Sparse Binary Relation Representations for Genome Graph Annotation

Mikhail Karasikov[1,2,3], Harun Mustafa[1,2,3], Amir Joudaki[1,2], Sara Javadzadeh No[1], Gunnar Rätsch[1,2,3], and André Kahles[1,2,3]

[1] Department of Computer Science, ETH Zurich, Zurich, Switzerland
[2] University Hospital Zurich, Biomedical Informatics Research, Zurich, Switzerland
[3] SIB Swiss Institute of Bioinformatics, Zurich, Switzerland {raetsch,andre.kahles}@inf.ethz.ch

Abstract. High-throughput DNA sequencing data is accumulating in public repositories, and efficient approaches for storing and indexing such data are in high demand. In recent research, several graph data structures have been proposed to represent large sets of sequencing data and allow for efficient query of sequences. In particular, the concept of colored de Bruijn graphs has been explored by several groups. While there has been good progress towards representing the sequence graph in small space, methods for storing a set of labels on top of such graphs are still not sufficiently explored. It is also currently not clear how characteristics of the input data, such as the sparsity and correlations of labels, can help to inform the choice of method to compress the labels. In this work, we present a systematic analysis of five different state-of-the-art annotation compression schemes that evaluates key metrics on both artificial and real-world data and discusses how different data characteristics influence the compression performance. In addition, we present a new approach, Multi-BRWT, that shows an up to 50% improvement in compression performance over the current state-of-the-art and is adaptive to different kinds of input data. Using our comprehensive test datasets, we show that this improvement can be robustly reproduced for different representative real-world datasets.

Keywords: Sparse binary matrices · Binary relations · Genome graph annotation.

1 Introduction

Over the past decade, there has been an astronomical growth in the global capacity for generating DNA sequencing data [29]. Various sequencing efforts have started to amass data from populations of humans [30] and other organisms [31, 32]. For these well studied organisms, already assembled reference sequences are the common starting point for comparative and functional analyses. Unfortunately, a large proportion of DNA sequencing data, in particular data originating from non-model organisms or collected in metagenomics studies, are lacking a genome reference. Whereas general guidelines exist for the first [10], genome assembly for metagenomics is much less well defined. In addition to genome sequencing, a large amount of data is created from cDNA sequencing-based transcriptome analyses (and comparable techniques). Such data is often generated for a specific purpose but not analyzed further for questions outside of the scope of the studies that generated them. A consequence of this is that a large amount of raw sequencing data is publicly available but remains untouched. Its vastness and the currently lacking standards for indexing such data make an integrated analysis daunting even for field experts.

To make this host of data efficiently searchable, it is necessary to employ a search index. However, indexing techniques traditionally used for linear reference sequences, such as methods based on the FM-index [17], can not be readily applied to solve this task. To address this problem, different graph data structures for the representation of multiple genomes have been proposed. One class of structures are generalizations of the FM-index, such as generalized compressed suffix arrays on a genome graph [27, 11] or succinct de Bruijn graphs [7, 18]. Other possibilities include hashing-based implicit representations of de Bruijn graphs using Bloom filters [9, 26] or related representations [14, 28].

When building an index on the sequences alone, this approach merely allows for querying the presence of sequences in the constructed index. To support relating queries to information such as source genomes, haplotypes, or functional annotations, additional labels must be associated with the index. To facilitate this, approaches for storing additional data on an indexed graph have been suggested, such as the gPBWT [21] for genome graphs or succinct representations of *colored* (i.e. *labeled*) de Bruijn graphs [18, 15, 3] for the representation of sets of sequences. In this context, there has also been recent development towards a dynamic representation of such data has to be mentioned [19, 24].

The problem of efficiently representing these types of relations is also addressed in other fields. Commonly referred to as *compressed binary relations*, a growing body of theoretical work addresses such approaches [6]. Successful applications of similar techniques include the efficient representation of large web-graphs [8] and RDF data sets [4]. We will provide a more detailed description of some of these approaches in Section 2.

In this work, we present a new method for compressing binary relations. Providing as background a comprehensive benchmark of existing compression schemes, we show that our approach has superior performance on both artificial and real world data sets. Our paper has the following structure. After introducing our notation, we begin by defining the abstract graph and annotation structures that we wish to compress (Section 2.1). We then provide descriptions of our proposed compression technique and competing methods (Section 2.2). Finally, we compare the compression performance of these methods on different types of graph annotations (Section 3) and close with a brief discussion of our results and an outlook on future work (Section 4).

2 Methods

After introducing notation, we will give an overview of all methods implemented for this work and provide a description of our methodological contributions.

2.1 Preliminaries

We will operate in the following setting: We are given a k-dimensional de Bruijn graph $G = (V, E)$ over a set of given input sequences S. The node set V shall be defined as the set of all consecutive sub-sequences of length k (k-mers) of sequences in S

$$V = \{s_{i:i+k-1} \mid s \in S,\ i = 1, \ldots, |s| - k + 1\}, \tag{1}$$

where $s_{i:j}$ denotes the sub-sequence of s from position i up to and including position j, and $|s|$ is the length of s. Directed edges exist between any two nodes that overlap by $k - 1$:

$$E = \{(u, v) \mid u, v \in V,\ s.t.\ u_{2:k} = v_{1:k-1}\}, \tag{2}$$

where $u_{2:k}$ and $v_{1:k-1}$ describe the k-mers $u = u_{1:k}$ and $v = v_{1:k}$ without their first and last characters, respectively.

In order to represent relations between sources of the input sequences S and the nodes V, we now define the concept of a *labeled de Bruijn graph* and proceed by discussing the more general problem of representing a graph labeling.

Each node $v \in V$ that we refer to as object is assigned a finite set of labels $\ell(v) \subset L$. We represent this graph labeling as a binary relation $\mathcal{R} \subset V \times L$. A trivial representation of $\mathcal{R}$ taking $|V| \cdot |L|$ bits of space is a binary matrix $A \in \{0, 1\}^{|V| \times |L|}$. We will use A^i and A_j to denote its rows and columns, respectively.

