BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 20 2013, pages 2572–2578
doi:10.1093/bioinformatics/btt460

Sequence analysis

Advance Access publication August 21, 2013

Genome compression: a novel approach for large collections

Sebastian Deorowicz[1,*], Agnieszka Danek[1] and Szymon Grabowski[2]

[1]Institute of Informatics, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland and
[2]Computer Engineering Department, Technical University of Łódź, Al. Politechniki 11, 90-924 Łódź, Poland

Associate Editor: Michael Brudno

ABSTRACT

Motivation: Genomic repositories are rapidly growing, as witnessed by the 1000 Genomes or the UK10K projects. Hence, compression of multiple genomes of the same species has become an active research area in the past years. The well-known large redundancy in human sequences is not easy to exploit because of huge memory requirements from traditional compression algorithms.

Results: We show how to obtain several times higher compression ratio than of the best reported results, on two large genome collections (1092 human and 775 plant genomes). Our inputs are variant call format files restricted to their essential fields. More precisely, our novel Ziv-Lempel-style compression algorithm squeezes a single human genome to ~400 KB. The key to high compression is to look for similarities across the whole collection, not just against one reference sequence, what is typical for existing solutions.

Availability: http://sun.aei.polsl.pl/tgc (also as Supplementary Material) under a free license.

Supplementary data: Supplementary data are available at *Bioinformatics* online.

Contact: sebastian.deorowicz@polsl.pl

Received on February 16, 2013; revised on July 5, 2013; accepted on August 5, 2013

1 INTRODUCTION

The DNA sequencing technology has become so affordable that there are several large-scale projects in which at least hundreds of individuals of some species are sequenced. From many perspectives, including the advent of personalized medicine, the *Homo sapiens* data belong to the most interesting, and this is the reason why large projects like the 1000 Genomes Project (1000GP) (The 1000 Genome Project Consortium, 2012) and the UK10K Project (http://www.uk10k.org), with thousands of human genomes sequenced so far, were initiated. Among such projects, the most ambitious perhaps is the Personal Genome Project (PGP) (Ball *et al.*, 2012), with genomes of 100 000 individuals as the anticipated outcome. Large repositories are built not only for human genomes, to mention 1001 Genomes Project (1001GP), with *Arabidopsis thaliana* genetic variation (http://www.1001genomes.org/about.html).

It is a well-known fact that two organisms of the same species are highly similar; it was estimated that the genomes of two persons are identical in 99.5% (Levy *et al.*, 2007). The huge amount of data obtained in the large-scale projects demands efficient ways of storing them. Taking into account the high similarity of organisms, it becomes obvious that some compression method may be effectively applied.

Compression of single genomic sequences is hardly efficient, as the best obtained compression ratios achieve a factor of 4 or 5 (Cao *et al.*, 2007; Manzini and Rastero, 2004; Pinho *et al.*, 2011) only. When realized that instead of the complete genomic sequence, storing only differences between it and some referential sequence is enough, the task became easier. In their seminal paper, Christley *et al.* (2009) showed how to store the description of variations between James Watson's (JW) genome and a referential genome in only 4.1 MB. However, the authors' prior knowledge was not only the reference sequence but also the single nucleotide polymorphism (SNP) map. As the input, they took the information about the SNPs and insertion or deletion (indels) variations between JW genome and referential genome, and compressed these data using some clever, but simple, techniques. Comparing with ~3.1 Gbases of human genome, this means ~750-fold compression.

In the following years, a number of articles on relative compression of genomes were published (Deorowicz and Grabowski, 2011; Kuruppu *et al.*, 2011; Pinho *et al.*, 2012; Wandelt and Leser, 2012). In all the articles, the input sequences were complete genomes, not differences between genomes and a reference genome. This complicates the compression problem, as it is necessary to find the differences between genomes without any prior knowledge and without a database of variants (i.e. SNP and indel database). The most successful of the algorithms seems to be GDC (Deorowicz and Grabowski, 2011), which differentially compressed a collection of 69 human genomes from Complete Genomics Inc. to 215 MB (3.1 MB per individual). It is based on the Ziv–Lempel (Salomon and Motta, 2010; Storer and Szymanski, 1982) paradigm and finds approximate matches between the genome sequences.

Recently, Pavlichin *et al.* (2013) showed how to improve the technique from Christley *et al.* (2009), compressing the JW genome to 2.5 MB, with very similar average results on multiple 1000GP genomes. The introduced novelties are partly biologically inspired, e.g. making use of tag SNPs characterizing haplotypes.

