

**A solution for context based blocksort compression.
The M03 algorithm.**

By Michael A. Maniscalco

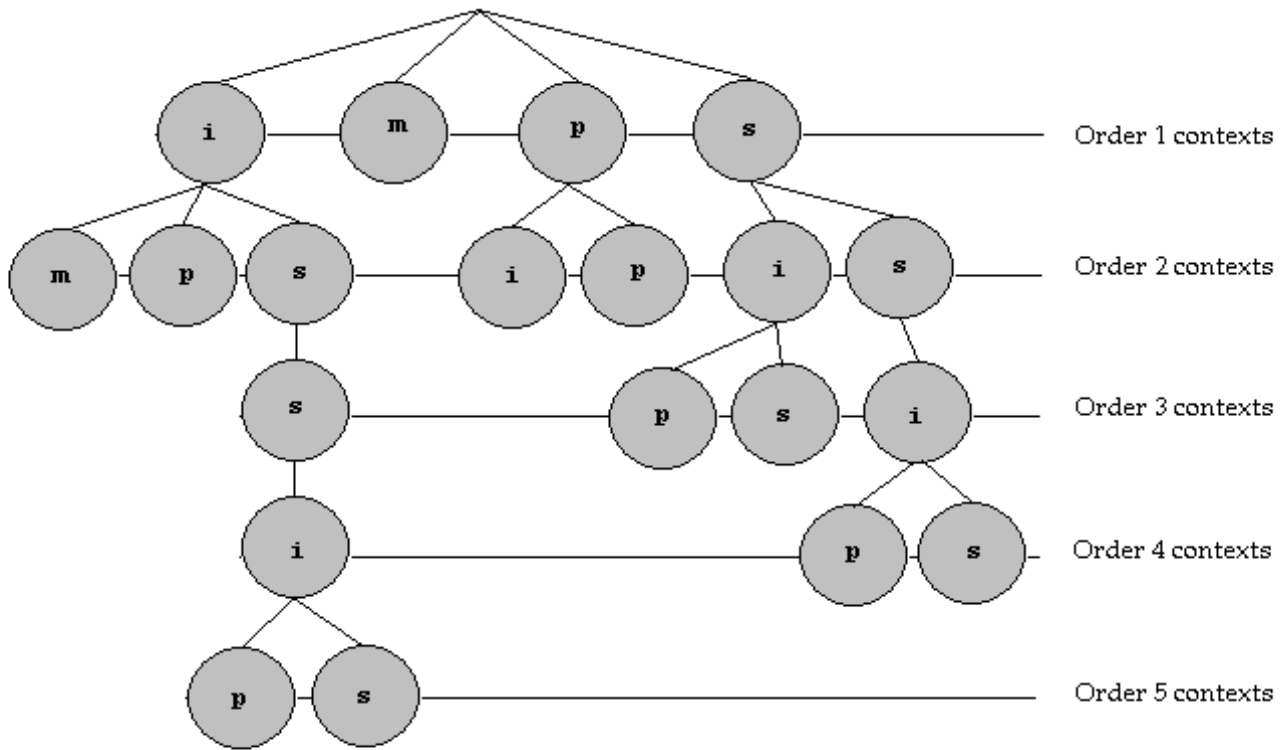
The following document details a novel method for encoding and decoding Blocksor transformed data with respect to all context switches in the source signal which generated the blocksort transformed data.

Note from author:

This is a rough draft paper. The actual information for encoding and decoding is not currently detailed in this paper. However, the method for encoding and decoding with respect to all context switches is completely disclosed.