In the following sections, we will discuss various methods described in the recent literature and present our improvements in efficiently representing $\mathcal{R}$. In addition to minimal space, we also require that the following set of operations can be carried out efficiently on the compressed representation of $\mathcal{R}$:

- `query_labels`$(v) = \{l \in L \mid (v, l) \in \mathcal{R}\}$ Given an object $v \in V$ (a k-mer in the underlying de Bruijn graph), return the set of labels $\ell(v)$ assigned to it.
- `query_objects`$(l) = \{v \in V \mid (v, l) \in \mathcal{R}\}$ Given a label $l \in L$ (e.g., a genome or sample ID), return the set of objects assigned to that label.
- `query_relation`(v, l) Given an object $v \in V$ and a label $l \in L$, check whether (v, l) is in the relation $\mathcal{R}$, `query_relation`$(v, l) = 1_{\{(v,l) \in \mathcal{R}\}}$.

2.2 Binary Relation Representation Schemes

For compressing the binary relation $\mathcal{R}$, we consider the following representations suggested in recent literature. As an abstraction, we will use the representation of $\mathcal{R}$ as a binary matrix $A \in \{0, 1\}^{|V| \times |L|}$ (referred to as the *binary relation matrix*) to illustrate the individual methods.

Column-major Sparse Matrix Representation As a simple baseline technique, we compress the positions of the non-zero indices in each column independently using Elias-Fano encoding [22]. While this method does not take into account correlations between columns for compression, it allows for parallel construction of the binary relation matrix by mapping the k-mers in each input sequence to the constructed graph in a separate process. This acts as an initial representation from which all other compression techniques are constructed.

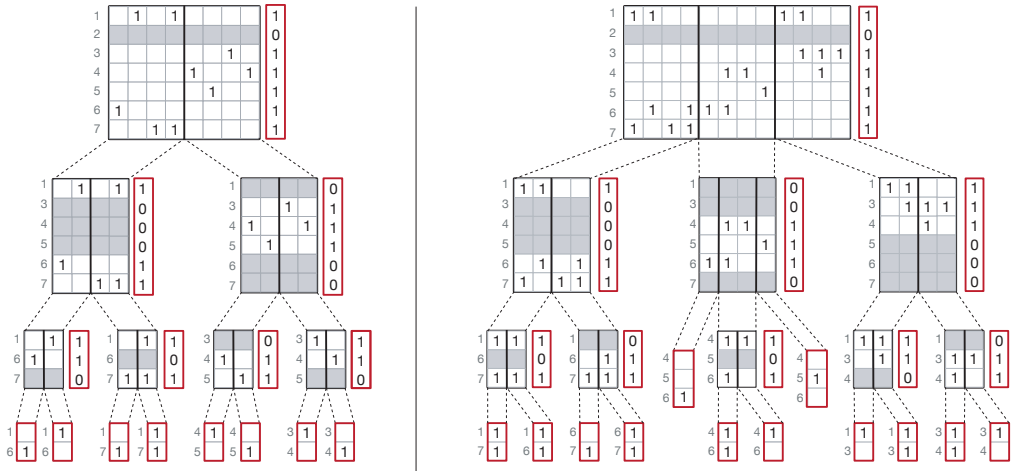
Flat Row-major Representation As a second baseline method, this representation concatenates all rows of A into a joint vector that is subsequently compressed using Elias-Fano encoding. This approach, for instance, is used by VARI [18] and its extensions [2].

Rainbowfish The current state-of-the-art for genome graph labeling is a row-major representation of the binary relation matrix A in which an optimal coding is constructed for the set of rows in A [3]. More precisely, let $A^{i_1}, \ldots, A^{i_r} \in \{0,1\}^{|L|}$ be all the unique rows in A, where $r \leq |V|$. Let $i : V \to \{1, \ldots, r\}$ be the mapping assigning each row in A to the corresponding index of the equal unique row. We then define the count vector $C \in \mathbb{N}^r$ such that $c_j = \#\{v \in V \mid i(v) = j\}$. We sort the values in C in non-increasing order to construct the permutation map $\sigma : \{1, \ldots, r\} \to \{1, \ldots, r\}$ (i.e. $c_i > c_j \implies \sigma(i) \leq \sigma(j)$).

To encode A, we start by permuting the unique rows according to σ, and form a matrix $A' \in \{0,1\}^{r \times |L|}$, $A'^{\sigma(t)}_j = A^{i_t}_j$. Then we compress the resulting matrix with the flat row major representation using an RRR vector [25] as the underlying storage technique and construct a *coding vector* $(\sigma(i(v)) - 1)_{v \in V}$ mapping the rows of the initial matrix A to the rows of encoded matrix A'. The coding vector is represented in a variable-length packed binary coding with a delimiter vector [3]. The delimiter vector is compressed with Elias-Fano encoding [22].

Binary Relation Wavelet Trees (BinRel-WT) This method involves a translation of the $|\mathcal{R}|$ non-zero elements of A into a string, which is then represented using a conventional wavelet tree [6]. Given the binary relation matrix, its set bits are iterated in row-major order and their respective column indices are stored contiguously in a string s over the alphabet $\{1, \ldots, |L|\}$. The lengths of each row of A are then stored in a delimiter vector d with unary coding. Finally, we represent s with a wavelet tree whose underlying binary vector is compressed into an RRR vector as well as for the delimiter vector.

Hierarchical Compressed Column-major Representation (BRWT) In contrast to BinRel-WT, BRWT acts directly on binary matrices without the translation to sequence [6]. First, an index vector I is computed by merging all matrix columns through bitwise-OR operations on its rows, $I_i = \bigvee_j A^i_j$, and stored to represent the root of the tree. Then, the rows composed entirely of 0s are discarded from A and two equal-sized submatrices A' and A'' (which may contain rows composed entirely of 0s) of the binary relation matrix A are passed to the left and right children of the root. The compression proceeds recursively. Construction terminates when a node is assigned a single column, which is stored as its index column (see Figure 1 a)). For reconstruction of matrix elements, it is sufficient to only store the index vectors associated with each node of the BRWT tree and the leaves.



a) Binary BRWT b) Multiary BRWT

Fig. 1: BRWT schematic representation. a) BRWT for the binary case. Grey rows correspond to all-0 rows, also indicated through the vector to the right of each matrix. Each child encodes only non-zero rows of the submatrix passed to it extracted from the matrix encoded in its respective parent. Numbers to the left of each matrix are the respective row-indices in the initial matrix. b) BRWT in the multiary case. Notation is as in the binary case. Stored vectors are shown in red.