Another line of research concerns indexing genomic collections (or more generally, repetitive sequences), i.e. building data structures enabling fast pattern search in the genomes (Claude *et al.*, 2010; Do *et al.*, 2012; Gagie *et al.*, 2011, 2012; Kreft and Navarro, 2013; Mäkinen *et al.*, 2010). Such indexes are efficient when the index resides in the main computer memory, which is challenging considering the sheer volume of indexed data. Some of the listed works are rather theoretical and their

*To whom correspondence should be addressed.

implementations are not yet available, whereas the works that have been implemented (Claude *et al*., 2010; Gagie *et al*., 2012; Kreft and Navarro, 2013; Mäkinen *et al*., 2010) are tested on relatively small collections, not exceeding ~1 GB.

In this article, we try to answer the question how well a collection of genomes of the same species can be compressed, when *knowledge of the possible variants* is given. The cited works of Christley *et al*. (2009) and Pavlichin *et al*. (2013) are so far the only attempts to compress a (single) genome sequence with a variant database. In this work, we take two large collections of genomes (*H.sapiens* and *A.thaliana*) and try to exploit cross-sequence correlations in the variant loci. Our solution is a specialized Ziv-Lempel-style compressor, where the input sequences are basically formed by binary flags denoting if successive variants from the database were found in the individuals. This approach appears highly successful, allowing to store the human collection in 432 MB (395 KB per individual) and the plant collection in 110 MB (142 KB per individual). We point out that the general idea of exploiting common features for improved compression is known for some other NGS tasks, including compression of (both mapped and unmapped) reads (Bonfield and Mahoney, 2013; Cox *et al*., 2012; Hach *et al*., 2012; Jones *et al*., 2012; Popitsch and Haeseler, 2013).

In the next section, we present the input data and the general idea of our approach. Then, we show some details of the proposed compression algorithm. Finally, we evaluate the compressor. The last section concludes the article.

2 MATERIALS AND METHODS

2.1 Datasets

Large collections of genomes of the same species in public repositories are nowadays often represented as one reference genome and multiple variant files. There are several formats for storing the variants, e.g. variant call format (VCF) (Danecek *et al*., 2011) used in the 1000GP, general feature format (GFF) used in the PGP. These formats are much more compact than, e.g. FASTA (with raw genomic sequences), yet large collections may still require hundreds of gigabytes of storage.

We use two large datasets in the experiments. The publicly available database of Phase 1 of the 1000GP contains data for 1092 human individuals. The genomes are in VCF files, one file for each chromosome, and so there are 24 files in total. These files contain the information about each variant [SNP, insertion (INS), deletion (DEL) and structural variant (SV)] that was found in at least one genome in the dataset. The genomes are phased, i.e. there is information on which of the two chromosomes of each pair (or on none/both) each variant is found.

Similar information about variants is present in the 1001GP for *A.thaliana*. For this collection, we have 775 haploid sequences, each consisting of seven chromosomes. These data were scattered, and we gather them from four 'subprojects'.

Our research goal concerns only genome collection compression [similarly as in the works of Christley *et al*. (2009) and Pavlichin *et al*. (2013)], and VCF files usually contain much more information than needed to recover the DNA sequences (e.g. in FASTA format). We ignore the non-essential VCF fields, i.e. keep only the information on which positions the changes in each genome may be found. For the 1000GP dataset, it meant removing extra fields from the original VCF files. For the 1001GP dataset, we directly converted the available data to a stripped VCF (sub)format, which we call VCF minimal (VCFmin). We point out that these are valid VCF files. As a side note, let us remark that the GFF data (http://evidence.personalgenomes.org/guide_upload_and_annotated_file_formats) used in the large-scale PGP (Ball *et al*., 2012) are compatible with that of ours stored in VCFmin.

Table 1. Characteristics of the datasets used in the experiments

H.sapiens data			
Variant types	Total	Average found on	
		23 chromosome pairs	23 chromosomes
No. of SNPs	38 267 471	3 701 254	2 553 479
No. of 1-symbol insertions	392 515	97 023	67 586
No. of 2-symbol insertions	81 462	27 325	19 223
No. of longer insertions	103 918	35 502	25 205
No. of 1-symbol deletions	393 748	94 031	65 503
No. of 2-symbol deletions	166 554	38 316	25 795
No. of longer deletions	305 317	64 907	43 186
No. of SVs	14 422	746	462
Total no. of variants	39 725 407	4 059 104	2 800 439

A.thaliana data		
Variant types	Total	Average found on 5 chromosomes, chloroplast, mitochondria
No. of SNPs	13 123 220	552 825
No. of 1-symbol insertions	261 428	2271
No. of 2-symbol insertions	35 005	249
No. of longer insertions	5652	39
No. of 1-symbol deletions	1 046 670	29 194
Total no. of variants	14 471 975	584 579

The basic dataset characteristics are presented in Table 1 (URLs and other technical descriptions of the data, including preprocessing details for the 1001 dataset, are given in the Supplementary Material).

