, $m = \sum_{i=1}^{n} |V_i|$, $p = \log_2 |V|$ and $x\|y = xy$. The compound sequence of all adjacency lists of G will result in a binary string of a total $D = (n+m+1) \times p$ bits in length. Importantly, how would the integers be reproduced from this composite binary sequence? For this, a fixed-size integer of 2 bytes has to append with the binary string as a

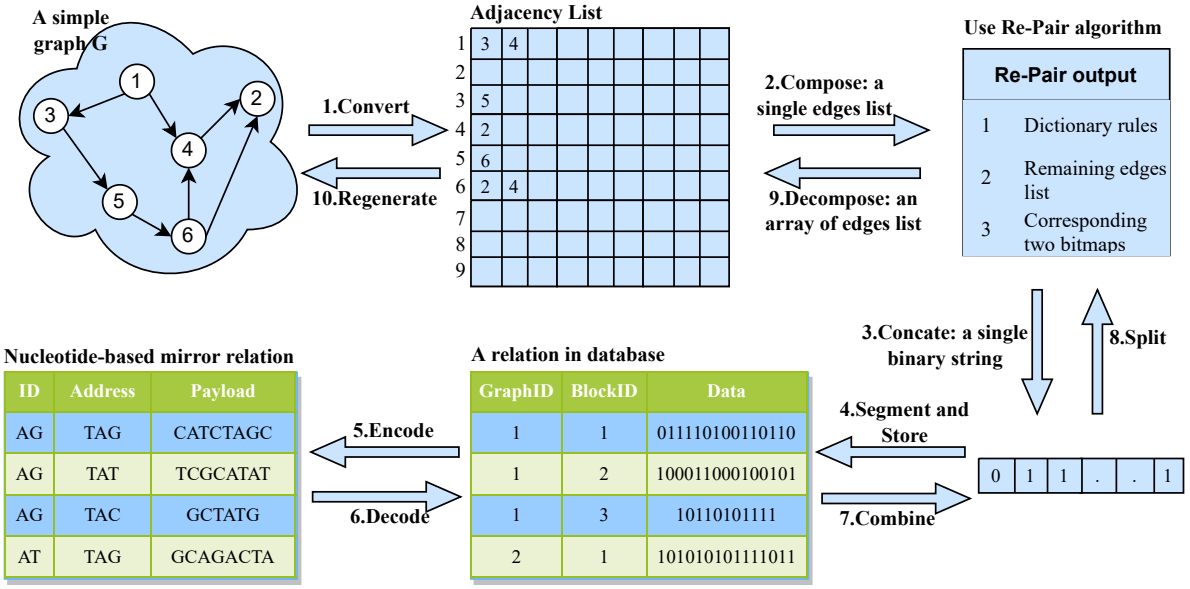


Figure 4: Illustrates the step by step database system storage process of a simple graph by using the Re-Pair algorithm.

prefix. This prefix will contain the value of p and aid in splitting all the integers and regenerating the original AL correctly. Hence, a total of $D+16$ bits are enough for AL storage as raw data in DNA.