In the next section we consider the problem of topology optimization in the BRWT tree construction. Afterwards, we propose two modifications to the BRWT construction scheme for improving its compression performance.

2.3 Multiary, topology-optimized BRWTs

Our first extension to the BRWT scheme is the introduction of an n-ary tree topology (BRWT_n), allowing for matrices to be split into more than two submatrices (see Figure 1 b)). The construction and querying for BRWT_n is analogous to the case of binary BRWT. In computational experiments on artificial and real data we show that in most cases, BRWT_n with arity greater than two but smaller than $|L|$ provides a higher compression ratio than the simple binary BRWT scheme (see Section 3). Note, that BRWT_n with the maximal allowed arity $n = |L|$ is equivalent to the baseline column-major sparse matrix representation.

To proceed with our second extension, let us consider binary relations with the number of labels much less than the number of objects, $|L| \ll |V|$. This condition is actually met in real annotated genome graphs where the number of k-mers, is usually in the billions and the number of labels is in the order of thousands (see Section 3.2).

Our second extension consists in introducing arbitrary assignments of columns from the matrices encoded in the nodes of the BRWT to their children. These assignments are represented by dictionaries stored in the BRWT nodes, but the $|L| \ll |V|$ constraint makes the space overhead from storing these negligible compared to the space needed to encode the index vectors. Thus, we exclude the problem of representing these assignments from further consideration and leave that as a small technical detail. Here and further, we refer to this proposed method as Multi-BRWT.

It is easy to see that all Multi-BRWT trees form a space where the root is defined for the initial matrix and the leaves correspond to the single columns of the initial binary matrix. Here, we leave out the question of enumerating these trees and counting the equivalence classes, but focus on the problem of choosing a tree structure that optimizes the compression ratio of Multi-BRWT.

Let us set this problem formally. Given a binary matrix A, let $\mathcal{T}$ be the space of all Multi-BRWT trees representing A. Let $Size(I)$ denote the size in bits of a compressed binary vector I (for instance, if I is of length n with m set bits, $Size(I) = n$ for an uncompressed bit vector, and $Size(I) \approx \lceil \log_2 \binom{n}{m} \rceil$ for RRR vectors [25]). We then neglect the space required for dictionaries defining the column assignments and we define the size of the Multi-BRWT tree $T \in \mathcal{T}$ as the space required to store all its index vectors including the vectors in leaves:

$$Size(T) := \sum_{i \in N} Size(I_i), \tag{3}$$

where N is the set of all nodes of the Multi-BRWT tree T. Thus, we wish to find an optimal Multi-BRWT tree by minimizing the storage space,

$$T^* = \arg\min_{T \in \mathcal{T}} Size(T).$$

We will refer to this as the Multi-BRWT problem.

By analogy to the NoSQL table compaction problem [12], it can be easily shown that Multi-BRWT constrained on the space of binary trees with the uncompressed bit vector representation as the underlying structure for storing the index vectors is NP-hard. Thus, we consider a two-step heuristic approach for finding an optimal Multi-BRWT structure. First, we build a binary Multi-BRWT tree by greedily matching the pairs of index columns according to their similarity: the number of shared set bits. Then, we optimize the arity of the chosen Multi-BRWT by selecting a subset N' from the set of internal nodes of the Multi-BRWT tree with its root r and all the leaves $v_1, \ldots, v_{|L|}$, $\{r\} \cup \{v_1, \ldots, v_{|L|}\} \subset N' \subset N$. To keep the resulting Multi-BRWT tree valid (allowing for reconstruction of the initial matrix A), we reassign all nodes left in N' without their parents to their grandparents, and repeat recursively if their grandparents are not present in N' as well. This naturally leads us to a greedy algorithm for optimizing the arity of the BRWT tree. Namely, starting the procedure in the leaves' parents and applying it to each node except for the root recursively, we estimate the cost of removing each current node by the following formula

$$\text{CostRem}(v) = \sum_{c \in \text{Children}(v)} Size(I'(c)) - \left[Size(I(v)) + \sum_{c \in \text{Children}(v)} Size(I(c)) \right],$$

where $I(v)$ denotes the index vector stored in the node v and $I'(c)$ denotes the index vector that will be stored in the node c after removing its parent v and reassigning the node c to its grandparent. Now we simplify the formula for estimating the cost of removing a node in BRWT by introducing an assumption that the size of a bit vector I of length n with m set bits is fully defined by these two parameters, i.e. $Size(I) = Size(n, m)$. Now, it is easy to see that after reassigning the node c with the index vector $I(c)$ of length n_c with m_c set bits to the parent of its parent v with index vector $I(v)$ of length n_v with m_v set bits, the node c updates and replaces its index vector $I(c)$ with a vector $I'(c)$ of length n_v with m_c set bits. This provides us with the following simplified formula for estimating the cost of removing a node from the BRWT tree

$$\text{CostRem}(v) = \sum_{c \in \text{Children}(v)} Size(n_v, m_c) - \left[Size(n_v, m_v) + \sum_{c \in \text{Children}(v)} Size(n_c, m_c) \right]. \tag{4}$$

Formula 4 can be efficiently computed without the need to actually rebuild the current structure of the BRWT. As a result, a decision about removing node v from the BRWT is made if the cost is negative $\mathrm{CostRem}(v) < 0$, leading thereby to a decrease of the BRWT in size. In our practical implementation we use the following formula for approximating the size required for storing an RRR bit vector [20] with the block size t: $Size(n, m) = \lceil \log_2 \binom{n}{m} \rceil + n\lceil \log_2(t+1)/t \rceil$.

