Small and fast data format for genomic numerical signals

Do Huy Hoang[1] *and Sung Wig Kin*[2]

[1]*Genome Institute of Singapore, 60 Biopolis Street, #02-01, Genome, Singapore 138672*

[2]*School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417*

ABSTRACT

Background: Read density data is one of the most popular data types. They can be used to represent the peak intensity in ChIP-seq, the transcript expression in RNA-seq, the copy number variation in whole genome sequencing, etc. Density data is currently represented using the file format BigWig. To save space, this file format is compressed and it allows certain types of queries (minimum, maximum, average, standard derivation) that enable interactive analysis or browsing of density datasets.

UCSC genome browser is heavily using this file format. Currently, out of 5.2TB UCSC hg19 database, 1.8TB (34% of the total space) is used to store BigWig files and BigWig can support about 1.5 thousands queries per second.

Results: Although BigWig is good enough now, as sequencing gets cheaper, both storage space and query time are expected to become limited. This paper proposed a new method to store density data. Our new format uses half the size of existing bigWig files and improves random query speed by 100 times.

1 INTRODUCTION AND BACKGROUNDS

BigWig is the main format in UCSC to store and visualize numerical signals associated with each base in a genome. It is frequently used to store density information for ChIP-seq, RNA-seq, DNase, exome sequencing and whole genome sequencing. BigWig provides four summary operations over some input interval to support fast analysis and visualization in UCSC genome browser. These operations are: mean, min, max, coverage, and standard deviation. The formal definitions are given as follows. (1) Mean: The arithmetic mean of the values, (2) Min/Max: the minimum/maximum values, (3) Coverage: the number of bases contains actual data (4) Standard deviation: the standard deviation of the values.

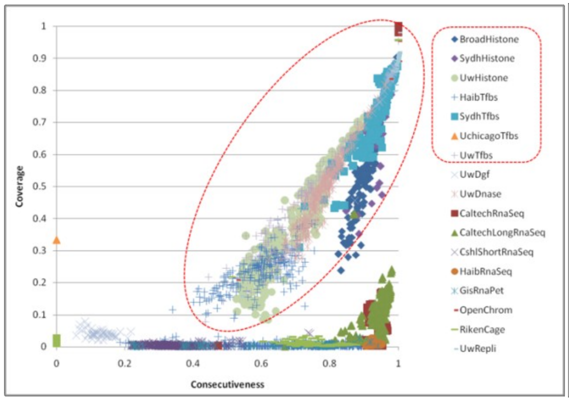
To support fast queries to these 4 operations, BigWig pre-computes the answers in different zoom levels. For example, zoom level 1 stores answers for regions of length 50,000 bases; zoom level 2 stores answers for regions of length 5,000 bases, etc. The indexes of the pre-computed regions are indexed by R-tree.

2 OBERVATIONS

We start with a similar data model as bigWig. Instead of storing signal values of individual bases, we also group bases that have the same values into intervals. The problem becomes storing a set of tuples i.e. $\{(s_i, e_i, v_i)\}$ where s_i and e_i are the positions of the start and end of the intervals in a genome; and v_i is the signal value of the bases in the interval $s_i..e_i$.

Interval positions: For high density regions, NGS reads are often overlapped. Once we pileup the reads and generate the coverage data, each high density region is expected to form a set of consecutive intervals. To precisely measure this characteristic, we define a measurement called consecutiveness, which is the percentage of intervals in a BigWig file that have their start positions equal the end positions of their adjacent intervals. Fig. 2(a) shows the consecutiveness against the coverage of the datasets in UCSC hg19 database. In this figure, most of the Chip-seq data files (highlighted in red area) have high coverage and high consecutiveness. Most RNA-seq files have low coverage. Moreover, many of them have high consecutiveness.

Signal values: We have two observations on signal values. Firstly, among all UCSC bigWig files, the average entropy of the raw signals is about 4.9, while the entropy of the differences is around 3.2. This means that, with a suitable compression scheme, storing the differences uses less space than storing the raw signal values on average. Secondly, we observed that there are only a few frequent signal (or difference) values occur in the bigWig file. To be precise, we define the number of frequent signal (or difference) values in a bigWig file as the minimum number of distinct values whose sum of occurrences makes up 75% of the total number of values in that file. Fig. 2 (b) shows the number of bigWig files that have x frequent signal (or difference) values for all x. Out of 4400 bigWig files in UCSC hg19, about 1500 files have less than 6 frequent raw signal values. About 2500 files have less than 6 frequent differences values. Most of the files have less than 60 differences values.