Arguably, we find a significant compression ratio for a simple directed graph G using Re-Pair algorithm. As indicated above, let us say $l = |C(G)|$, $d = max(C(G))$, $n = |L(G)|$, $p = \log_2 |V|$ and $r = |T|$ then l, d, n, p and r accumulatively determine the total bits sufficient to compress a graph configration list $L(G)$ using this technique, where $|B_1| = l$ and $|B_2| = n$. Assuming each dictionary rule in T is a triplet of integers, which can be stored in an array R. Hence, $|R| = 3 \times |T|$. Apparently, a total $S_1 = (3r + l) \times p + n + l$ bits are required to store a compressed form of $L(G)$ in this case. Moreover, we know that each dictionary rule comprises one non-terminal and two terminals. For optimization, a pattern exists in the numeric values of non-terminals of dictionary rules in T, which starts from $|V|+1$ and then increments by one for every succeeding rule. Since we store a tuple of terminals with a non-terminal value $|V|+i$ at i^{th} index of the array R for $i \geq 1$, an index value of every tuple of terminals in R in addition to $|V|$ equals the value of its non-terminal. Therefore, this numeric pattern advantageously omits the storage demand for the non-terminal integers in the array R. Desirably, apart from compression, we have found substantial exploitation in graph-aware data for DNA storage using this strategy. Accordingly, a composite binary string including $C(G)$, R, B_1 and B_2 objects with their cardinalities as prefixes can be structured as follows: $p\|l\|C(G)\|2r\|R\|l\|B_1\|n\|B_2$. Remember, a fixed-size 2 bytes integer containing the value p is used to distinguish all the integers from each other. Optimally, now a total of $S_2 = (2r + l + 4) \times p + n + l + 16$ bits (equals to $\frac{S_2}{2}$ nucleotides) is necessary to store a compressed graph representation using DNA in this case. Therefore, the compression ratio can be calculated as $\frac{D+16}{S_2}$ for the Re-Pair algorithm.

Of course, the experimental analysis will reveal what proportion of compression would be achieved. However, assuming that the fraction value of $\frac{D+16}{S_2}$ would be significantly greater than 1 to deliver a valuable compression ratio. Then, using this, fewer nucleotides would be needed for simple graph storage in DNA to AL requirements considering the same encoding scheme. Since, $a = \lceil \frac{D+16}{m} \rceil \geq b = \lceil \frac{S_2}{m} \rceil$ when $\frac{D+16}{S_2} \geq 1$, hence, fewer DNA strands with each size m would be synthesised and sequenced using PUT and GET methods offered in the database engine (Appuswamy et al., 2019). Eventually, an efficient DNA storage mechanism demonstrated above will profit us by saving a cost of at least $(a-b) \times m \times \$0.05$. It also assists in data portability because the nucleotide-based relation keeps track of the primer addresses required to fetch the corresponding data from DNA in future. So, an existing graph data can be retrieved randomly from DNA using the GET method to refill the concerned nucleotide relation in the database. We can reconstruct a simple graph once all its relevant tuples would recover appropriately. Formerly, the reengineering process should be followed by orderly attending to steps from 6 to 10 in Figure 4.

4 EXPERIMENTAL EVALUATION

This section shows experimentally that the Re-Pair compressor produces significant compression ratios over other graph representations like AL and AM. The results shown in the Figure 5 are based on simple graphs with randomly generated edge-lists. Where both of the Figures 5(a) and 5(b) illustrate the compression ratio (y-axis) with respect to various number of nodes (x-axis) for different graphs with an average EpV (edges per vertex) ratios of 10% and 20% respectively. As we know, the AM representation of a graph needs $|V|^2+16$ bits hence the compression ratio using the RA over AM can be calculated as $\frac{|V|^2+16}{S_2}$, where a total $\frac{S_2}{2}$ nucleotides are required to represent a graph using the Re-Pair algorithm in DNA. The blue and red lines in the Figure 5 act for RA compression ratio achieved over AL and AM representations respectively. Intuitively, the Re-Pair algorithm should be compared only with AL. Instead, it reveals in the Figure 5(b) that AL behaves worse than AM representation if the average EpV ratio of any simple graph inclined over 20%. This argument can be proved by using a mathematical relation that $0.20 \times |V|^2 \times \log_2 |V| \geq |V|^2$ for all $V \geq 32$, where $\log_2 |V|$ is the total bits to store a single integer value. Therefore, the potential compression ratio gain of the RA algorithm over AL representation does not guarantee us a real benefit in terms of data compression because it may still be lower than either AM representation or not significantly greater enough. Hence, the k^2-tree compression should be an obvious choice if a graph has an average EpV ratio above 20%. Otherwise, the Re-Pair algorithm compression technique performs better, as depicted in Figure 5(a). However, our experimental study reveals that the average EpV ratio for most of the graphs does not accede over 20% in practice which recommends utilisation of the Re-Pair algorithm generically for all simple graphs. Two exemplary datasets have been compressed using the RA. Moreover, the results are shown graphically in Figures 6 and 7. To pick the best compression technique for a graph to take maximum advantage of DNA storage, we will evaluate all potential compression techniques for that candidate graph data. Then, that graph data will compress accordingly. Certain bits as compressor ID should append to the final compressed binary string as a prefix because graph data will reconstruct according to its specific compression technique. For instance, a two bits prefix is enough to distinguish four different compression techniques. Generically, the additional eight bits as a prefix to the final compressed binary string should be reserved only for this purpose. Conclusively, for graphs in the metabolic dataset, a minimum compression ratio between 1.21 to 1.53 has been achieved using the RA. The data size compresses to at least 18% for all simple graphs in the case of the PPI (protein-protein interactions) network dataset. In summarizing, the compression ratio obtained would eventually benefit in worthwhile cost reduction of DNA storage. The implementation code for the Re-Pair compressor is available on our GitHub repository[1] in Python language.

