CAN-zip – Centroid Based Delta Compression of Next Generation Sequencing Data

Edward Steere[1], Lin An[2], Liqing Zhang[3]
University of the Witwatersrand, Johannesburg, South Africa[1]
Georgetown University, Georgetown, USA[2]
Virginia Tech, VA, USA[3]

***Abstract*—We present CANzip, a novel algorithm for compressing short read DNA sequencing data in FastQ format. CANzip is based on delta compression, a process in which only the differences of a specific data stream relative to a given reference stream are stored. However CANzip uniquely assumes no given reference stream. Instead it creates artificial references for different clusters of reads, by constructing an artificial representative sequence for each given cluster. Each cluster sequence is then recoded to indicate only how it differs relative to this artificially created reference sequence. Remodeling the data in this way greatly improves the compression ratio achieved when used in conjunction with commodity tools such as bzip2. Our results indicate that CANzip outperforms gzip on average and that it can outperform bzip2.**

I. INTRODUCTION

Contemporary bioinformatics and biology research projects have the potential to address some of the world's most challenging health problems. However landmark projects such as the 1000 Genomes Project [3], are producing a deluge of important and relevant data which is difficult to process.

Current technology trends [16] are reaching a plateau in terms of capacity for storage and transmission of data. This is a cost and storage problem for researchers in these fields. Compression of data is one means to address this problem. The compression of genetic data presents two key challenges. Firstly, because of the size of the data, algorithms which are inefficient in terms of speed and memory must be avoided. Secondly, although it remains controversial, the lossless compression algorithms for genetic data are preferred over lossy ones as every bit of information might prove to be useful for later research. These two factors make it difficult to achieve good data compression ratios.

Section II of this paper reviews some basic compression techniques. *Section III* examines how these techniques have been applied to the compression of genetic data. *Section IV* introduces the CANzip algorithm. *Sections V & VI* present the testing procedures and results of CANzip on select data sets. *Sections VII & VIII* briefly discusses the research noting the potential for future research. *Section IX* concludes the paper.

II. COMPRESSION PRIMER

In this section we survey the fundamental algorithms used in compression. The material is well established and is available in textbooks such as [14]. We review four types: entropy encoding, dictionary based encoding, the Burrows Wheeler transform and delta compression.

A. Entropy Encoding

Entropy codes serve to recode source data so that fewer bits are used to represent symbols which occur with high frequency in the source data. Two entropy encoding algorithms are commonly used.

Huffman encoding is an optimal prefix free scheme which is fast, and has little overhead. It does not, however, produce an entropy optimal encoding.

Huffman encoding begins with a collection of singleton tries for each symbol frequency in the source. Repeated merging of the roots of the two tries with the lowest frequencies leads to a single trie which corresponds to the binary encoding of every symbol [14]. A Huffman encoding will converge to within $p_{max}+0.086$ of the optimal entropy encoding rate – where p_{max} is the relative frequency of the highest frequency source symbol [14]. However, highly skewed alphabets cause a large p_{max}, resulting in poor compression rates.

Arithmetic encoding is an entropy optimal encoding which (unlike Huffman encoding) doesn't recode symbols. Instead the entire message is represented as a fraction between zero and one, with the placement in the range, along with source symbol frequencies, defining the encoded message.

Huffman encoding and arithmetic encoding yield effective encodings of source symbols without a priori information. However more effective compression can be achieved if the structure of the underlying data is identified.

B. Dictionary Based Encoding

Dictionaries of frequently repeating patterns can be used in compression algorithms. In such cases the algorithm usually constructs a dictionary of patterns found in the source data. A reference to the location of a pattern may then be used to represent patterns which recur in the source.

In 1977 Lempel and Ziv proposed the first such algorithm which they named LZ77 [17]. Since then various adjustments have been made to LZ77; such as LZW, LZ78 and LZMA [14]. Variants of these algorithms are found in general purpose compression tools such as gzip and 7zip [6, 13].

C. The Burrows Wheeler Transform

Observe that "This sentence should be compressed" is more difficult to compress than a permutation of the same sentence "ssss eeeenncc mprdhh bi tTnosulddc." Clearly the ordering of symbols in the source has a significant effect on its capacity to be compressed. The Burrows Wheeler Transform (BWT) is an efficient method for determining effective permutations of source symbols thereby enabling more efficient compression of data [1].

bzip2 is a popular open source compression tool which uses the BWT followed by a Huffman encoding. The combination of these two algorithms yields a better compression than Huffman encoding on it's own [15].

