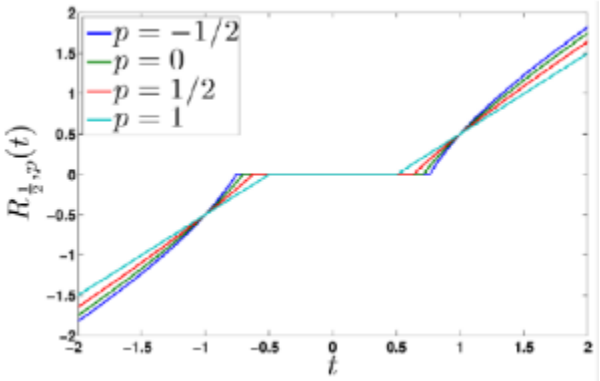
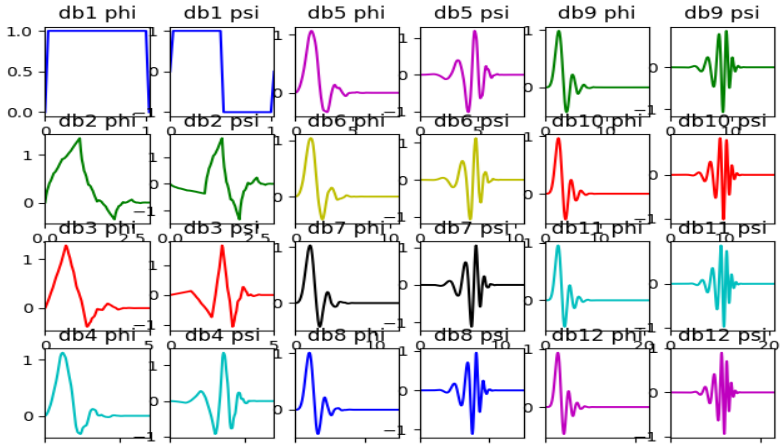
Lossy compression

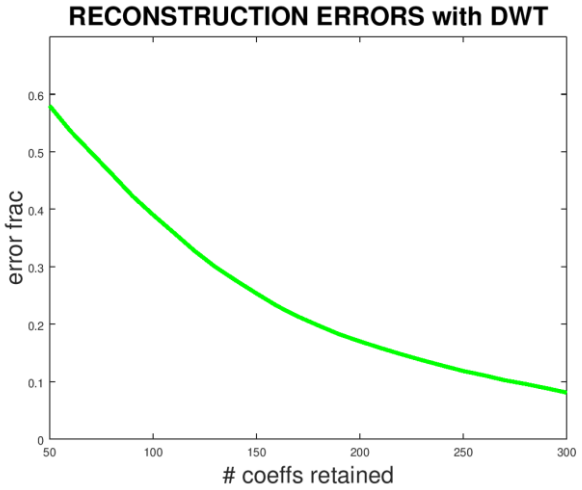
Based on transform / thresholding schemes for small coefficients. Can use e.g. CDF 9-7 wavelets and firm thresholding. Idea based on relation: $s \approx W_i^{\{-1\}}\left(Thr(W_i s)\right)$

```
function w2=pThreshold(w, lambda, p)
    w2 = w;
    for i=1:length(w)
        w2(i) = sign(w(i))*max(0,abs(w(i)) - lambda*abs(w(i))^(p-1));
    end
end
```



Easy adaptivity for different blocks of data based on the above relation.

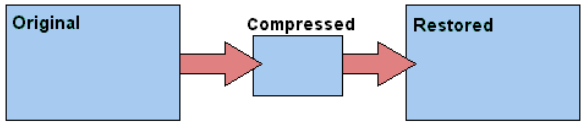




Sergey Voronin, 8.2019

Lossless compression

Lossless compression based on reducing alphabet size and encoding frequently occurring symbols with fewer bits. Needed for e.g. text/numerical data, when loss of information is not acceptable.