Greedy algorithm for finding an initial tree approximation As an approximate solution to the first step, we propose a greedy algorithm in which an initial greedy pairwise matching step is performed on the columns to optimize their initial order prior to construction. Given the input columns $A_1, \ldots, A_{|L|}$ and their corresponding object queries $o_i = \texttt{query\_objects}(i)$, we first compute cardinalities of their pairwise intersections s_{ij}, $s_{ij} = |o_i \cap o_j|$. Then, we sort all computed similarities by decreasing order and perform match pairs of columns greedily.

Efficient pair-wise distance estimation The proposed greedy approximation method takes as input the matrix of the pairwise column similarities. For m columns of length n, computing each pairwise similarity costs $\mathcal{O}(n)$, and thus, computing the full similarity matrix takes $\mathcal{O}(nm^2)$, which is a considerable overhead for datasets with $m > 1000$ and $n \sim 10^9$. To make the procedure of estimating the pairwise similarities cheaper, we estimate the pairwise similarities from a sample rows of the matrix A. Moreover, we prove the following lemma to show that using just $\mathcal{O}\left(\frac{\log m}{\varepsilon^2}\right)$ random rows is sufficient for approximating the pairwise similarities with a small error ε with high probability.

Lemma 1. *Subsampling lemma: Suppose we are given sets $o_1, ..., o_m \subseteq \{1, \ldots, n\}$, with the minimal cardinality d defined as $d = \min_{i=1}^{m} |o_i|$. We sample the elements of $\{1, \ldots, n\}$ independently with the same probability p, to form a sampled set of objects $S \subset \{1, \ldots, n\}$ and subsampled sets designated by $\tilde{o}_i = o_i \cap S$. Now if we define $u_{ij} = |o_i \cup o_j|$ and its approximator $\hat{u}_{ij} = \frac{1}{p}|\tilde{o}_i \cup \tilde{o}_j|$, then for $p = \frac{12 \log m}{d\varepsilon^2}$ we claim:*

$$\Pr\Big(\bigcap_{i,j \in [m]} \Big\{ |\hat{u}_{ij} - u_{ij}| \leq \varepsilon u_{ij} \Big\} \Big) > 1 - \frac{1}{m^2}.$$

For a full proof, see Supplementary Section 2.

According to Lemma 1, with the subsampling technique we can approximate the cardinality of unions up to an ε-fraction with high probability, and therefore, approximate the column similarities required in the greedy matching algorithm as well. Note that similarly to the Johnson–Lindenstrauss lemma [5], the required number of sampled objects does not depend on the total number of objects n.

2.4 Implementation Details

We implement the underlying de Bruijn graph as a hash table storing k-mers packed into 64bit integers with 64bit indexes assigned to the k-mers, or as a complete de Bruijn graph represented by a mapping of k-mers to 4^k row indexes of the binary relation matrix.

In the column-major representation, the columns of the binary relation matrix are stored using SD vectors implemented in `sdsl-lite` [13]. The single long vector of the row flat representation is also compressed using SD vectors from `sdsl-lite`.

BinRel-WT (sdsl) compressor uses the implementation of wavelet tree from the `sdsl-lite` library, using an RRR vector to store its underlying bit vector. The delimiter vector uses the RRR vector implementation from `sdsl-lite`.

The BinRel-WT compressor uses the binary relation implementation from `https://github.com/dieram3/binrel_wt`.

Our BRWT is implemented as a tree in memory, compressing the index and leaf vectors as RRR vectors. To avoid multiple passes through the matrix rows, we construct the BRWT using a bottom-up approach. Given a fixed clustering of the matrix columns, the leaves of the BRWT are constructed first, followed by their parents constructed for the index vectors propagated from the children nodes. To speed up the greedy matching algorithm, we sample randomly 10^6 rows in each experiment and use those to approximate the number of shared bits in the input columns and the index vectors during the Multi-BRWT construction. For optimizing the tree arity, we use formula $Size(n, m) = \lceil \log_2 \binom{n}{m} \rceil + n\lceil \log_2(t+1)/t \rceil$ when estimating sizes of bit vectors of length n with m set bits in the RRR representation with a block size of $t = 127$.

All SD vectors are constructed with default template parameters, while all RRR vectors are constructed with a block size of 127.

Code Availability All methods implemented and evaluated in this paper are available at `https://github.com/ratschlab/genome_graph_annotation`.

2.5 Data

Simulated data To profile our compressors, we generated several different series of synthetic binary matrices of varying densities (see Supplementary Section 1 for a more detailed description). In total we generated three different kinds of series: i) random matrices with uniformly distributed set bits, ii) initially generated random matrix rows duplicated and permuted randomly, iii) initially generated random matrix columns duplicated and permuted randomly. The motivation behind these series is as follows: The best performing state-of-the art compressors exploit correlation between rows of the binary relation matrix [24]. However, the usual structure of annotated de Bruijn graphs often implies a correlation structure on the columns rather than on rows. While for a small number of columns and sufficient label redundancy this effect translates to row redundancy, for larger label sets this is not necessarily the case and approaches exploiting correlation structure on the columns might fare better. To test this hypothesis, the generated the different kinds of synthetic data, reflecting uncorrelated rows/columns, correlated rows, and correlated columns for series i), ii), and iii), respectively.

Real-world data For evaluating all approaches in a real-world setting, we have chosen two data sets well-known in the community and representative of typical applications.

Kingsford Human RNA-Seq This dataset consists of 2,652 Human RNA-Seq experiments originally drawn from [28] and subsequently used by [24] for comparison.

NCBI RefSeq This dataset consists of all 79,448 reference sequences from Release 88 of the NCBI RefSeq database [23]. Each sequence has been annotated with its associated `family` rank taxonomic ID from the NCBI Taxonomy [1]. This results in a total of 3,173 unique labels for the sequences.

3 Results and Discussion

3.1 Experiments on artificial data

Based on the artificial dataset described in Section 2.5, we evaluated the relation between characteristics of the binary relation matrix A and the compression performance.

Dependency of compression ratio on matrix structure We measure density of A as the number of set bits divided by the total size of A. For reference, the labels for a sequencing-based de Bruijn graphs typically exhibit very low densities, commonly $< 0.5\%$.