D. Delta Compression

Delta compression represents data as a set of changes from a specified reference data set. This form of compression is effective when each source has a low number of differences (or no differences at all) from a reference source, as in the case between the genomes of organisms of the same species.

Next Generation Sequencing (NGS) is particularly interesting in the context of this research. NGS achieves high throughput by sequencing small fragments of DNA sourced from tens of copies of the same genome. This process generates redundancy in both the high number of replicate sequences, as well as redundancy in the completed sequence itself. Our research takes advantage of the redundancy in the replicate sequences.

III. CONTEMPORARY RESEARCH

General purpose compression tools trade effectiveness in compression for ease of use and generality. A general purpose tool enables collaboration through the adoption of standards which are widely used. In addition we have found through use that general purpose tools handle most file types well. However, a general model fails to highlight all the redundancy in every dataset. This is especially true in the compression of genetic information. For example a gzip encoding is usually

no smaller than simply recoding every base to two bits in a fixed length code [10].

We now present six contemporary approaches to compressing genetic data. Hach et al. examined algorithms which rearrange the ordering of sequences to highlight patterns to general purpose compression tools [5]. Using this technique the compression ratios of both gzip and bzip2 were improved by a factor of 4.19 [5].

Grassi et al. separated quality scores from sequences and compress the two streams with commodity compression tools [8]. Separating the sequences from quality scores in a FastQ formatted file (a common NGS format) produces two files with very distinct dictionaries [8]. When compared with a combined stream, their method resulted in approximately 5.3 times better compression of FastQ formatted data [8].

Korodi et al. noted that FastQ formatted sequences use natural language in annotations [12]. The use of grammar models and higher order Markovian models resulted in a 2 times improvement in the compression of highly annotated sequences using bzip2 and gzip [12].

Janin et al. realised that the quality scores, like images, represent a high amount of detail which cannot be compressed with lossless compression [10]. It was found that a 0.68 bit per quality score encoding could be achieved using a smoothing method on BWT transformed quality scores without significant loss in information [10].

The most promising results have been achieved using delta compression, i.e., where only the differences from a reference are maintained. Using an available reference genome it has been shown that a genetic sequence may be compressed by as much as a factor of 0.39 of the original file size [7].

In certain circumstances a reference genome is not available. In the absence of a reference genome there are two known methods for delta compressing short-reads. These algorithms use highly efficient De Novo assembly and clustering. For example, Jones et al. used a streaming assembly algorithm to create the reference on line. This allows for a progressive delta encoding of the data to be achieved [11]. Zhang et al. proposed that reads are clustered and then delta compressed within clusters to the mean sequence of each cluster [16].

IV. CANzip ALGORITHM

We present the CANzip algorithm as an effective delta compression scheme for short-reads generated through NGS. CANzip is based on the approach proposed in [16]. Our contributions include an improved encoding scheme, and modifications to the steps in the algorithm. In addition CANzip assumes that no reference genome is available.

Central to the success of the algorithm is the concept of coverage in NGS data sets which are comprised of millions of reads of approximately a hundred bases. In order for a genetic sequence to be assembled from the small fragments generated in NGS, the target genome is "covered" many times over. That is if we assembled all of the data perfectly the result would be several copies of the same genome. In NGS this condition is necessary in order to increase the quality of an assembly. It stands to reason that with a high coverage there are many similar reads. If similar reads are clustered then significant redundancy becomes apparent.

CANzip seeks to leverage this redundancy. It does so by first creating a representative sequence for each of the clusters. It then notes only the differences which each member has to its representative sequence. This Delta compressed sequence can then be further compressed using a commodity compression tool. This process yields a better compression ratio than using a commodity tool on its own.

The algorithm consists of four stages – illustrated in *Figure 1*.

A. Stage 1: Separation

The most common format for NGS data is the FastQ format. The FastQ format stores a name, the sequence itself,