4.1 Datasets

4.1.1 Metabolic and Protein Networks

This dataset[2] is about the network of metabolic reactions of (Escherichia coli) bacteria. In the metabolic network, each node represents a metabolite and each directed edge $X \rightarrow Y$ denotes the reaction between metabolites X and Y, where X is given as input and Y is a product of the reaction (e.g. $X + A \rightarrow Y + B$). The sizes of simple graphs data in this dataset are between 26-45 KB.

4.1.2 Students Network

This dataset[3] is about the students cooperation social network. In the students network, each node represents a student and each directed edge $X \rightarrow Y$ denotes an interaction or friendship between two students X and Y. The size of this simple directed graph data is only 6 KB.

4.1.3 Protein-Protein Interactions Network

This dataset[4] is about the normal and cancerous cell network with respect to protein-protein interaction. In this network data, each node symbolises a protein, and each directed edge $X \rightarrow Y$ represents the physical connection between proteins X and Y in the cell. The sizes of simple graphs data in this dataset are between 8-46 KB.

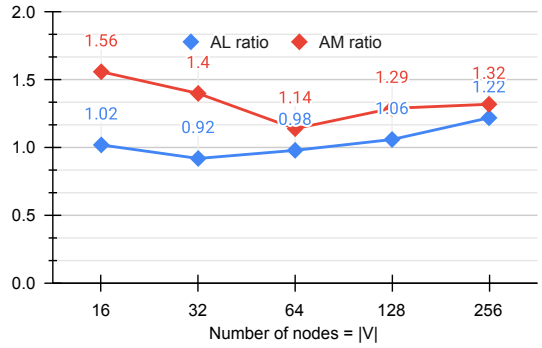
The main features of the above datasets are specified quantitatively in Table 1. Noticeably, the EpV ratio remains between 1.99 to 7.11. We have not illustrated the experimental results for the graph datasets about the cancerous category rather than the normal category in the PPI network.

[1] https://github.com/asadru90/DNA-simpleGraphRACompressor.git

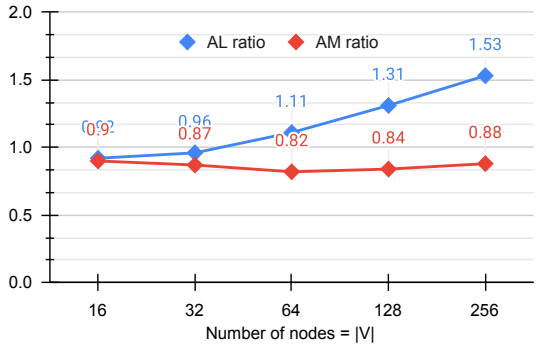
[2] http://networksciencebook.com/translations/en/resources/data.html

[3] https://data4goodlab.github.io/dataset.html

[4] https://rahmanidashti.github.io/PPINetworkAnalysis/



(a) Compression of G with average $EpV = 10\% \times |V|$.



(b) Compression of G with average $EpV = 20\% \times |V|$.