2.2 The compression algorithm

Our tool, Thousands Genomes Compressor (TGC), assumes that the input data are in VCFmin form. Such textual file consists of rows, one per each variant. A single row contains the following data:

- Description of a variant (position, type and information about the changes to the reference genome),
- Evidence of occurrence of the variant in each single genome.

The VCFmin files can be compressed quite efficiently by general tools, but much better results are possible. The biggest hurdle for a generic compressor is the 'non-locality' of the VCFmin format, i.e. the genomes are stored in columns, so the occurrences of the successive variants of the same genome are at long distances. This means that if two genomes have similarities, the compressor must find and encode their similar (identical) areas, which are far away and are relatively short (the description of the occurrence of a variant in a single genome takes a few bytes). The main idea of our algorithm is to transform the input data in a way to increase the locality and lengths of similar (identical) genome areas. To this end,

```
...
5 13957 DEL 1
6 13980 SNP C
7 30923 SNP T
8 46402 INS TGT
9 47190 INS A
...
1000 776646 SNP A
1001 776769 SV 15112 TGA
1002 776814 SNP G
...
```

Fig. 1. Excerpt of the variant database. Successive columns contain variant id, chromosomal location, type and change: (i) substituting symbol for SNPs, (ii) inserted symbols for INSs, (iii) number of removed symbols for DELs and (iv) number of removed symbols and (optionally) symbols to insert for SVs

we transform the VCFmin file containing N genomes to the following files:

- A single database of variants containing only the basic information about each variant (see Fig. 1); multiple variant alleles are also supported,
- $2N$ (for diploid) or N (for haploid genomes) bit-vectors. Value 1 at some jth position in this vector means that jth variant in the database is found in the genome. To reduce the space, these bit-vectors are packed into byte-vectors (8 bits in a byte). A byte is then the smallest processing unit in the compression scheme.

For example, if the 1000th variant in the database is 776646 SNP A, and the 1000th bit for some genome is 1, then we know that an SNP occurs in this genome at position 776646 and the resulting nucleotide there is A. The collection of the database of variants and the byte vectors is later called as variant database + byte vectors (VDBV) format. Using the dense byte vectors has a few advantages:

- Processing compact input is usually faster and less memory demanding than with more 'bloated' input;
- Identical patterns of successive variants in different genomes are represented with repeating byte sequences (which can be easily handled with standard dictionary compression techniques);
- Same variants in different genomes have the same positions, and thus encoding the repetitions of the common patterns in successive genomes is cheaper.

The resulting VDBV representation is already well compressible with generic tools, especially 7z, but universal solutions neglect some existing redundancy; in particular, they are not 'aware' about the aligned repetitions between genomes in our byte vectors. To exploit this, we devised a specialized compressor loosely based on the LZSS algorithm (Storer and Szymanski, 1982). Each vector is processed from left to right. At every analyzed position k, we look for the longest match (identical byte substring) starting at kth position in any previously processed byte vector. The match position restriction is obvious, as we look for common haplotypes, and matches elsewhere in the vectors are accidental and thus unlikely to be long. (Moreover, with the restriction, the matches need fewer bits to encode.) The already processed data from the vectors are indexed using two hash tables (HTs) to make the search faster; one HT is for searching for long matches and if none such is found then the other HT is used (some more details on the hash scheme are given in the Supplementary Fig. S2).

Similarly as for the classical LZSS algorithm, at each position we can find a match of length at least *mml* (minimal match length, set experimentally to five in our implementation) or a literal (if no sufficiently long match can be found). A match is described as a triple $\langle 1, vid, len\rangle$, where *vid* is the id of the vector in which we found the longest match, and *len* is the match length. A literal is represented with a pair $\langle 0, bv\rangle$, where *bv* is the value of the byte at position k in the byte vector. The first fields (flags), 0 or 1, distinguish between literals and matches. After processing a match, we shift to $(k + len)$th position, and in a case of literal to $(k + 1)$th position.