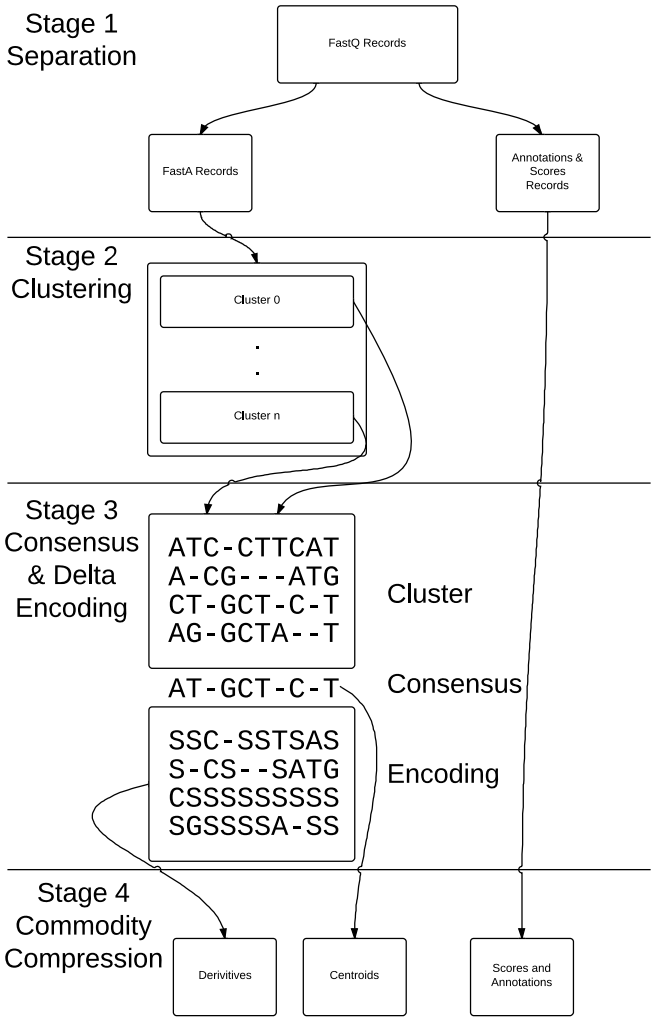

Fig. 1. Illustration of the four stages involved in the CANzip algorithm.

extra annotations and a string with a score per nucleotide on four separate lines.

The annotations and scores don't contain the same redundancy as the strings. We have therefore opted to separate the strings from the annotations and scores and compress the streams separately.

As illustrated in *figure 1*, we consider the strings for the next two stages and only re-introduce the scores and annotations at the final stage of the program.

B. Stage 2: Clustering

Sequence clustering is computationally expensive. For example the well known k-Means algorithm has smoothed complexity given in *Equation 1* [2].

$$\frac{O(1).n^{34}k^{34}ln^4(n)}{\sigma^6} \tag{1}$$

There are alternatives to k-Means which trade accuracy for running time, or which learn the number of clusters thereby improving the quality of the clustering process. Unfortunately, the large size of typical NGS sets prohibits the use of such algorithms due to time and memory constraints[1].

A practical approach is to cluster sequences using a greedy algorithm. The algorithm proceeds as illustrated in *Listing 1*.

```
clusters = {}
for sequence in sequences
    match <- false
    for cluster in clusters
        if distance ( sequence, cluster ) < T
            cluster <- sequence
            match   <- true
            break
    if not match
        clusters <- cluster {sequence}
```

Listing 1. Greedy clustering algorithm

Sequence distance (similarity) is ordinarily calculated using the number of insertions or deletions necessary to transform one sequence to another. These methods are based on alignments of the two sequences. However, sequence alignment is $O(n^2)$, which is unacceptable when the data set contains a large number of reads.

Instead the $d2$ distance metric is used – which is $O(n)$. $d2$ is a measure of the number of $k-grams$ which aren't shared by each sequence. This implies that we would have to count the number of $0-grams$, $1-grams$ and so forth until l where l is the length of the sequence. This too would

[1] An additional complication is the complexity of computing a high quality similarity measure for two sequences

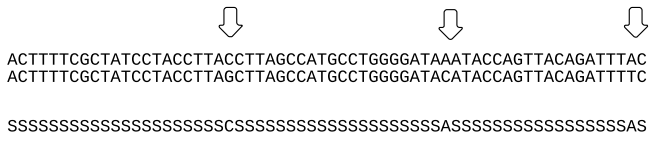

Fig. 2. Illustration of how a typical sequence is transformed to a derivative sequence given its centroid.

be unacceptable. It has been established that setting k to 6 produces satisfactory results [9].

C. Stage 3: Consensus & Delta Encoding

A consensus sequence is derived from the sequences in each cluster and is used as the representative for that cluster. The process proceeds as follows.

First the sequences are aligned to each other using a Multiple Sequence Alignment algorithm. This may be achieved with programs such as MUSCLE [4].

The modal base in each position across all sequences in a cluster is selected as the base in the corresponding position in the consensus sequence.