Figure 5: Compression ratio achieved using Re-Pair algorithm for different G with various $|V|$ as compared to AM and AL.

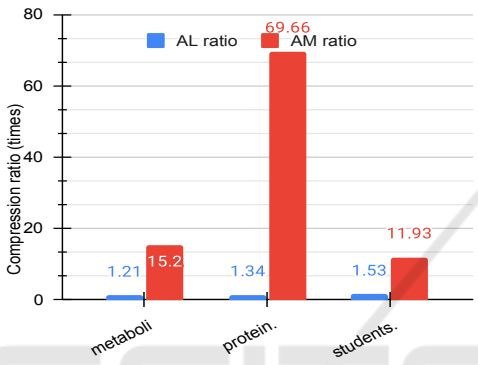


Figure 6: Compression ratio achieved using Re-Pair algorithm as compared to AL and AM representation for some graphs in metabolic and students network datasets.

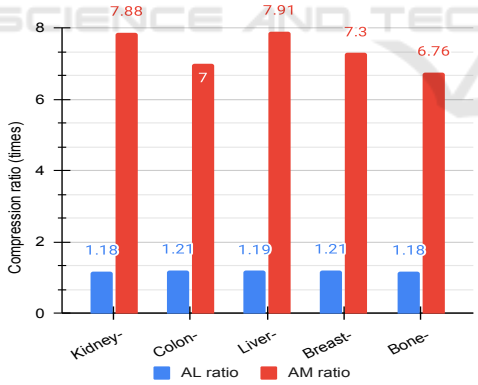


Figure 7: Compression ratio achieved using Re-Pair compared to AL and AM representation for various exemplary graphs in PPI network dataset of normal category only.

5 DISCUSSION

Given the high cost of DNA data storage but its ability to replace tape, potential data must be compressed in advance. Considering simple graph data archival demand, we have noticed that exploitation of graph-aware data storage has an advantage over raw data storage using DNA besides compression. Optionally, we can use the k^2-tree compression technique to reduce the relevant input data size for optimal DNA storage, but this does not support additional data compaction benefits using graph-aware data exploitation. However, the Re-Pair compressor claims a substantial edge of graph-aware data utilization over raw data for DNA storage. Furthermore, due to potential errors in synthesis and sequencing methods (Schwarz et al., 2020) of a DNA storage system, the reconstruction of the original graph seems to be impossible after retrieving the data from DNA storage based on the k^2-tree compression technique. Anticipatedly, even a single bit error will completely disturb the decompression step in Figure 3. Subsequently, the expected adjacency matrix would not be appropriate for reconstructing the original graph. In contrast, errors caused by biological constraints could have a little impact on overall graph reconstruction from DNA data storage based on the Re-Pair compressor unless certain critical bits, such as prefixes, are damaged. As expected, archived data should be retrieved from DNA precisely, even after a century. Therefore, a standard protocol must also be developed and globally followed by a corresponding DBMS regarding prefixes sizes. Aside from that, we observed that the Re-Pair algorithm takes too much time to produce the output, especially for large graph datasets. For instance, it finished execution in more than 10 hours to produce the final results for the (Breast-Cancer.csv) dataset that we run on a system with the specifications of i5-7300U CPU @ 2.60GHz and 32GB RAM. Therefore, promptly compression of such datasets will require a high-performance computing environment. Regarding implementation, we know that the GET and PUT operations for a relational DBMS have already been enforced practically by *OligoArchive* to read and write oligo(s) within a DNA storage system, so those can be used straightforwardly as an interface in our proposed theoretical model implementation.

Table 1: Shows some characteristics of corresponding exemplary graphs in the PPI network and other datasets from the biological domain. Further, it reveals the representation requirements of these simple graphs data using the Re-Pair algorithm compared to the AL and AM in terms of bits.