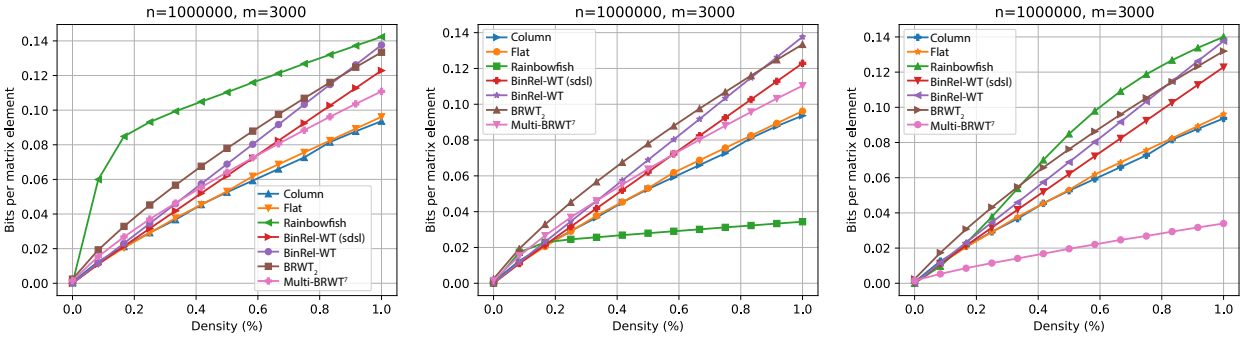


Fig. 2: Size of the representation of $A \in \{0,1\}^{10^6 \times 3 \cdot 10^3}$ with densities $d < 0.01$ using different approaches: a) Uniformly random bits; b) Uniformly random rows with multiplicity 5; c) Uniformly random columns with multiplicity 5. Approach c is closest to data in a real-world setting.

Especially in this low-density region, we find that the properties of the binary relation matrix have a strong effect on the compression ratio of individual methods. A second determinant of performance is whether any assumptions are made on the properties of the data.

On sparse, fully random data, the baseline compressors fare very well (Figure 2a), as no assumptions can be made about relationships. Notably, Rainbowfish, which exploits redundancy among rows, generates considerable overhead for very low densities. In the field of BRWT methods, the Multi-BRWT is closest to the best performing choices.

In the setting of correlated rows (data set ii, Figure 2b), as expected, Rainbowfish shows the strongest performance, clearly exploiting row redundancy. Again, among the BRWT methods the Multi-BRWT performs best.

Finally, in the setting that comes closest to a typical labeling task occurring on de Bruijn graphs derived from sequencing data (Figure 2c), the Multi-BRWT approach shows superior performance. Exploiting the correlated columns of the matrix, Multi-BRWT achieves a 5-fold improvement in compression ratio compared to Rainbowfish and more than 2-fold compared to the closest competitor. Notably, the baseline binary BRWT has no advantage over other baseline methods. Further, we observe that this performance gain increases with the total number of columns in the matrix (Supplemental Figures 1 and 2).

3.2 Experiments on real data

To compare the compression performance of our chosen methods under a variety of conditions, we have constructed two test datasets that exhibit different matrix sparsity characteristics.

Kingsford Human RNASeq (2,652 read sets) We filtered the 2,652 raw sequencing read sets with the KMC [16] tool to extract frequent unique canonical k-mers from each ($k = 20$). We used the same thresholds for the k-mer frequency level as in [24]. Using the k-mers extracted, we constructed a de Bruijn graph containing 3,693,178,415 nodes and annotated these with their source read sets, which resulted in 2,586 labels (66 filtered read sets were empty). As a baseline for comparison, we used the straightforward column compressed annotation, which required a total of 36.56 gigabytes (G) of space. We used this as a starting point to convert the annotation into other formats.

The results are summarized in Table 1. As expected, the simple row- and column-based compressors but also BinRel-WT require more than 30G in total. The current state-of-the art, Rainbowfish, reduces this by one third to 19.22G, exploiting correlated rows in the input matrix. The recently suggested BRWT benefits from column correlation through iterative grouping of columns and drastically improves on Rainbowfish, showing a 30% lower size. We further reduce this size through our generalized approach using multiary tree representations for BRWT. While some increase in arity reduces size compared to the binary case, a higher arity does not necessarily translate into lower space, as certain sub-matrices do not benefit from grouping. The lowest representation of a fixed arity is BRWT_7 with 12.13G.

We can improve compression performance of a binary BRWT through greedy pairing of node index vectors (see Section 2.3). This strategy further decreases size by another 25% to 9.68G. Finally, optimizing the tree topology using the Multi-BRWT method and selectively removing inner nodes (reassigning children to their grandparents) while maintaining a constraint on each node's maximum number of children, leads to the smallest space achieved in our experiments. Applying this technique, we decrease the required space to 9.19G (Multi-BRWT^5, with at most 5 children for each node), an almost 50% improvement over Rainbowfish.

RefSeq reference genomes Compression of the complete RefSeq genome annotation (release 88) resulted in a de Bruijn graph of dimension $k = 15$ containing $n = 1,073,685,700$ nodes, leading to a binary relation matrix of n rows and $m = 3,173$ columns with the density $\sim 3.8\%$, which is relatively high for a genome graph annotation and can be explained by the small k-mer size used.

This is a substantially larger dataset with less dependency between labels (columns). Due to the size of the graph annotation, we computed only the column compressed baseline, flat row major representation, our implementation of the BinRel-WT method (BinRel-WT (sdsl)), binary BRWT baseline, the greedily optimized binary BRWT representation, and the proposed Multi-BRWT^z for a subset of z values (cf. Table 1). We were able to a achieve a compressed storage size of only 42.28G. Also here, the BRWT approach improves drastically over the column compressed baseline, and the Multi-BRWT method considerably surpasses the baseline BRWT approach.

For our implementations, we would like to note that the other competing methods either exceeded our available memory (512G) or computed longer than 24h.