The sequence of pairs and triples is then compressed using an arithmetic coder (Salomon and Motta, 2010) (We use a popular and fast arithmetic coding variant by Schindler (http://www.compressconsult.com/rangecoder/), also known as a range coder.). Broadly speaking, arithmetic coder encodes each symbol occurrence on (in general, fractional) number of bits related to the probability of the symbol occurrence. These probabilities are estimated on the basis of already encoded symbols, i.e. if a symbol has occurred frequently, the corresponding probability is high and the number of bits spent for encoding it is small. We note that using Huffman coding instead for our data would result in a few percent compression loss.

There are several contextual models for compressing various fields, which means that different by-products of our scheme are handled based on different collected statistics (more details are given in the Supplementary data). The flags are compressed in one model. For the literals, another model is used, but before passing their byte values to the entropy coder, we do some trick. The byte value of the first literal after a match $\langle 1, vid, len\rangle$ is 'xored' (if its value is not 0) with the byte following the repetition in *vid*th byte vector, i.e. the byte of index $k + len$. We know these two byte values cannot be equal (otherwise the match could be extended by at least 1 byte). It was found experimentally that it is more likely to have a decreased number of resulting set bits in *bv* after the 'xor' operation than to have it increased (even if in most cases, the number of set bits is unchanged). In this way, the distribution of the *bv* values gets more skewed, which helps the compression.

Contextual compression models are usually more practical if they are not too large, and this is the reason why numbers from a broad interval are often split before being processed by a statistical model (this approach is used, e.g. in gzip, bzip2). Following this rule, in our scheme, the length of the match is stored in two parts, both compressed with an entropy coder. First, we compress the binary logarithm of the match length (rounded to an integer) and then the remaining bits needed to recover its length. More precisely, the first part is $\lceil \log_2(len - mml + 1)\rceil$, and if $len - mml \geq 2$, then the value of $len - mml - 2^{\lceil \log_2(len-mml+1)\rceil - 1}$ is encoded.

Similarly, the *vid* field is split into two (byte) parts: $\lfloor vid/256\rfloor$ and $vid - 256\lfloor vid/256\rfloor$, both encoded with an entropy coder.

The compression of variant database (excerpt of which is presented in Fig. 1) is rather straightforward. The main idea is to compress each variant type separately: SNP, insertion, deletion and SV. Thus, for each variant, we first store its type. The variants positions are then differentially encoded, i.e distances between consecutive SNPs, consecutive DELs, etc, are stored. Then, for SNP, we store the substituting symbol. For INS, we store its length and the symbols to insert. For DEL, only its length is stored. For SV, we encode the deletion length, the insertion length and finally (if necessary) the inserted symbols. All the values are encoded using a variant of arithmetic coding with appropriate contextual models (details can be found in the Supplementary data).

3 RESULTS

As mentioned earlier, for the evaluation of the proposed compression algorithm, we use two datasets of 1092 *H.sapiens* and 775 *A.thaliana* genomes. In all experiments, the data are processed chromosome by chromosome. This approach, typical in the genomic compression literature (see, e.g. Deorowicz and

Grabowski, 2011; Pavlichin *et al.*, 2013; Wandelt and Leser, 2012), reduces the memory footprint, speeds up computations and improves the compression ratio for the generic algorithms.

There are several tools for compressing genomic sequences in FASTA format. Unfortunately, the amount of our test data, ~6.8 TB of raw sequences if converted to FASTA, is so huge that the running times of some of those compressors would be counted in months. Thus, we started from a preliminary test in which we evaluated the most powerful as well as the most recent tools for only two human chromosomes (14 and 21) and also two plant chromosomes (1 and 4). The results are presented in Table 2. This and all further experiments were performed on a computer equipped with four 4-core 2.4 GHz AMD Opteron CPUs with 128-GB RAM running Red Hat 4.1.2-46 linux.

Two of the presented compressors may be considered fast with regard to the compression speed: ABRC (Wandelt and Leser, 2012) with ~100 MB/s (run with 8 threads) and GDC-normal (Deorowicz and Grabowski, 2011) with ~40 MB/s speed (only a serial implementation exists), while the others are by about one [GDC-ultra (Deorowicz and Grabowski, 2011) and RLZ (Kuruppu *et al.*, 2011)] or about two (7z) orders of magnitude slower. Interestingly, the only generic compressor in the tests, 7z, is the second best in the compression ratio (after GDC-ultra), but its compression speed is low (0.4 MB/s).