Delta encoding follows introducing a high degree of monotony in the data as follows. Each sequence is compared to it's cluster's representative. Each base which matches the corresponding base in the representative is replaced by the character "S." Each base which doesn't match the corresponding base in the representative is maintained.

Ideally one would like to see a number of derivative sequences in each cluster which are predominantly S's as this would indicate that there is a high delta compression rate.

This process is illustrated in *Figure 2*. Here we see that the original sequence only differs from the centroid at positions 21, 42 and 60. Note that at these indices in the derivative sequence the original value has been maintained, but at all other indices the original base has been replaced with the character S.

D. Stage 4: Commodity Compression

Finally we compress the delta encoded sequences, references and score files and archive the result to produce a single file.

This can be achieved using many popular compression programs. The `bzip2` program was found to be very effective for the delta encoding scheme; furthermore, `bzip2` may be used in conjunction with the `tar` program to produce a single compressed archive for convenience.

V. TESTING METHODOLOGY

We present an implementation of the CANzip algorithm in the C++ programming language. For the remainder of the paper we refer to the program as CANzip.

A program was devised to generate arbitrary input data which mimics the NGS sampling process. The program proceeds as follows.

First a sequence of uniformly distributed random nucleotides is generated and used as the genome. Given a sequence length l, a read length k and an estimation of the coverage c, n samples of length k of the genome are taken at random points. n is defined as $n = \frac{l*c}{k}$. Finally sequencing errors are simulated by inducing a chance that each base may randomly mutate to another.

A uniform random distribution of ASCII characters was used as the model for sequence scores and a uniform random distribution of English characters was used for sequence names and annotations.

Two data sets were generated using this random model of FastQ data as noted in *Table I*.

The data sets were compressed using CANzip and the selected commodity compression tools.

VI. RESULTS

Results follow in *Table II*.

TABLE I
RANDOMLY GENERATED FASTQ DATASETS

Length (bp)	Read Length (bp)	Coverage	% Noise	File Size (bytes)
10 000	75	60	1	1 515 520
100 000	75	60	1	15 158 363

TABLE II
COMPRESSION RESULTS

Program	Original Size (bytes)	Compressed Size (bytes)	Compression Ratio (bytes)
gzip	1 515 520	890 779	0.588
gzip	15 158 363	9 639 808	0.636
bzip2	1 515 520	769 764	0.508
bzip2	15 158 363	8 356 872	0.551
xz	1 515 520	774 336	0.511
xz	15 158 363	7 788 504	0.514
CANzip	1 515 520	794 424	0.524
CANzip	15 158 363	8 007 107	0.528
dsrc	1 515 520	630 163	0.416
dsrc	15 158 363	6 337 919	0.418

The compression ratio in each test was calculated using *Equation 2*.

$$CR = \frac{Compressed\ size}{Original\ size} \tag{2}$$

For the two datasets tested, our algorithm CANzip produced competitive results to the general purpose compression tools gzip, bzip2, xz and dsrc a tool designed specifically for genetic data. Overall, two factors will affect the compression outcome of CANzip, one is the extent of similarities among reads in the FastQ file, which can be greatly influenced by the amount of sequence coverage, and the degree of redundancy in the genetic sequences. The other is the quality of clusters, i.e., bad quality clusters will adversely influence the compression ratio, thus putting a great emphasis on selecting the right clustering algorithms.

VII. DISCUSSION

CANzip achieved good compression ratios; however, the running time for such an elaborate compression tool brings to mind an important question about trade offs. *Is it worth the trade off in time to achieve a better compression ratio?* The greedy clustering algorithm which occupies the majority of CPU time for a run of CANzip is $O(n^2)$. For large data sets the algorithm will be ineffective because quadratic algorithms do not scale.

To verify this understanding an approximate model of the running time of the CANzip algorithm was established. Data sets of sizes which doubled were used as input to CANzip and the running time was measured from start to finish.

Table III indicates the times taken by the program for the various problem sizes.

Using a power law regression the relation in *equation 3* was established. This closely models the results.

$$2.24 \times 10^{-4} n^{1.915} \tag{3}$$

It is not unusual for a real data set to contain more than a million reads. The program may take longer than seven and a half million days for a real data set if the relation holds for

TABLE III
TIMING MEASUREMENTS FOR VARIOUS PROBLEM SIZES

Problem Size (# of reads)	Time (seconds)
200	5.591
400	22.508
800	78.568
1600	301.162
3200	1164.12

large data sets.

VIII. FUTURE WORK

The algorithm could be improved by only generating artificial references when the clusters are large. This would remove much redundancy and might yield better results where clusters are smaller.