To reduce our memory footprint while constructing the Rainbowfish compression of the RefSeq annotation data, we attempted to use the annotation of the Mantis pipeline as a surrogate [24]. We would like to note that this pipeline does not natively support construction from FASTA files, and construction of a Mantis index from conversions of the RefSeq FASTA files to FASTQ failed due to inherent size limits of Mantis' data structures.

Table 1: The measured size of the compressed binary relation matrix for different methods in Gigabytes (G).

Methods	**Kingsford**	**RefSeq**
Column	36.56G	80.18G
Flat	41.21G	121.60G
Rainbowfish	19.22G	-
BinRel-WT	49.57G	-
BinRel-WT (sdsl)	31.40G	46.84G
$BRWT_2$	12.97G	51.82G
$BRWT_7$	12.13G	-
$BRWT_{10}$	12.28G	-
$BRWT_{13}$	12.61G	49.36G
$BRWT_2$ (greedy)	9.68G	48.10G
Multi-$BRWT^3$	9.36G	45.45G
Multi-$BRWT^5$	**9.19G**	42.75G
Multi-$BRWT^7$	9.21G	-
Multi-$BRWT^{20}$	9.22G	**42.28G**

4 Conclusion

We have presented a series of compressed representation methods for binary relations, building upon and improving on the existing literature. By generalizing BRWTs to multiary trees with improved partitioning schemes and adaptive arity to reduce data representation overhead, we have improved on state-of-the-art compression techniques for both simulated and real-world biological datasets.

We have shown that the structure of the input data has strong influence on the compression performance. Methods such as Rainbowfish or the Flat compressor benefit from correlated rows (objects with a similar set of labels). It is to note that in a real-world setting, where more and more labels are added to the set, the number of highly correlated rows decreases (ultimately leading to a set of mostly independent rows) and these methods work less well. Interestingly, it is especially this setting that regularly occurs in the labeling of genome graphs, were an underlying set of (related) sequences is assigned a growing set of different labels.

We have presented a method that copes very well with an increasing number of related columns and we could show that this results in considerable performance gains not only on synthetic, but also on typical real-world data. Our method, Multi-BRWT, led to an 80% decrease in compressed size compared to the baseline method and a 50% decrease compared to the closest competitor, Rainbowfish.

A natural extension of this work will involve the utilization of dynamic binary vectors in the underlying storage of BRWTs to allow for their use in dynamic database contexts. Of particular interest are the use of dynamic compressed structures to avoid expensive decompression and recompression steps when performing updates, and the prospects for post-construction column rearrangements to accommodate dynamic data.

By the nature of BRWT, frequent rows (i.e. those with high row density) would be present in more subtrees, and thus, exhibit greater row query times. One way to resolve this would be through hybrid row and column partitioning, in which rows exhibiting certain properties (such as high frequency) are excluded and compressed through techniques better suited to those submatrices.

Overall, we conclude that, despite the advancements in compression over the recent years, there is still much room and many degrees of freedom in compressor design for further improvement.

5 Acknowledgements

We would like to thank the members of the Biomedical Informatics group for fruitful discussions and critical questions, and Torsten Hoefler and Mario Stanke for constructive feedback on the graph setup. Harun Mustafa and Mikhail Karasikov are funded by the Swiss National Science Foundation grant #407540_167331 “Scalable Genome Graph Data Structures for Metagenomics and Genome Annotation” as part of Swiss National Research Programme (NRP) 75 “Big Data”.

References

1. Agarwala, R., Barrett, T., Beck, J., Benson, D.A., Bollin, C., Bolton, E., Bourexis, D., Brister, J.R., Bryant, S.H., Canese, K., Charowhas, C., Clark, K., DiCuccio, M., Dondoshansky, I., Feolo, M., Funk, K., Geer, L.Y., Gorelenkov, V., Hlavina, W., Hoeppner, M., Holmes, B., Johnson, M., Khotomlianski, V., Kimchi, A., Kimelman, M., Kitts, P., Klimke, W., Krasnov, S., Kuznetsov, A., Landrum, M.J., Landsman, D., Lee, J.M., Lipman, D.J., Lu, Z., Madden,

T.L., Madej, T., Marchler-Bauer, A., Karsch-Mizrachi, I., Murphy, T., Orris, R., Ostell, J., O'Sullivan, C., Palanigobu, V., Panchenko, A.R., Phan, L., Pruitt, K.D., Rodarmer, K., Rubinstein, W., Sayers, E.W., Schneider, V., Schoch, C.L., Schuler, G.D., Sherry, S.T., Sirotkin, K., Siyan, K., Slotta, D., Soboleva, A., Soussov, V., Starchenko, G., Tatusova, T.A., Todorov, K., Trawick, B.W., Vakatov, D., Wang, Y., Ward, M., Wilbur, W.J., Yaschenko, E., Zbicz, K.: Database Resources of the National Center for Biotechnology Information. Nucleic Acids Research (2017). https://doi.org/10.1093/nar/gkw1071