Table 2. Evaluation of genome compressors for four sets: 2184 sequences of *H.sapiens* chromosome 14 (237.6 GB) and chromosome 21 (106.5 GB), 775 sequences of *A.thaliana* chromosome 1 (23.9 GB) and chromosome 4 (14.6 GB)

Data	7z	RLZ	GReEn	ABRC	GDC Normal	GDC Ultra
H.sapiens						
Chromosome 14	1068	270	218	472	674	2455
Chromosome 21	1561	269	211	460	685	2397
A.thaliana						
Chromosome 1	242	86	64	67	154	254
Chromosome 4	234	80	59	61	141	230

Note: The values are compression ratios (rounded to the nearest integer) of the collection, understood as the original data size divided by the compressed size.

The memory available for the compression has a major impact on the compression ratio. For example, 7z was run with its maximum setting, 1 GB for its LZ-buffer (translating to >10 GB of total memory use), yet it fit only a small part of the input for *H.sapiens* data: <10 individuals (each of size ~108 MB) for chromosome 14 dataset and 21 individuals (each of size ~48 MB) for chromosome 21 dataset. This, supposedly, was the main reason for which its compression ratio in the latter case is significantly higher. The hypothesis is indirectly confirmed by the results for much shorter *A.thaliana* chromosomes, for which more individuals fit the 1-GB LZ-buffer and the compression ratios are close to GDC-ultra.

In the next experiment, we compared a few well-known generic compressors (gzip, bzip2, 7z) on VCFmin input files (Table 3). Compressed sizes (in megabytes) and compression times are reported for selected chromosomes and the collections in total. Surprisingly perhaps, the best compression was obtained by bzip2 compressor.

Table 4 is similar, but now the inputs are in our temporary 'dense' representation, VDBV. Here, the generic compressors are compared against our proposal, TGC. Two simple observations can be made: (i) the more compact of these two input representations, VDBV, is clearly more appropriate for the best generic compressor, 7z, both from the point of compression ratio and from speed; (ii) TGC is significantly better than VDBV+7z in both measured aspects, which demonstrates that designing a specialized compression algorithm was a worthy goal in this case.

The results are summarized in Table 5. For comparison purposes, we also show the sizes of the raw sequences as well as the compression results of the best compressor working on such representation, GDC-ultra. We resigned, however, from presenting the compression and decompression times of GDC-ultra, as it works on a completely different representation than VCFmin,

Table 3. Evaluation of universal compressors for variant data of 1092 (*H.sapiens*) and 775 (*A.thaliana*) samples

Data	VCF	VCFmin	VCFmin + gzip		VCFmin + bzip2		VCFmin + 7z	
	size	size	size	c-time	size	c-time	size	c-time
H.sapiens								
Chromosome 1	93 087	13 249	306.6	321	**138.9**	6259	144.2	2003
Chromosome 11	58 671	8351	196.5	205	**89.0**	3960	91.5	1400
Chromosome 21	16 065	2286	55.9	58	**25.7**	1090	26.9	467
Complete	1 223 470	173 505	4066.0	4114	**1849.2**	82 128	1936.8	27 352
A.thaliana								
Chromosome 1	—	4945	111.3	99	**63.3**	2596	86.4	1002
Chromosome 4	—	3386	76.3	67	**43.7**	1608	60.0	700
Complete	—	20 755	466.8	414	**265.0**	10 107	362.8	4100

Note: The input data are in VCF format without any non-essential fields (VCFmin). For comparison, the sizes of the original VCF files from the 1000GP are given. All sizes are in megabytes. Columns 'c-time' contain compression time in seconds. The values marked in bold indicate best compression. The extended version of this table (with results for all chromosomes) can be found in Supplementary Table S1.