Dataset	Nodes	Edges	EpV	Plain size (KB)	AM(bits)	AL(bits)	RA(bits)
Kidney-Normal.csv	315	1347	4.27	17	99225	14958	12581
Colon-Normal.csv	305	1487	4.87	18	93025	16128	13275
Liver-Normal.csv	302	1227	4.06	15	91204	13761	11520
Breast-Normal.csv	331	1694	5.11	21	109561	18225	14997
Bone-Normal.csv	192	618	3.21	8	36864	6480	5449
Kidney-Cancer.csv	491	3135	6.38	38	241081	32634	27301
Breast-Cancer.csv	541	3847	7.11	46	292681	43880	36263
Bone-Cancer.csv	351	1781	5.07	22	123201	19188	16107
protein.edgelist.txt	2018	5047	2.5	26	4072324	77715	58454
metabolic.edgelist.txt	1039	6399	6.15	45	1079521	81818	68203
studentsNetwork.csv	184	367	1.99	6	33856	4408	2878

6 FUTURE WORK

In future work, we want practical implementation of our proposed model. While working on this research, an idea about merging multiple graphs to make a composite or aggregate graph (Liu et al., 2016) followed by its compression and then DNA storage got our attention. It could be another optimization if we succeed after the exploration that way. We also intend to exploit other complex graph models, like property graphs and RDF graphs (Das et al., 2014; Angles, 2018) for efficient DNA storage. With data archives assistance of these graphs, many emergent databases and big graph processing frameworks supply persistent, highly scalable, diversified, and efficient analytical and query processing capabilities for future application demands (Sakr et al., 2021; Cai et al., 2018). Due to this, complex graph-based data should also be exploited for archival using DNA. It could be challenging to deal with such vibrant data. However, it seems to have more potential for compaction due to the repetition of labelling data. Additionally, we want to explore other storage media like Peptide sequences (Ng et al., 2021) similar to DNA.

7 CONCLUSIONS

We presented a theoretical model for DNA storage of simple graph-based data. The overall vision for defining this model was to provide a key idea for simple graph-aware data storage in DNA rather than raw data storage and show the potential benefit of reducing the number of nucleotides that need to be stored. With that concern, a comparative analysis of two techniques: k^2-tree and the Re-Pair algorithm, was performed experimentally. Moreover, the model focused on the simple graph-aware data exploitation using the Re-Pair compressor and accomplished at least 18% compaction of input data size for DNA storage beforehand. A comprehensive description and illustration of the proposed modelling will contribute to the optimal implementation of simple graph-aware DNA data storage.

ACKNOWLEDGEMENTS

I want to thank both (DAAD-Germany and HEC-Pakistan) organizations for financially supporting me during my PhD studies at GUF under the “Overseas Scholarships for PhD in Selected Fields, Phase III, Batch-1, 2020” program.

REFERENCES

Allentoft, M. E., Collins, M., Harker, D., Haile, J., Oskam, C. L., Hale, M. L., Campos, P. F., Samaniego, J. A., Gilbert, M. T. P., Willerslev, E., et al. (2012). The half-life of dna in bone: measuring decay kinetics in 158 dated fossils. *Proceedings of the Royal Society B: Biological Sciences*, 279(1748):4724–4733.

Álvarez, S., Brisaboa, N. R., Ladra, S., and Pedreira, Ó. (2010). A compact representation of graph databases.

In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 18–25.

Angles, R. (2018). The property graph database model. In *AMW*.

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

Appuswamy, R., Le Brigand, K., Barbry, P., Antonini, M., Madderson, O., Freemont, P., McDonald, J., and Heinis, T. (2019). Oligoarchive: Using dna in the dbms storage hierarchy. In *CIDR*.

Bancroft, C., Bowler, T., Bloom, B., and Clelland, C. T. (2001). Long-term storage of information in dna. *Science*, 293(5536):1763–1765.

Besta, M., Peter, E., Gerstenberger, R., Fischer, M., Podstawski, M., Barthels, C., Alonso, G., and Hoefler, T. (2019). Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *arXiv preprint arXiv:1910.09017*.

Boldi, P. and Vigna, S. (2004). The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602.

Bornholt, J., Lopez, R., Carmean, D. M., Ceze, L., Seelig, G., and Strauss, K. (2016). A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649.