Entropy of a set of elements $e_1,\ldots,e_n$ with probabilities $p_1, \ldots p_n$ is:

$$H(p_1 \ldots p_n) \equiv -\sum_{\forall i} p_i \log_2 p_i$$

Critically, one must seek to reduce data entropy to improve compression performance.

$$Max \ \ when \ \ p_1 = \cdots = p_n = \frac{1}{n} \rightarrow H = \log_2(n)$$

Example 1: A p_A=0.5, B p_B=0.5

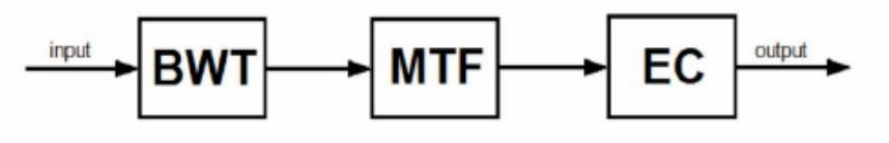
$$H(A,B) = -p_A \log_2 p_A - p_B \log_2 p_B = \\ = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

Example 2: A p_A=0.8, B p_B=0.2

$$H(A,B) = -p_A \log_2 p_A - p_B \log_2 p_B = \\ = -0.8 \log_2 0.8 - 0.2 \log_2 0.2 \cong 0.7219$$

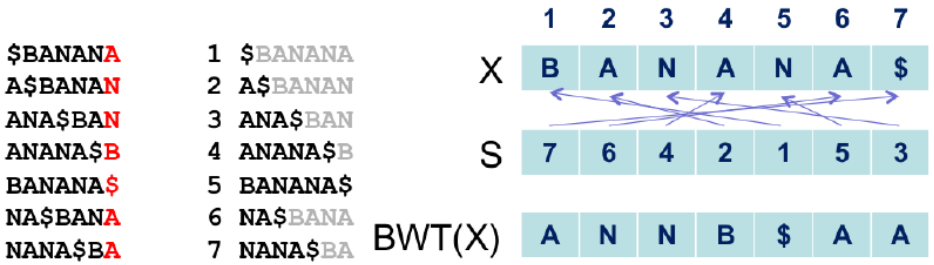
Burrows-Wheeler transform based

One of the best general purpose compressors (bzip2, lbzip2).




```
(a) BWT input    : 61 62 72 61 63 61 64 61 62 72 61 61 62 72 61 63 61 64 61 62 72 61
(b) BWT output   : 61 72 72 64 64 61 72 72 63 63 61 61 61 61 61 61 61 61 62 62 62 62
(c) GST output   : 61 72 00 65 00 02 02 00 65 00 02 00 00 00 00 00 00 00 65 00 00 00
(d) RLE0 output  : 62 73 00 66 00 03 03 00 66 00 03 00 00 00 66 00 00
(e) EC output    : 00 0D 01 8D B3 FF 81 00 72 A8 E8 2B
```

```
$BANANA    1 $BANANA
A$BANAN    2 A$BANAN
ANA$BAN    3 ANA$BAN
ANANA$B    4 ANANA$B
BANANA$    5 BANANA$
NA$BANA    6 NA$BANA
NANA$BA    7 NANA$BA
```