Table 4. Evaluation of universal compressors and the proposed algorithm (TGC) for variant data stored in the intermediate VDBV format

Data	VCFmin	VDBV		VDBV + gzip		VDBV + bzip2		VDBV + 7z		TGC	
	size	size	c-time	size	c-time	size	c-time	size	c-time	size	c-time
H.sapiens											
Chromosome 1	13 249	890.6	136	368.9	311	326.3	251	55.9	1070	**32.3**	690
Chromosome 11	8351	560.1	86	238.7	200	194.8	160	33.1	598	**20.2**	392
Chromosome 21	2286	153.0	25	66.8	54	39.3	44	10.4	107	**6.3**	96
Complete	173 505	11 679.9	1819	4813.6	4075	3907.0	3396	733.8	11 176	**431.6**	8364
A.thaliana											
Chromosome 1	4945	405.8	64	117.3	121	102.9	107	34.7	395	**26.1**	231
Chromosome 4	3386	281.4	42	79.7	92	67.9	72	23.7	271	**18.2**	120
Complete	20 756	1722.5	270	489.0	526	424.1	453	143.5	1,673	**109.7**	815

Note: All sizes are in megabytes and times are in seconds. The 'VDBV c-time' column contains the conversion times from VCFmin format to VDBV format. In the remaining columns titled 'c-time', total compression time is given (i.e. 'VDBV + 7z c-time' denotes the sum VCFmin-to-VDBV conversion time and 7z compression time; 'TGC c-time' is the total TGC processing time, comprising the conversion to VDBV and the actual compression). Note that the variant database (part of the VDBV representation) is of size 933 MB for *H.sapiens* and 320 MB for *A.thaliana*. After compressing by TGC, their sizes (included in 'TGC size' column) are ~51.0 MB and 12.5 MB, respectively. The values marked in bold indicate best compression. The extended version of this table (with results for all chromosomes) can be found in Supplementary Table S2.

Table 5. Summary of the results of the universal, specialized and proposed compressors

Data	Total size [MB]	Size per individual [MB]	Ratio to VCFmin	Ratio to raw	Compression time [s]	Decompression time [s]
H.sapiens						
Raw	6 669 797	6107.873	—	—	—	—
GDC-ultra	2952	2.703	58.8	2259	—	—
VCFmin	173 505	158.887	—	—	—	—
VDBV	11 680	10.696	14.9	571	1819	1730
VCFmin + 7z	1937	1.774	89.6	3444	27 352	1951
VDBV + 7z	734	0.672	236.4	9088	11 176	1936
TGC	**432**	**0.395**	**402.0**	**15 453**	8364	2067
A.thaliana						
Raw	94 047	121.351	—	—	—	—
GDC-ultra	384	0.495	54.1	245	—	—
VCFmin	20 756	26.782	—	—	—	—
VDBV	1723	2.223	12.1	55	270	286
VCFmin + 7z	363	0.468	57.2	259	4100	239
VDBV + 7z	144	0.185	144.6	655	1673	316
TGC	**110**	**0.142**	**189.1**	**857**	815	363

Note: There are two columns containing ratios: 'Ratio to raw' tells how many times the compressed file is smaller than the genome sequences in FASTA format. 'Ratio to VCFmin' is the compression ratio according to the size of VCFmin files. For GDC-ultra, compression and decompression times are not given, as this compressor uses a different input form than others and such a comparison would be irrelevant. The values marked in bold indicate best compression.

which we use in the article. The compression ratios of GDC can be treated as a reference point.

The most important 'numbers' from this summary are the average sizes of genomes in the most compact, TGC, representation. The obtained 395 KB (for the human data) is more than six times smaller than offered by the best so far genome sequence, GDC-ultra, compressor. Also, the very recent paper by Pavlichin *et al*. (2013), working on a representation similar to our VCFmin, reports more than six times larger files. When expressing the compression ratios in relation to the raw genome sequence sizes, it means that our algorithm squeezes *H.sapiens* genomes ~15 500 times and *A.thaliana* ~850 times.

In the last experiment, we compared TGC against SpeedGene (Qiao *et al*., 2012), an algorithm for efficient storage of SNP datasets. For an honest comparison, we restricted the 1000GP set of variants to SNPs only. SpeedGene requires the input data to be in LINKAGE format, in which there is no distinction between chromosomes in each pair, i.e. for each SNP it describes only whether no SNP is found in a genome, one SNP (on any chromosome) is found, two SNPs are found (on both chromosomes) or the status of SNP is unknown. Thus, we changed our algorithm slightly and instead of processing each single chromosome, we joined chromosomes of each pair, and obtained vectors of 'dibits', i.e. bit pairs. These vectors are then transformed into

Table 6. Comparison of the ways of storing the dataset of SNPs from the 1000GP

Data	No. of SNPs	VCFmin (SNP only)	LINKAGE/ PLINK	SpeedGene	TGC
Chromosome 1	2 896 960	12 687	791.8	180.5	**42.4**
Chromosome 11	1 827 284	8003	498.8	115.3	**26.9**
Chromosome 21	497 824	2180	135.9	33.0	**8.0**
Complete	38 267 471	167 569	10 444.4	2426.4	**564.8**

Note: VCFmin means a simplified VCF with SNP calls only that spends only 4 bytes for each genotype. All sizes are in megabytes. The values marked in bold indicate best compression.

byte vectors (each byte contains four consecutive dibits). In this way, our tool can compress these byte vectors without any change. Table 6 presents the compressed sizes obtained by SpeedGene and TGC. We show the results for three chromosomes, as well as for the complete genome. As one can see, TGC reduces the dataset size more than four times better than SpeedGene.