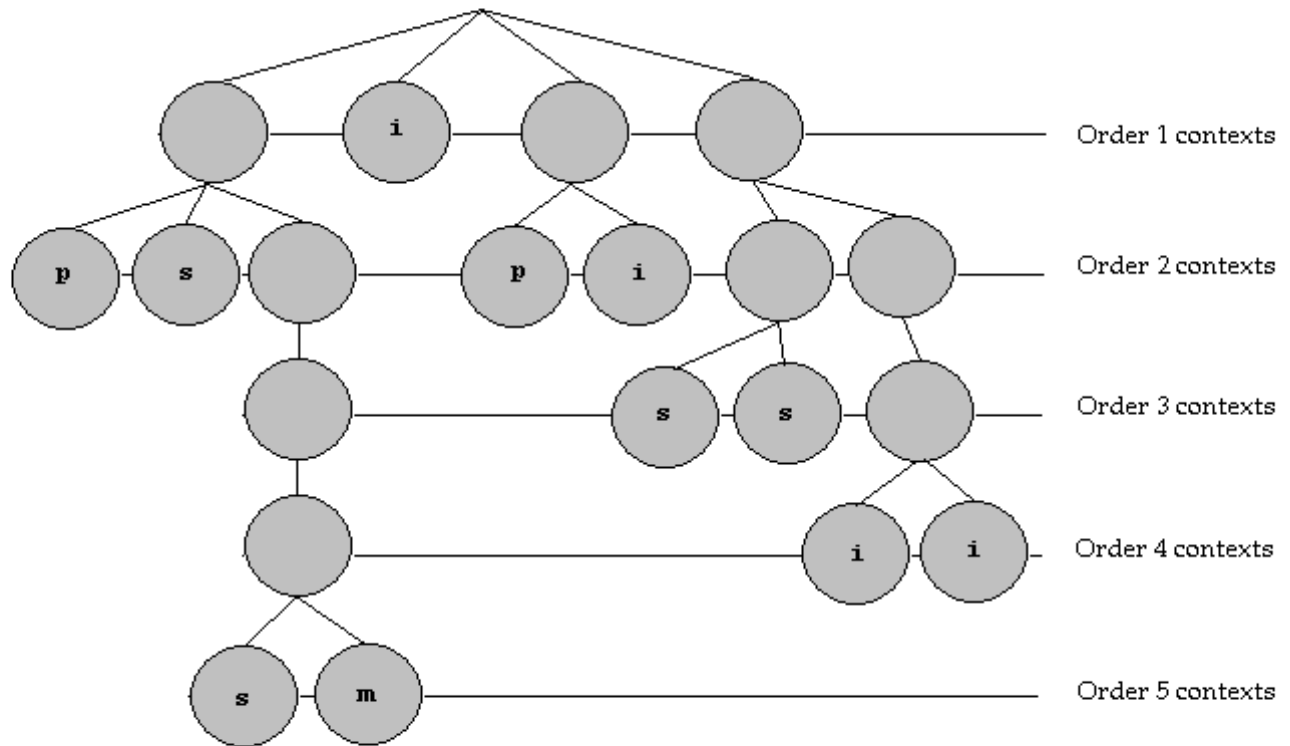
- Michael

The unique contexts of 'mississippi'



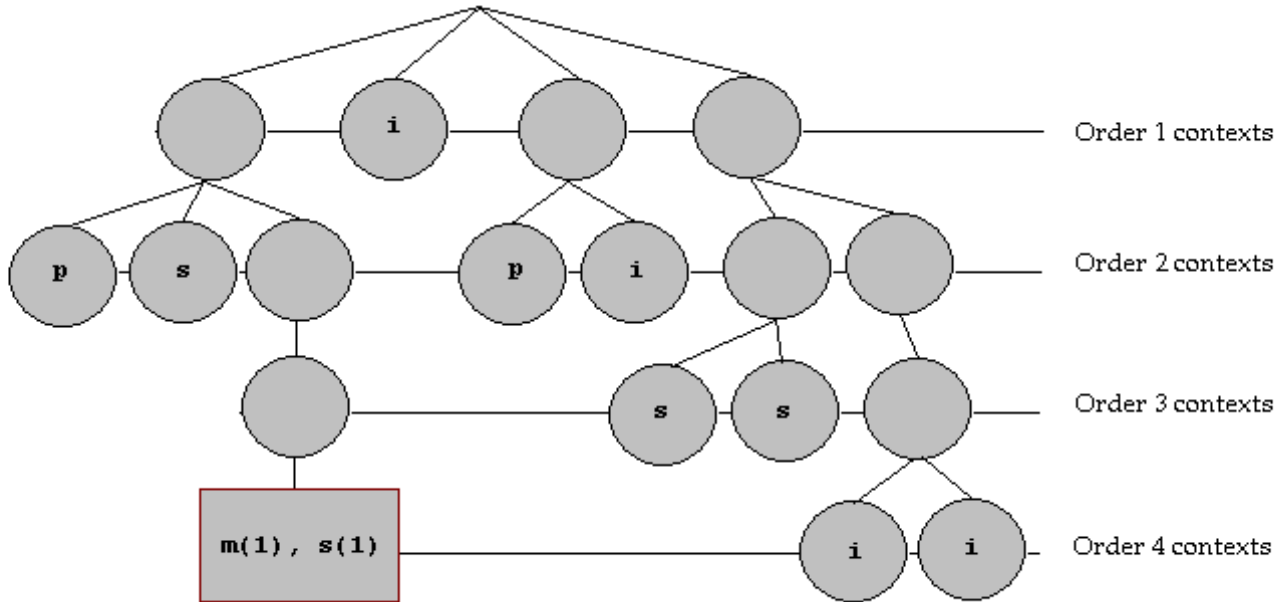
Begin by mapping the unique contexts of the source string. In this case the source is 'mississippi'. This is done with a suffix sort. We are only interested in each context up to the order at which it does become unique in the source. This means that there will be exactly one leaf node in the tree for each symbol in the source string with the tree mapping each unique context in sorted order, from left to right.