2. Alipanahi, B., Muggli, M.D., Jundi, M., Noyes, N., Boucher, C.: Resistome SNP Calling via Read Colored de Bruijn Graphs. bioRxiv (2018). https://doi.org/10.1101/156174
3. Almodaresi, F., Pandey, P., International, R.P.L.L., 2017, u.: Rainbowfish: A succinct colored de Bruijn graph representation. drops.dagstuhl.deSign in (2017), http://drops.dagstuhl.de/opus/volltexte/2017/7657/
4. Álvarez-García, S., Brisaboa, N.: Compressed k2-triples for full-in-memory RDF engines. arXiv preprint arXiv: ... (2011)
5. Baraniuk, R., Davenport, M., DeVore, R., Wakin, M.: A simple proof of the restricted isometry property for random matrices. Constructive Approximation (2008). https://doi.org/10.1007/s00365-007-9003-x
6. Barbay, J., Claude, F., Navarro, G.: Compact binary relation representations with rich functionality. Information and Computation (2013). https://doi.org/10.1016/j.ic.2013.10.003
7. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de Bruijn graphs. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2012). https://doi.org/10.1007/978-3-642-33122-0_18
8. Brisaboa, N.R., Ladra, S., Navarro, G.: k2-trees for compact web graph representation. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2009). https://doi.org/10.1007/978-3-642-03784-9_3
9. Chikhi, R., Rizk, G.: Space-efficient and exact de Bruijn graph representation based on a Bloom filter. Algorithms for Molecular Biology **8**(1), 22 (9 2013). https://doi.org/10.1186/1748-7188-8-22, http://almob.biomedcentral.com/articles/10.1186/1748-7188-8-22
10. Church, D.M., Schneider, V.A., Graves, T., Auger, K., Cunningham, F., Bouk, N., Chen, H.C., Agarwala, R., McLaren, W.M., Ritchie, G.R., Albracht, D., Kremitzki, M., Rock, S., Kotkiewicz, H., Kremitzki, C., Wollam, A., Trani, L., Fulton, L., Fulton, R., Matthews, L., Whitehead, S., Chow, W., Torrance, J., Dunn, M., Harden, G., Threadgold, G., Wood, J., Collins, J., Heath, P., Griffiths, G., Pelan, S., Grafham, D., E. Eichler, E., Weinstock, G., Mardis, E.R., Wilson, R.K., Howe, K., Flicek, P., Hubbard, T.: Modernizing Reference Genome Assemblies. PLoS Biology **9**(7), e1001091 (7 2011). https://doi.org/10.1371/journal.pbio.1001091, https://dx.plos.org/10.1371/journal.pbio.1001091
11. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., Paten, B., Durbin, R.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. nature biotechnology **36** (2018). https://doi.org/10.1038/nbt.4227, https://www.nature.com/articles/nbt.4227.pdf?origin=ppub
12. Ghosh, M., Gupta, I., Gupta, S., Kumar, N.: Fast Compaction Algorithms for NoSQL Databases. In: Proceedings - International Conference on Distributed Computing Systems (2015). https://doi.org/10.1109/ICDCS.2015.53
13. Gog, S., Beller, T., Moffat, A., Petri, M.: From theory to practice: Plug and play with succinct data structures. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2014). https://doi.org/10.1007/978-3-319-07959-2_28
14. Holley, G., Wittler, R., Stoye, J.: Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. Algorithms for Molecular Biology **11**(1), 3 (12 2016). https://doi.org/10.1186/s13015-016-0066-8, http://almob.biomedcentral.com/articles/10.1186/s13015-016-0066-8
15. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de Bruijn graphs. Nature Genetics (2012). https://doi.org/10.1038/ng.1028
16. Kokot, M., Długosz, M., Deorowicz, S.: KMC 3: counting and manipulating k-mer statistics. Bioinformatics **33**(17), 2759–2761 (2017). https://doi.org/10.1093/bioinformatics/btx304, http://dx.doi.org/10.1093/bioinformatics/btx304
17. Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics (2009). https://doi.org/10.1093/bioinformatics/btp324
18. Muggli, M.D., Bowe, A., Noyes, N.R., Morley, P.S., Belk, K.E., Raymond, R., Gagie, T., Puglisi, S.J., Boucher, C.: Succinct colored de Bruijn graphs. Bioinformatics (2017). https://doi.org/10.1093/bioinformatics/btx067
19. Mustafa, H., Schilken, I., Karasikov, M., Eickhoff, C., Rätsch, G., Kahles, A., Hancock, J.: Dynamic compression schemes for graph coloring. Bioinformatics (2018). https://doi.org/10.1093/bioinformatics/bty632
20. Navarro, G., Providel, E.: Fast, Small, Simple Rank/Select on Bitmaps. In: Proceedings of the 11th International Conference on Experimental Algorithms. pp. 295–306. SEA'12, Springer-Verlag, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30850-5_26, http://dx.doi.org/10.1007/978-3-642-30850-5_26
21. Novak, A., Paten, B.: A Graph Extension of the Positional Burrows Wheeler Transform and its Applications. Algorithms for Molecular Biology (2016). https://doi.org/10.1101/051409
22. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proceedings of the Meeting on Algorithm Engineering & Expermiments. pp. 60–70. Society for Industrial and Applied Mathematics (2007), https://dl.acm.org/citation.cfm?id=2791194
23. O'Leary, N.A., Wright, M.W., Brister, J.R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., Farrell, C.M., Goldfarb, T., Gupta, T., Haft, D., Hatcher, E., Hlavina, W., Joardar, V.S., Kodali, V.K., Li, W., Maglott, D., Masterson, P., McGarvey, K.M., Murphy, M.R., O'Neill, K., Pujar, S., Rangwala,

S.H., Rausch, D., Riddick, L.D., Schoch, C., Shkeda, A., Storz, S.S., Sun, H., Thibaud-Nissen, F., Tolstoy, I., Tully, R.E., Vatsan, A.R., Wallin, C., Webb, D., Wu, W., Landrum, M.J., Kimchi, A., Tatusova, T., DiCuccio, M., Kitts, P., Murphy, T.D., Pruitt, K.D.: Reference sequence (RefSeq) database at NCBI: Current status, taxonomic expansion, and functional annotation. Nucleic Acids Research (2016). https://doi.org/10.1093/nar/gkv1189