4 DISCUSSION

We examined the possibility of obtaining much better compression ratios of genomic collections than from existing tools, when additional knowledge is given. The knowledge was the information about the possible variants in genomes and the occurrence of these variants in specific genomes. This helps a lot in compression of genomic sequences, as all input sequences are perfectly aligned and the task of finding repetitions in data (usually the most important and time-consuming task handled by data compression algorithms) becomes rather simple. We should mention that in theory such perfect alignments can be found by compression algorithms, but the computational burden would be enormous. Thus, compression tools usually make some heuristic decisions when comparing the sequences in the hope that they do not lose too much.

The success of our algorithm was possible not only because of the variant database, but also because we searched for cross-correlations between individuals. In other words, for each individual, similarities to any other previously processed individual (i.e. runs of repeating variants) can be found. In principle, the available memory may be a limiting factor but processing the collection on the chromosome level resulted in <2.5 GB memory use for the larger (human) of the tested datasets. In the future, when much more genomes are available, we may need to re-address the memory issue though, possibly via working on blocks smaller than whole chromosomes, or trying to re-order the sequences in a way to maximize local similarities.

In the compression method design, we sometimes traded compression ratio for reduced memory requirements, e.g. some (rather minor) improvements in compression would be possible owing to higher-order contextual modeling. Probably, a more practical approach is to make use of more biological knowledge; the very recent work of Pavlichin *et al*. (2013) gives new insight, which might be possible to use in our scheme, but we leave it for future work.

Why such experiments can be interesting? Although accurate and efficient analyses of such huge (several terabytes in raw format) genomic collections remain a major challenge, we believe that the mere compressibility of human genomes (e.g. as a 'lower bound' for memory requirements of future algorithms and tools) is a question worth investigating. For example, our compressed collection takes ~430 MB, so including also a compressed reference genome (at most 700 MB) requires ~1.1 GB of space, which seems quite modest. Naturally, running efficient queries over such data is another matter (clearly with some overhead in space use), but our results suggest *this is not impossible*.

The information kept in VCF or genome variation format (GVF) files is often more detailed (e.g. may include quality scores) than what our tool preserves. Although clearly efficient compression methods for such data are also needed, we do not anticipate a possibility to obtain similar compression ratios to TGC, unless a (strongly) lossy mode is used. Unfortunately, we cannot see a way to easily adapt our compression techniques for such data.

TGC allows extracting an arbitrary chromosome (or a whole genome) from the compressed collection, yet this solution is simple and rather slow. Making this extraction faster, or (even better) allowing for quick access to position-restricted arbitrary snippets of the genomes in the collection, is an important task left for future work. Clearly, there must be some space-time tradeoffs for such functionalities. A somewhat related functionality will be to add or remove an individual genome to/from the collection. Currently, changing the archive content requires recompressing the collection from scratch.

The performed experiments showed that even the best genomic sequence compressor, GDC-ultra, is significantly (up to seven times) poorer in compression ratio than what can be obtained with extra knowledge. The main conclusions from our work are:

- Modern genomic sequence compressors cannot come close in compression ratio to the proposed algorithm basically because of two, not fully independent, reasons: (i) (almost) all of them ignore external knowledge (variant information), and (ii) working on consensus sequences is extremely resource-consuming and keeping full statistics needed for efficient compression is practically impossible for a large collection even on a 128-GB machine.
- Even huge human genome databases can be stored in relatively small space, as the data size of a single individual is only 395 KB on average. When extrapolated, this would mean that modern 2-TB hard drive is sufficient to store the genomes of ~5 million humans, size of a large city.

ACKNOWLEDGEMENTS

We thank the authors of SpeedGene for providing us with the source codes of their tool, used in the experiments, and the anonymous reviewers for helpful suggestions.

Funding: The work was partially supported by the Polish Ministry of Science and Higher Education under the project [DEC-2011/01/B/ST6/06868] (S.D.) and by the European Union from the European Social Fund [UDA-POKL.04.01.01-00-106/09] (A.D.).