Buneman, P., Khanna, S., Tajima, K., and Tan, W.-C. (2004). Archiving scientific data. *ACM Transactions on Database Systems (TODS)*, 29(1):2–42.

Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30:1616–1637.

Carlson, R. (2014). Time for new dna synthesis and sequencing cost curves. *Synthetic Biology News*.

Church, G. M., Gao, Y., and Kosuri, S. (2012). Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628.

Claude, F. and Navarro, G. (2010). Fast and compact web graph representations. *ACM Transactions on the Web (TWEB)*, 4(4):1–31.

Clelland, C. T., Risca, V., and Bancroft, C. (1999). Hiding messages in dna microdots. *Nature*, 399(6736):533–534.

Das, S., Srinivasan, J., Perry, M., Chong, E. I., and Banerjee, J. (2014). A tale of two graphs: Property graphs as rdf in oracle. In *EDBT*, pages 762–773.

Davoudian, A., Chen, L., and Liu, M. (2018). A survey on nosql stores. *ACM Computing Surveys (CSUR)*, 51(2):1–43.

Doorn, P. and Tjalsma, H. (2007). Introduction: archiving research data. *Archival science*, 7(1):1–20.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., et al. (2009). Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138.

Epstein, J. M. (2008). Why model? *Journal of artificial societies and social simulation*, 11(4):12.

Goldman, N., Bertone, P., Chen, S., Dessimoz, C., LeProust, E. M., Sipos, B., and Birney, E. (2013). Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *nature*, 494(7435):77–80.

Heinis, T. and Alnasir, J. J. (2019). Survey of information encoding techniques for dna. *arXiv preprint arXiv:1906.11062*.

Kosuri, S. and Church, G. M. (2014). Large-scale de novo dna synthesis: technologies and applications. *Nature methods*, 11(5):499–507.

Lee, H. H., Kalhor, R., Goela, N., Bolot, J., and Church, G. M. (2018). Enzymatic dna synthesis for digital information storage. *bioRxiv*, page 348987.

Lehne, B. and Schlitt, T. (2009). Protein-protein interaction databases: keeping up with growing interactomes. *Human genomics*, 3(3):1–7.

Liu, Y., Safavi, T., Dighe, A., and Koutra, D. (2016). Graph summarization methods and applications: A survey. *arXiv: Information Retrieval*.

Neiman, M. S. (1964). Some fundamental issues of microminiaturization. *Radiotekhnika*, 1(1):3–12.

Ng, C. C. A., Tam, W. M., Yin, H., Wu, Q., So, P.-K., Wong, M. Y.-M., Lau, F., and Yao, Z.-P. (2021). Data storage using peptide sequences. *Nature Communications*, 12(1):1–10.

Organick, L., Ang, S. D., Chen, Y.-J., Lopez, R., Yekhanin, S., Makarychev, K., Racz, M. Z., Kamath, G., Gopalan, P., Nguyen, B., et al. (2018). Random access in large-scale dna data storage. *Nature biotechnology*, 36(3):242–248.

Sakr, S., Bonifati, A., Voigt, H., Iosup, A., Ammar, K., Angles, R., Aref, W., Arenas, M., Besta, M., Boncz, P. A., et al. (2021). The future is big graphs: a community view on graph processing systems. *Communications of the ACM*, 64(9):62–71.

Schwarz, M., Welzel, M., Kabdullayeva, T., Becker, A., Freisleben, B., and Heider, D. (2020). Mesa: automated assessment of synthetic dna fragments and simulation of dna synthesis, storage, sequencing and pcr errors. *Bioinformatics*, 36(11):3322–3326.

Seeman, N. C. (2003). Dna in a material world. *Nature*, 421(6921):427–431.

Simecek, I. (2009). Sparse matrix computations using the quadtree storage format. In *2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 168–173.

Whitlock, M. C., McPeek, M. A., Rausher, M. D., Rieseberg, L., and Moore, A. J. (2010). Data archiving. *The American Naturalist*, 175(2):145–146.