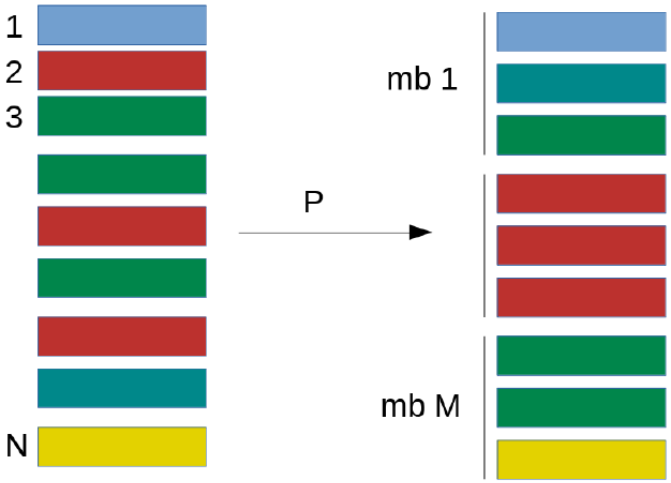


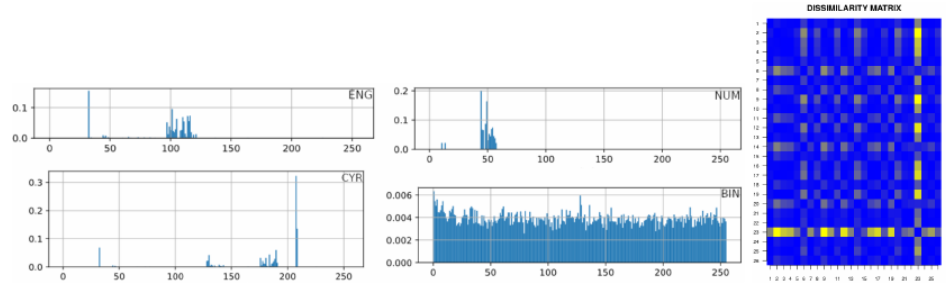
BWT rearranges input to reveal patterns, MTF/RLE move common symbols to front, compress sequences of identical digits, EC (Huffman or Arithmetic coding), encodes remaining data in fewer bits.

BWT needs to be performed over small chunks due to expensive string sorting.

Parallel lossless compression

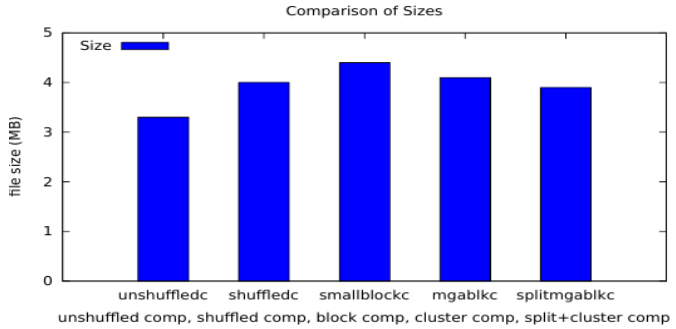
Easy parallel compression: subdivide input in small chunks, use sub-chunks for BWT. Disadvantages: doesn't take advantages of patterns throughout the whole file. Use of small blocks hurts compression ratio. Instead, we like to cluster together similar small blocks.





Approximate symbol probability distributions constructed for small blocks. Relevant metrics e.g. KLM, used for construction (di)-similarity matrix based on distances, in turn utilized by different clustering algorithms to make larger megablocks which are parallel compressed.

Special inputs (text / numerical) can be bundled separately and e.g. PPM employed on text.



Faster string sorting

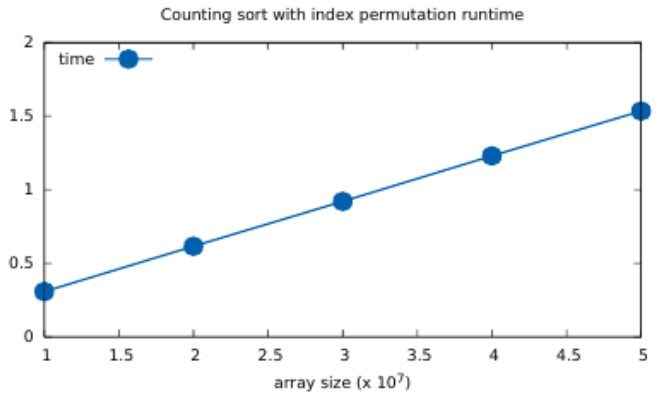
Advantageous to perform BWT on larger blocks. For this to be efficient, several levels of O(n) pre-sorting can be done to create several smaller buckets to sort with O(n log n) sorts.

```
bananasale          ale          nbucket = 1, cur =  (10)
ananasale           ananasale    nbucket = 2, cur = ananasale (97)
nanasale            anasale      nbucket = 2, cur = anasale (97)
anasale             asale        nbucket = 2, cur = asale (97)
nasale        ──►   bananasale   nbucket = 2, cur = ale (97)
asale               e            nbucket = 3, cur = bananasale (98)
sale                le           nbucket = 4, cur = e (101)
ale                 nanasale     nbucket = 5, cur = le (108)
le                  nasale       nbucket = 6, cur = nanasale (110)
e                   sale         nbucket = 6, cur = nasale (110)
                                 nbucket = 7, cur = sale (115)
```

```
struct val_inds
{
    int val;
    int num_inds;
    int *inds;
};
```

```
before sort:
5  4  2  1  1  3  4  12  10
after sort:
1  1  2  3  4  4  5  10  12
inds after sort:
3  4  2  5  1  6  0  8  7
```