Conflict of Interest: none declared.

REFERENCES

The 1000 Genome Project Consortium. (2012) An integrated map of genetic variation from 1092 human genomes. *Nature*, **491**, 56–65.

Ball,M.P. *et al.* (2012) A public resource facilitating clinical use of genomes. *Proc. Natl Acad. Sci. USA*, **109**, 11920–11927.

Bonfield,J.K. and Mahoney,M.V. (2013) Compression of FASTQ and SAM format sequencing data. *PLoS One*, **8**, e59190.

Cox,A.J. *et al.* (2012) Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinformatics*, **28**, 1415–1419.

Cao,M.D. *et al.* (2007) A simple statistical algorithm for biological sequence compression. In: *Proceedings of the Data Compression Conference IEEE Computer Society Press*. IEEE Computer Society, Washington, DC, USA, p. 4352.

Christley,S. *et al.* (2009) Human genomes as email attachments. *Bioinformatics*, **25**, 274–275.

Claude,F. *et al.* (2010) Compressed *q*-gram indexing for highly repetitive biological sequences. In: *Proceedings of the 10th IEEE Conference on Bioinformatics and Bioengineering*. Philadelphia, Pennsylvania, USA, pp. 86–91.

Danecek,P. *et al.* (2011) The variant call format and VCFtools. *Bioinformatics*, **27**, 2156–2158.

Deorowicz,S. and Grabowski,S. (2011) Robust relative compression of genomes with random access. *Bioinformatics*, **27**, 2979–2986.

Do,H.H. *et al.* (2012) Fast relative Lempel-Ziv self-index for similar sequences. In: Snoeyink,J. *et al.* (ed.) *Proceedings of the Joint International Conference on Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM)*. Springer, Beijing, China, pp. 291–302.

Gagie,T. *et al.* (2012) A faster grammar-based self-index. In: Dediu,A.H. and Martín-Vide,C. (eds) *Proceedings of the 6th International Conference on Language and Automata Theory and Applications*. Springer, A Coruña, Spain, pp. 240–251.

Gagie,T. *et al.* (2011) Faster approximate pattern matching in compressed repetitive texts. In: Asano,T. *et al.* (ed.) *Proceedings of the 22nd International Symposium on Algorithms and Computation*. Springer, Yokohama, Japan, pp. 653–662.

Hach,F. *et al.* (2012) SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, **28**, 3051–3057.

Jones,D.C. *et al.* (2012) Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Res.*, **40**, e171.

Kreft,S. and Navarro,G. (2013) On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, **483**, 115–133.

Kuruppu,S. *et al.* (2011) Optimized relative Lempel-Ziv compression of genomes. In: Reynolds,M. (ed.) *Proceedings of the ACSC Australasian Computer Science Conference*. Australian Computer Society, Inc., Sydney, Australia, pp. 91–98.

Levy,S. *et al.* (2007) The diploid genome sequence of an individual human. *PLoS Biol.*, **5**, e254.

Mäkinen,V. *et al.* (2010) Storage and retrieval of highly repetitive sequence collections. *J. Comput. Biol.*, **17**, 281–308.

Manzini,G. and Rastero,M. (2004) A simple and fast DNA compressor. *Software Pract. Ex.*, **34**, 1397–1411.

Pavlichin,D. *et al.* (2013) The human genome contracts again. *Bioinformatics*, **29**, 2199–2202.

Pinho,A.J. *et al.* (2011) On the representability of complete genomes by multiple competing finite-context (Markov) models. *PLoS One*, **6**, e21588.

Pinho,A.J. *et al.* (2012) GReEn: a tool for efficient compression of genome resequencing data. *Nucleic Acids Res.*, **40**, e27.

Popitsch,N. and von Haeseler,A. (2013) NGC: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic Acids Res.*, **41**, e27.

Qiao,D. *et al.* (2012) Handling the data management needs of high-throughput sequencing data: SpeedGene, a compression algorithm for the efficient storage of genetic data. *BMC Bioinformatics*, **13**, 100.

Salomon,D. and Motta,G. (2010) *Handbook of data compression*. Springer, London.

Storer,J.A. and Szymanski,T.G. (1982) Data compression via textual substitution. *J. ACM*, **29**, 928–951.

Wandelt,S. and Leser,U. (2012) Adaptive efficient compression of genomes. *Algorithms Mol. Biol.*, **7**, 30.