As CANzip does not consider the compression of quality scores, the approach adopted by [10] could be adopted to improve the overall compression ratio.

Another potential extension to this work would be to investigate the effectiveness of the CANzip algorithm as a general purpose compression tool. Texts and other media may be decomposed into short sequences for clustering. The algorithm would then follow the same procedure as for genetic data.

There are two challenges in particular which would have to be solved in order for the algorithm to be applied to general purpose compression. The first is that there may not be an easy or efficient way to divide a given source such that the divisions may be clustered and easily reassembled. The second is that there may never exist repeated sequences of source symbols in the same way that we view coverage in sequence analysis.

IX. CONCLUSION

Current trends in the growth of storage capacity will not meet the rate at which genetic sequencing data is being created. We have investigated compression as a means to address this problem. CANzip is a novel compression algorithm which uses delta compression for short read sequencing data for which there is no reference genome. Applied to next generation sequencing data the algorithm could help alleviate the rate at which storage hardware will have to be expanded. However, the greedy clustering algorithm employed by CANzip is too slow for real data sets. We have found that CANzip outperforms gzip in most cases and can outperform bzip2 for a random model of FastQ data.

X. Acknowledgements

We thank Tian Yu and Pawel Weber for their discussion and technical assistance. This work is partially supported by NSF No. OCI-1124123 to L Zhang.

References

[1] D.and Bell T. Adjeroh and Mukherjee A. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*. Springer, 2008. ISBN: 978-0-387-78908-8.

[2] David Arthur, Bodo Manthey, and Heiko Röglin. "k-Means Has Polynomial Smoothed Complexity". In: *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 405–414. ISBN: 978-0-7695-3850-1. DOI: 10.1109/FOCS.2009.14. URL: http://dx.doi.org/10.1109/FOCS.2009.14.

[3] EBI. *An integrated map of genetic variation from 1092 human genomes*. Oct. 2012. URL: http://www.1000genomes.org/announcements/integrated-map-genetic-variation-1092-human-genomes-2012-10-31.

[4] Robert C. Edgar. "MUSCLE: multiple sequence alignment with high accuracy and high throughput". In: *Nucleic Acids Research* 32 (5 2004), pp. 1792–1797.

[5] Hach F. et al. “SCALCE: Boosting Sequence Compression Algorithms Using Locally Consistent Encoding”. In: *Bioinformatics* 28 (23 2012), pp. 3051–3057.

[6] A. Feldspar. *An Explanation of the DEFLATE Algorithm*. Aug. 1997. URL: http://www.gzip.org/deflate.html.

[7] M.H. Fritz et al. “Efficient Storage of High Throughput Dna Sequencing Data Using Reference-Based Compression”. In: *Genome Research* 21 (2011), pp. 734–740.

[8] E. Grassi, F. Di Gregorio, and I. Molineris. “KungFQ :A Simple and Powerful Approach to Compress FastQ Files”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9 (6 2012), pp. 1837–1842.

[9] S Hazelhurst. “Algorithms for Clustering Expressed Sequence Tags: the WCD Tool”. In: *SACJ* (2008).

[10] L. Janin, G. Rosone, and A.J. Cox. “Adaptive Reference-Free Compression of Sequence Quality Scores”. In: *Bioinformatics* Advance Access (2013), pp. 1–8.

[11] D.C. Jones et al. “Compression of Next-Generation Sequencing Reads Aided by Highly Efficient De Novo Assembly”. In: *Nucleic Acids Research* (2012), pp. 1–9.

[12] G. Korodi and I. Tabus. “Compression of Annotated Nucleotide Sequences”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4 (3 2007), pp. 447–457.

[13] I. Pavlov. *7z File Format*. 2013. URL: http://www.7-zip.org/7z.html.

[14] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann series in multimedia information and systems. Morgan Kaufmann, 2012. ISBN: 9780124157965.

[15] J. Sewerd. *bzip2 and libbzip2, version 1.0.3*. Feb. 2005. URL: http://www.bzip.org/1.0.3/html/index.html.

[16] L. Zhang et al. “A Framework For A General Purpose Sequence Compression Pipeline: A Centroid Based Compression”. In: *The Proceedings of the 6th World Congress in Computer Science Computer Engineering and Applied Computing*. 2012.

[17] J. Ziv and A. Lempel. “A Universal Algorithm for Sequential Data Compression”. In: *IEEE Transactions on Information Theory* IT-23 (3 1977), pp. 347–342.