Figure 8.6 might be regarded as being emitted from two different states – one giving rise to the string "rvrvvvvvvvr", and then a second generating "dtdtt". (It might equally be possible that no such distinction would be made, since the string of characters prior to the section in the figure is "vdtrvvdrrvvtdr", and the string following is "vtvdtvrrdvvrrddvdrr".) Each of the sections then becomes a miniature coding problem – exactly as it is in each context of a PPM compression system.

The best of the BWT-based compression mechanisms are very good indeed. Michael Schindler's [1997] SZIP program compresses the file `bible.txt` to 1.53 bits per character (see the results listed at `corpus.canterbury.ac.nz/results/large.html`), and the public-domain BZIP2 program of Julian Seward and Jean-loup Gailly [Seward and Gailly, 1999] is not far behind, with a compression rate 1.67 bits per character. (At time of writing, the best compression rate we know of for `bible.txt` is 1.47 bits per character, obtained by Charles Bloom's PPMZ program, and reported to the authors by Jorma Tarhio and Hannu Peltola.) Note that the compression effectiveness of BWT systems is influenced by the choice of block size used going into the BWT; in the case of SZIP, the block size is 4.3 MB, whereas BZIP2 uses a smaller 900 kB block size. One common theme amongst these "improved" BWT implementations is the use of ranking heuristics that differ slightly from simple MTF. Chapin [2000] analyses a number of different ranking mechanisms for BWT compression, concluding that the most effective compression is achieved by employing a mechanism that switches between two different rankers, depending on the nature of the text being compressed [Volf and Willems, 1998]. Deorowicz [2000] has also investigated BWT variants.

The implementation of BZIP2 draws together many of the themes that have been discussed in this section and in this book. After the BWT and MTF transformations the ranks are segmented into "blocklets" of 50 values. Each blocklet is then entropy coded using one of six semi-static minimum-redundancy codes derived for that block of (typically) 900 kB. The six semi-static codes are transmitted at the beginning of the block, and are chosen by an iterative clustering process that seeds six different probability distributions; evaluates codes for those six distributions; assigns each blocklet to the code that minimizes the cost of transmitting that blocklet; and then reevaluates the six distributions based upon the ranks in the blocklets assigned to that distribution. This process typically converges to a fixed point within a small number of iterations; in the coded message each blocklet is then preceded by a selector code that indicates which of the six minimum-redundancy codes is being used. The use of multiple probability distributions and thus codes allows sensitivity to the nature of the localized probability distribution within each segment; while the use of semi-static minimum-redundancy coding allows for quick encoding and