The BWT of 'mississippi' (pssmipissii) mapped onto the unique contexts of 'mississippi'.



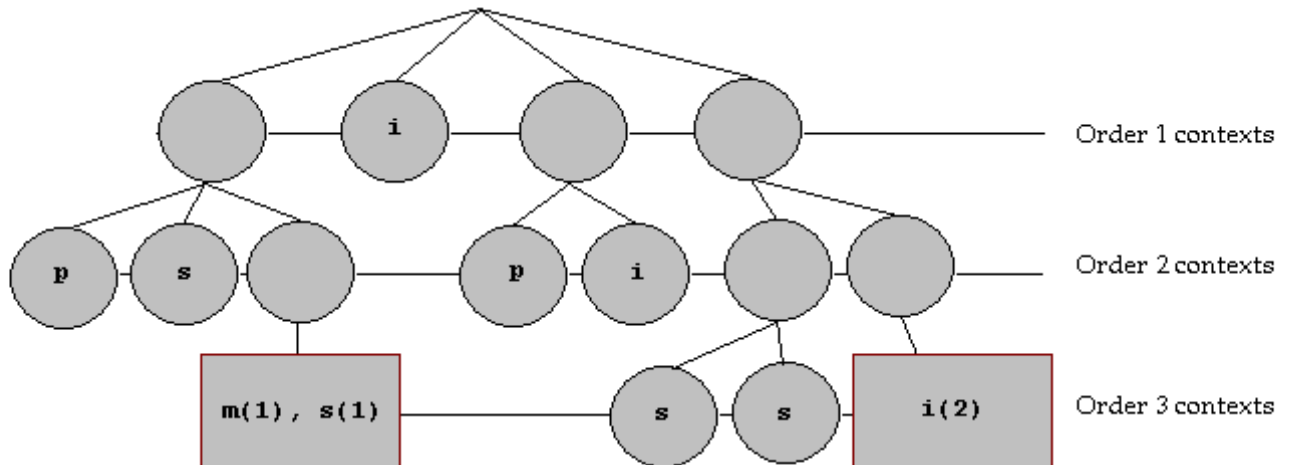
Now 'map' the BWT of the source onto the leaf node for each context in the original source. This, in essence, maps the symbol which preceded the given context onto the leaf node of that context.

Encoding step 1: Merge order 5 contexts and promote to order 4 context.