24. Pandey, P., Almodaresi, F., Bender, M.A., Ferdman, M., Johnson, R., Patro, R.: Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index. Cell Systems (7 2018). https://doi.org/10.1016/j.cels.2018.05.021, http://dx.doi.org/10.1016/j.cels.2018.05.021
25. Raman, R., Raman, V., Rao, S.S.: Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees and Multisets *. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 233–242. Society for Industrial and Applied Mathematics (2002), http://delivery.acm.org/10.1145/550000/545411/p233-raman.pdf
26. Salikhov, K., Sacomoto, G., Kucherov, G.: Using cascading Bloom filters to improve the memory usage for de Brujin graphs. Algorithms for Molecular Biology **9**(1), 2 (2 2014). https://doi.org/10.1186/1748-7188-9-2, http://almob.biomedcentral.com/articles/10.1186/1748-7188-9-2
27. Siren, J., Valimaki, N., Makinen, V.: Indexing Graphs for Path Queries with Applications in Genome Research. IEEE/ACM Transactions on Computational Biology and Bioinformatics **11**(2), 375–388 (3 2014). https://doi.org/10.1109/TCBB.2013.2297101, http://ieeexplore.ieee.org/document/6698337/
28. Solomon, B., Kingsford, C.: Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. Journal of Computational Biology **25**(7), 755–765 (7 2018). https://doi.org/10.1089/cmb.2017.0265, http://www.liebertpub.com/doi/10.1089/cmb.2017.0265
29. Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., Robinson, G.E.: Big data: Astronomical or genomical? PLoS Biology (2015). https://doi.org/10.1371/journal.pbio.1002195
30. Walter, K., Min, J.L., Huang, J., Crooks, L., Memari, Y., McCarthy, S., Perry, J.R., Xu, C., Futema, M., Lawson, D., Iotchkova, V., Schiffels, S., Hendricks, A.E., Danecek, P., Li, R., Floyd, J., Wain, L.V., Barroso, I., Humphries, S.E., Hurles, M.E., Zeggini, E., Barrett, J.C., Plagnol, V., Richards, J.B., Greenwood, C.M., Timpson, N.J., Durbin, R., Bala, S., Clapham, P., Coates, G., Cox, T., Daly, A., Du, Y., Edkins, S., Ellis, P., Flicek, P., Guo, X., Guo, X., Huang, L., Jackson, D.K., Joyce, C., Keane, T., Kolb-Kokocinski, A., Langford, C., Li, Y., Liang, J., Lin, H., Liu, R., Maslen, J., Muddyman, D., Quail, M.A., Stalker, J., Sun, J., Tian, J., Wang, G., Wang, J., Wang, Y., Wong, K., Zhang, P., Birney, E., Boustred, C., Chen, L., Clement, G., Cocca, M., Smith, G.D., Day, I.N., Day-Williams, A., Down, T., Dunham, I., Evans, D.M., Gaunt, T.R., Geihs, M., Hart, D., Howie, B., Hubbard, T., Hysi, P., Jamshidi, Y., Karczewski, K.J., Kemp, J.P., Lachance, G., Lek, M., Lopes, M., MacArthur, D.G., Marchini, J., Mangino, M., Mathieson, I., Metrustry, S., Moayyeri, A., Northstone, K., Panoutsopoulou, K., Paternoster, L., Quaye, L., Ring, S., Ritchie, G.R., Shihab, H.A., Shin, S.Y., Small, K.S., Artigas, M.S., Soranzo, N., Southam, L., Spector, T.D., St Pourcain, B., Surdulescu, G., Tachmazidou, I., Tobin, M.D., Valdes, A.M., Visscher, P.M., Ward, K., Wilson, S.G., Yang, J., Zhang, F., Zheng, H.F., Anney, R., Ayub, M., Blackwood, D., Bolton, P.F., Breen, G., Collier, D.A., Craddock, N., Curran, S., Curtis, D., Gallagher, L., Geschwind, D., Gurling, H., Holmans, P., Lee, I., Lönnqvist, J., McGuffin, P., McIntosh, A.M., McKechanie, A.G., McQuillin, A., Morris, J., O'Donovan, M.C., Owen, M.J., Palotie, A., Parr, J.R., Paunio, T., Pietilainen, O., Rehnström, K., Sharp, S.I., Skuse, D., St Clair, D., Suvisaari, J., Walters, J.T., Williams, H.J., Bochukova, E., Bounds, R., Dominiczak, A., Farooqi, I.S., Keogh, J., Marenne, G., Morris, A., O'Rahilly, S., Porteous, D.J., Smith, B.H., Wheeler, E., Al Turki, S., Anderson, C.A., Antony, D., Beales, P., Bentham, J., Bhattacharya, S., Calissano, M., Carss, K., Chatterjee, K., Cirak, S., Cosgrove, C., Fitzpatrick, D.R., Foley, A.R., Franklin, C.S., Grozeva, D., Mitchison, H.M., Muntoni, F., Onoufriadis, A., Parker, V., Payne, F., Raymond, F.L., Roberts, N., Savage, D.B., Scambler, P., Schmidts, M., Schoenmakers, N., Semple, R.K., Serra, E., Spasic-Boskovic, O., Stevens, E., Van Kogelenberg, M., Vijayarangakannan, P., Williamson, K.A., Wilson, C., Whyte, T., Ciampi, A., Oualkacha, K., Bobrow, M., Griffin, H., Kaye, J., Kennedy, K., Kent, A., Smee, C., Charlton, R., Ekong, R., Khawaja, F., Lopes, L.R., Migone, N., Payne, S.J., Pollitt, R.C., Povey, S., Ridout, C.K., Robinson, R.L., Scott, R.H., Shaw, A., Syrris, P., Taylor, R., Vandersteen, A.M., Amuzu, A., Casas, J.P., Chambers, J.C., Dedoussis, G., Gambaro, G., Gasparini, P., Isaacs, A., Johnson, J., Kleber, M.E., Kooner, J.S., Langenberg, C., Luan, J., Malerba, G., März, W., Matchan, A., Morris, R., Nordestgaard, B.G., Benn, M., Scott, R.A., Toniolo, D., Traglia, M., Tybjaerg-Hansen, A., Van Duijn, C.M., Van Leeuwen, E.M., Varbo, A., Whincup, P., Zaza, G., Zhang, W.: The UK10K project identifies rare variants in health and disease. Nature **526**(7571), 82–89 (10 2015). https://doi.org/10.1038/nature14962, http://www.nature.com/articles/nature14962
31. Weigel, D., Mott, R.: The 1001 Genomes Project for Arabidopsis thaliana. Genome Biology **10**(5), 107 (5 2009). https://doi.org/10.1186/gb-2009-10-5-107, http://genomebiology.biomedcentral.com/articles/10.1186/gb-2009-10-5-107
32. Zhang, G.: Bird sequencing project takes off. Nature **522**(7554), 34–34 (6 2015). https://doi.org/10.1038/522034d, http://www.nature.com/articles/522034d