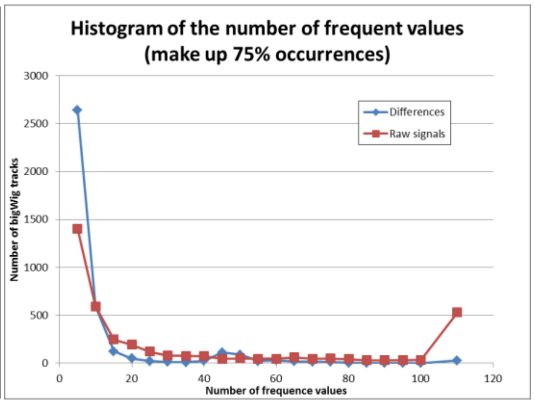
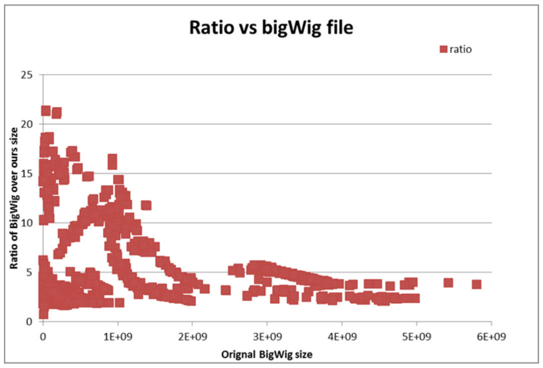


Fig. 1: (a) coverage vs. consecutiveness in UCSC hg19 bigWig files. Red oval highlights most Chip-seq datasets (b) Histogram of frequent values

3 METHODS

Based on the observations in Section 3.2, we design our scheme for storing values, and the auxiliary data structure to support the required query. For the storage scheme, there are two main stages. The first stage converts the signals into integers and decides if we store the raw signal values or the differences based on the entropy. It also applies some common transformations to make numbers easier to compress. The second stage uses a mixtures of methods to compress the integers. Based on the observations , we uses: Huffman code and Elias delta code. Delta code is good when the numbers are uniformly random in a very large range. Huffman code is better when there are few symbols unequal probabilities. We use Huffman code to encode the 127 most frequent numbers. Delta code to encode the rest of the numbers. For the auxiliary data structures, we use succinct and compression data structures to support all the queries in bigWig.

4 EXPERIMENT RESULTS



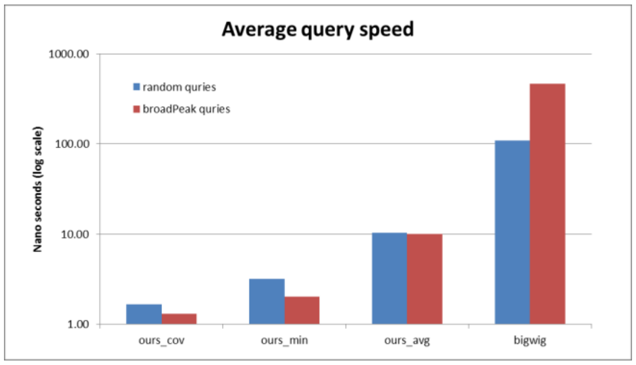


Fig. 2: (a) Ratio between BigWig file sizes and our file sizes. (b) Average query speed for random generated queries and a broakpeak file.

We uses two datasets in this section. The first dataset is all hg19 bigWig files in UCSC database. The second dataset contains a small subset of UCSC hg19 database that are grouped by value types (i.e. integer signal vs. real signal), and by data types (i.e. ChIP-seq, RNA-seq, and other).

Fig. 2(a) plots the original bigWig size versus the reduction that we can achieve. It shows that our file are about 3 times smaller than bigWig files. Fig 2(b) shows that the query speed of our program is about 10 to 500 times faster than bigWig depending on query type. In our program, coverage queries are much faster than minimum and average queries, since it only uses the interval position component. The minimum queries are faster than average queries in sparse files where there are a lot of regions without values.