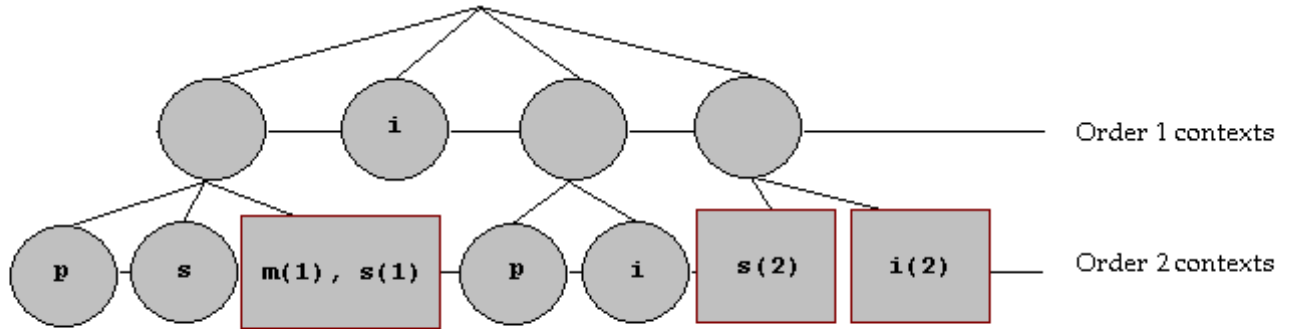
Now, beginning at the highest order, merge contexts which share the same context at the next lowest order. In this case, we merge order 5 contexts into order 4 contexts. We then promote this context to order 4.

Encoding step 2: Merge order 4 contexts and promote to order 3 contexts.



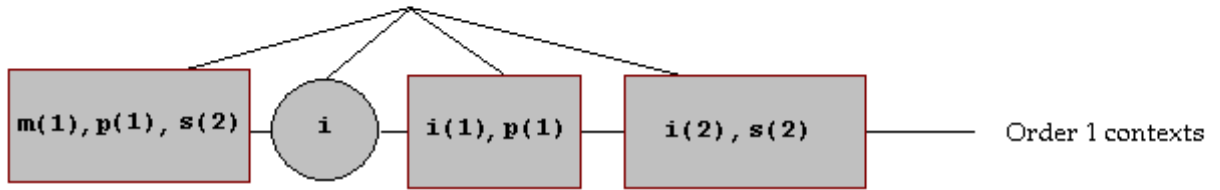
Now we continue with the encoding process by merging all order 4 contexts with same parent nodes (same parent context) together, then promoting them to order 3 contexts.

Encoding step 3: Merge order 3 contexts and promote to order 2 contexts.



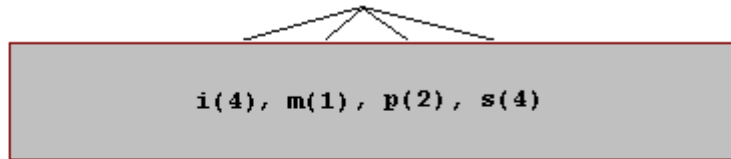
Now we continue with the encoding process by merging all order 3 contexts with same parent nodes (same parent context) together, then promoting them to order 2 contexts.

Encoding step 4: Merge order 2 contexts and promote to order 1 contexts.



Now we continue with the encoding process by merging all order 2 contexts with same parent nodes (same parent context) together, then promoting them to order 1 contexts.

Final encoding step: Merge all order 1 contexts into the order zero context.



Finally, the last encoding merge is done. This produces one context which contains all of the symbols from the original source. All that remains of the encoding process is to encode the count for each symbol in the context.

This final state (above) is the starting point for the decoding process. This single context will be 'split' in such a way as to reverse the encoding merge which took place in the final encoding step. After this split is complete, a set of order 1 contexts is produced. And this new set of order 1 contexts will provide us with the needed information to understand where the order 2 context boundaries were located in the original source.

For all decoding steps, the context boundaries for the next highest context are found using the current set of contexts, as follows.

Examine all counts of each symbol type for all current order contexts. In our current state (the first decoding state) there is only one context. It contains (in order) four 'i' symbols, one 'm' symbol, two 'p' symbols and finally, four 's' symbols. From this we can know that the next highest context boundaries will be drawn at four symbols, followed by one symbol, followed by two symbols, and lastly four symbols. Or, more accurately, this single order zero context was the product of merging 4 order one contexts where the first context contained four symbols, the second contained one symbol, the third contained two symbols and the last contained four symbols. And the symbols in these contexts are made up of the symbols in the current (order zero) context. So decoding the merge splits the order zero context into the following order one contexts.

[m(1), p(1), s(2)] + [i(1)] + [i(1), p(1)] + [i(2), s(2)]

These are the order one contexts from the original source.

