Prediction by Partial String Matching (PPM)

- Apply higher-order context modeling to adaptive arithmetic coding.
- Related to BWT, but runs on-line and with prefix contexts.
Context Modeling

- Track higher-order contexts.
- Use tries containing symbol code and frequency count.
Encoding

• Track probabilities of next character, according to accumulated model.

• When encountering said context, use probabilities to code for current symbol.

• Use escape symbol when encountering new symbol.

• If new symbol, use order -1 and code specially.
Encoding Example (PPMA)

• Already encoded “brata”. Yet to encode “tb”.

• New symbol “t”; never occurred in context “ta” (zero-probability).

• Escape to shorter context with escape symbol: “<escape>”

• New context: “a”, which has occurred once before, with “t” following. Allowing for escape symbol with frequency 1, encode with 1/2: “<escape> <1/2>”.
Encoding Example (cont.)

- Encoded “bratat” so far, with post-”brata” being “<escape> <1/2>”. “b” remaining.

- Context is “at”, encoding “b”. Occurred once before, with “a” following then. Assign freqs 1 to escape, “a”.

- “b” not seen in context before. Escape, then try from context “t”: “<escape> <1/2> <escape>”.

- Nothing in context (seen once, “a” following): “<escape> <1/2> <escape> <1/2> <escape> <escape> <3/7>”; 6 symbols seen+escape: [a,a,b, ...].
Decoding

• Has built up same set of contexts as encoder.
• Thus, can provide arithmetic decoder correct set of probabilities upon which to decode incoming symbol.
• Relies on escape symbol to know when to switch to shorter context.
• Update model, shift context, iterate.
Decoding Example

• Use already encoded example: “<escape> <1/2> <escape> <escape> <escape> <3/7>”, after having built up context from “brata”.

• Start assuming order-2 context “ta”.

• Each “<escape>” reduces order of context.

• See first “<escape>” in stream, shift to order-1 context “a”.

• “<1/2>” indicates only non-escape symbol already known in context, “t”. Decoded up to “bratat”.
Decoding Example (cont.)

• Had decoded up to “bratat”, with “<escape> <escape> <3/7>” remaining, and back in an order-2 context, “at”.

• Read “<escape>” twice, up to order 0, with <3/7> remaining. From order-0 statistics, that's b.

• Throughout: “<escape>” is encoded as just another symbol, each time. This is why, for example, <1/2> with only one previous occurrence of context. In PPMA, it's a singleton. PPMC often multiply.
Variants

- PPMA, PPMB, PPMC, PPMP, PPMX, PPM*, PPMZ, Fast PPM
- Fast PPM: pseudo-arithmetic coding for speed.
- PPM*, PPMZ: unbounded context (PPMZ with local context order estimation).
- PPMA gives escape symbol singleton count.
- PPMC computes escape symbol frequency from number of distinct following symbols.
- PPMB like PPMC, but subtract 1 from all frequencies.
Performance

- From maximumcompression.com and uclc.info.
- Of the most effectively-compressing programs listed, most, such as WinRK, Durilca, Slim, PPMMonstr, UHARC, and Compressia, use PPM. It works well.
- So, more popularly, does WinRAR. It can work fast.
Empirical results shown suggest PPM might serve as a partial stand-in for Shannon's English experiments.

To compare cross-linguistic entropy, need translations of single document. Bible and UN official documents used here.

Acquire or create translations, and run PPM on them; compare compressed sizes.
Languages

- English, Spanish, French, Chinese, Korean, Arabic, Japanese, and Russian
Translations

- UN documents already available translated into official UN languages, including those listed.
- Bibles available too – but not used as such. Instead, machine translated with Systran.
Methodology and Assumptions

- Natural languages have similar expressive power, thus information content of semantically similar texts should be similar.
- Otherwise, either languages aren't similarly expressive or PPM is language-specific.
Results

• Assumptions hold.

• English, Spanish, French, and Russian all come out from 0.991 to 1 times the size of the compressed English text.

• Korean close: 0.970.

• Chinese, Arabic outliers at 0.941 and 1.09.

• Japanese even farther, at 1.209, but odd translation (annotated), throwing measurement off.
Problems

- Bibles were machine-translated. Poor translations could account for differences.
  - Turn this around. Detect poor translations.
- Different filtering processes (English -> [A-Z ], for example) differently effective.
- PPM, BWT implementations used not optimal. Show upper bound of entropy only.
- Textual styles differ. Bible can be unusual (consider KJV's language, intentionally archaic dialect even when written in 1611).