Some recent Burrows Wheeler ideas.

Some of my recent ideas on Burrows Wheeler compression were presented at a workshop at Rutgers University in August 2004, celebrating the 10th anniversary of the Burrows Wheeler Transform.

This material has now been published in
"**Burrows–Wheeler compression : Principles and reflections"** *Theoretical Computer Science* Vol 387(2007) pp 200–219 (see my publications list for an abstract).

The ideas fall into three groups, in order of decreasing work-done.

1. Use a BW transform to derive *all* contexts of some fixed order, say 4, and use these to drive an escape-free PPM compressor/decompressor. This is the main substance of the report and presentation; it worked, but not nearly as well as hoped.
2. "Break open" the reverse BW transform to generate partial contexts during both compression and decompression, so aiding the operation of the coder and decoder.
   The report starts by developing analogies between the Fourier Transform and its associated spectra, autocorrelations, etc, on the one hand and the the Burrows Wheeler transform and PPM on the other. Invoking ideas of symmetry leads to a significant new understanding of the Burrows Wheeler operation.
   (I have long thought that we did not really understand the Burrows-Wheeler Transform. But the forward transform is essentially simple, at least in concept, and showed little scope for improvement. With these ideas the reverse transform looks the more likely for further study and deeper understanding. At least I hope so.)
3. Is there some relation between data compression and error correction codes? Compression can be thought of as a very noisy channel (encoding a single symbol into a range of possible symbols) and then generation of the codes to select the correct symbol from the "noise" codes; the encoder and decoder track each other and the "corrections" are what the compressor emits. (The ideas follow most readily from my earlier work on "Symbol Ranking" compression which uses an explicit symbol predictor/corrector mechanism).
   This third idea may provide a useful starting point for somebody with more imagination than me. It leads to two other ideas relating to the possible relationship between Error Correction and compression.
4. Can we omit some symbols (giving an erasure code) and then use a Viterbi-like trellis to recover feasible decodings. (Whew!)
5. The relatively new error-correcting "turbo-codes" use two interacting but complementary error-correcting codes to give superlative performance. Can something similar be done for compression? (It also ties in with the previous point.)
   But do we have suitably different compressors? The two compressors must work with the same symbol order, probably precluding a BW and PPM combination. But of those which retain the original "text" order, PPM and LZ-77 were shown to be equivalent by Bell, Cleary & Witten (in their book "Text Compression"). So until somebody invents say a symbol-by-symbol compressor which does not depend on contexts, there may be no suitable complementary candidates!
   If you have ideas, please try them!

Success!

The vague ideas above eventually (eventually) led to a PPM compressor without escapes. The unsuccessful ideas included 2-pass compressors which found all contexts and their contents on the first pass, and then transmitted this information as needed on a second, compression, pass. The cure was worse than the disease.

The version that finally succeeded used binary contexts and a bit-wise compressor. The contexts in normal PPM (more correctly the *contents of the contexts*) have three possible conditions – empty, or unused, partly full, and fully populated. It is last two that require escapes, to add hitherto absent symbols; I had hoped to eliminate the need for escapes in fully-populated contexts. On the other hand, a binary context has only *two* states, absent and populated, with no need for escapes from within a context.

The final compressor uses byte contexts (as usual for PPM), but emits each byte as a sequence of bits, with bit-wise contexts for each symbol within each PPM order. If a particular context is new, the coder and decoder can detect the absent context and silently (without escapes) descend to a known bit-wise context. After processing the bit at this order, coder and decoder ascend up to higher orders creating new binary contexts as needed.

The final compressor gives quite acceptable compression, by amounts between GZIP and BZIP2.