We now have the order one contexts:

$$[\mathbf{m(1), p(1), s(2)}] + [\mathbf{i(1)}] + [\mathbf{i(1), p(1)}] + [\mathbf{i(2), s(2)}]$$

This is enough information to determine the where the order two context boundaries were. Again, by noting the count of each symbol type, in order we see that the symbol 'i' occurs in groupings of 1, 1, and 2. This (not by coincidence) totals the number of symbols in the first order one context. And it indicates that the first context ... $[\mathbf{m(1), p(1), s(2)}]$ was a result of merging three higher order contexts of length 1, 1, and 2. Continuing in sorted order, the symbol 'm' occurred in the grouping of 1. The symbol 'p' occurred in groupings of 1 and 1. And the symbol 's' occurred in groupings of 2 and 2.

Thus the order one context ...

$$[\mathbf{m(1), p(1), s(2)}] + [\mathbf{i(1)}] + [\mathbf{i(1), p(1)}] + [\mathbf{i(2), s(2)}]$$

should be split into contexts as follows:

$$[1] + [1] + [2] + [1] + [1] + [1] + [2, 2]$$

and when we decode the merging process for that step we get the order two contexts ...

$$[\mathbf{p(1)}] + [\mathbf{s(1)}] + [\mathbf{m(1), s(1)}] + [\mathbf{i(1)}] + [\mathbf{p(1)}] + [\mathbf{i(1)}] + [\mathbf{s(2)}] + [\mathbf{i(2)}]$$

We now have the order two contexts:

$[p(1)] + [s(1)] + [m(1), s(1)] + [i(1)] + [p(1)] + [i(1)] + [s(2)] + [i(2)]$

And, as before, these contexts give us the information needed to split these order two contexts into the original order three contexts by using the counts of each grouping of each symbol type in order. They are:

Symbol `i' -> 1, 1, 2

Symbol `m' -> 1

Symbol `p' -> 1, 1

Symbol `s' -> 1, 1, 2

Thus the context boundaries for the order three contexts are ...

$[1] + [1] + [2] + [1] + [1] + [1] + [1] + [1] + [2]$

and when we use the information encoded for the merge at this step we get the contexts ...

$[p(1)] + [s(1)] + [m(1), s(1)] + [i(1)] + [p(1)] + [i(1)] + [s(1)] + [s(1)] + [i(2)]$

this is the order three contexts.

We now have the order three context ...

[p(1)] + [s(1)] + [m(1), s(1)] + [i(1)] + [p(1)] + [i(1)] + [s(1)] +
[s(1)] + [i(2)]

and as with every step, this provides us with information about the boundaries of the order four contexts by using the count of each grouping of each symbol in sorted order. They are ...

symbol 'i' -> 1, 1, 2

symbol 'm' -> 1

symbol 'p' -> 1, 1

symbol 's' -> 1, 1, 1, 1

and so the context boundaries for order four are ...

[1] + [1] + [2] + [1] + [1] + [1] + [1] + [1] + [1] + [1]

and using the information encoded during the merging of these contexts we get the order four contexts ...

[p(1)] + [s(1)] + [m(1), s(1)] + [i(1)] + [p(1)] + [i(1)] + [s(1)] +
[s(1)] + [i(1)] + [i(1)]

This is the order four contexts ...

We now have the order four contexts ...

$[p(1)] + [s(1)] + [m(1), s(1)] + [i(1)] + [p(1)] + [i(1)] + [s(1)] + [s(1)] + [i(1)] + [i(1)]$

and as with every step, this provides us with information about the boundaries of the order five contexts by using the count of each grouping of each symbol in sorted order. They are ...

symbol 'i' -> 1, 1, 1, 1

symbol 'm' -> 1

symbol 'p' -> 1, 1

symbol 's' -> 1, 1, 1, 1

and so the context boundaries for order four are ...

$[1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1] + [1]$

and using the information encoded during the merging of these contexts we get the order five contexts ...

$[p(1)] + [s(1)] + [s(1)] + [m(1)] + [i(1)] + [p(1)] + [i(1)] + [s(1)] + [s(1)] + [i(1)] + [i(1)]$

And, since every context now contains only one symbol we have finished the decoding process.

The original BWT string was ...

pssmipissii

Decoding completed.