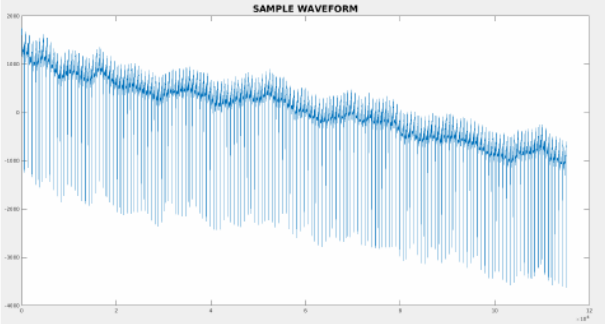
```
counting_sort_with_permind(arr,inds,n,high+1);
```

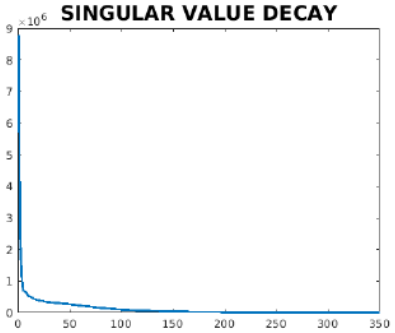


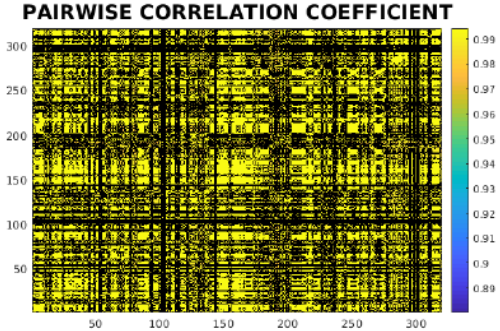
Counting sort with permutation record developed. Enables easy (one level) bucketing.

Similarity compression

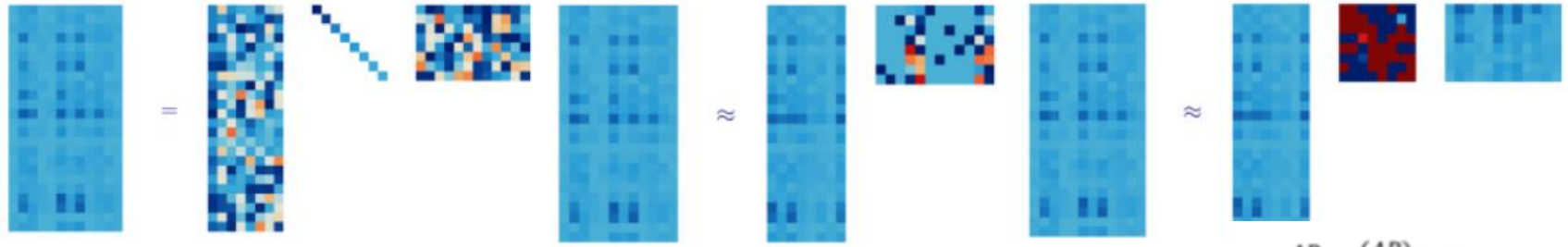
Multi-channel data often has similarity which can be exploited via approximate linear dependence and linear correlation (under transformation and sorting).







Singular value decay will most likely be non-linear. In this case, suitable low rank factorizations (interpolative decomposition and CUR can be exploited to return the ‘key subset’ of waveform data).



SVD returns eigenvectors which can be difficult to interpret. ID returns a subset of columns or rows of the original matrix, which allows for a reduction in the number of channels.