COMPRESSION AND CODING ALGORITHMS

by

Alistair Moffat
The University of Melbourne, Australia

and

Andrew Turpin
Curtin University of Technology, Australia

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

Preface

None of us is comfortable with paying more for a service than the minimum we believe it should cost. It seems wantonly wasteful, for example, to pay $\$5$ for a loaf of bread that we know should only cost $\$2$, or $\$10,000$ more than the sticker price of a car. And the same is true for communications costs – which of us has not received our monthly phone bill and gone "ouch"? Common to these cases is that we are not especially interested in reducing the amount of product or service that we receive. We do want to purchase the loaf of bread or the car, not half a loaf or a motorbike; and we want to make the phone calls recorded on our bill. But we also want to pay as little as possible for the desired level of service, to somehow get the maximal "bang for our buck".

That is what this book is about – figuring out how to minimize the "buck" cost of obtaining a certain amount of "bang". The "bang" we are talking about is the transmission of messages, just as in the case of a phone bill; and the "buck" we seek to minimize is the dollar cost of sending that information. This is the process of *data compression*; of seeking the most economical representation possible for a source message. The only simplification we make when discussing compression methods is to suppose that bytes of storage or communications capacity and bucks of money are related, and that if we can reduce the number of bytes of data transmitted, then the number of bucks spent will be similarly minimal.

Data compression has emerged as an important enabling technology in a wide variety of communications and storage applications, ranging from "disk doubling" operating systems that provide extra storage space; to the facsimile standards that facilitate the flow of business information; and to the high-definition video and audio standards that allow maximal use to be made of scarce satellite transmission bandwidth. Much has been written about data compression – indeed, we can immediately recommend two excellent books, only one of which involves either of us as an author [Bell et al., 1990, Witten et al., 1999] – and as a research area data compression is relatively mature.

As a consequence of that maturity, it is now widely agreed that compression arises from the conjunction of two quite distinct activities, *modeling* and

coding (and, as we shall see, the interfacing activity of *probability estimation* or *statistics maintenance*). The modeling task splits the data into symbols, and attempts to infer a set of probability distributions that predict how the message to be compressed is made up of those symbols. The objective is to predict the message with 100% accuracy, as all that remains to be transmitted is the difference between the model – which is a kind of theory – and the message in question. Hence, if an appropriate model can be determined for the data being represented, good compression will result, as the residual difference between model and message will be small. For example, in English text a letter "q" is usually followed by the letter "u", so a good model will somehow learn that relationship and use it to refine its probability distributions. Modeling is the public face of data compression, as it is where the creativity and intuition are generally agreed to lie. If a data compression system were an ocean liner, the model would correspond to the bridge – good all-round views, and control over course and speed.

On the other hand, coding represents the engine room of a compression system, and like its namesake on an ocean liner, requires a certain amount of sweat, steam, and grease in order to operate. The coder generates the sequence of bits that represent the symbols and probabilities asserted by the model, and then, at the decoding end of the pipeline, reproduces a stream of instructions that tells the model what symbols should be emitted in order to recreate the original source message. Coding requires – at face value at least – that a relatively straightforward task be carried out, with little more to be done than a mechanical translation process from probabilities to bits. Indeed, coding seemingly has little scope for creative and innovative mechanisms, and tends to be buried deep in the bowels of a compression system, just as is the engine room of an ocean liner.

Because of this relatively unglamorous role, little attention has been focussed on the task of coding. The two books mentioned above both devote more space to modeling than they do to coding, and the same is true of the work of other authors. Indeed, there is a widely held belief that the coding problem is completely solved, and that off-the-shelf packages are available that obviate any need for the compression system developer (or researcher, or student, or interested computing professional) to know much about coding.

Nothing could be further from the truth. As an area of study, source coding has a surprising depth and richness. Indeed, in some ways the intellectual endeavor that has been invested in this area perhaps rivals the energy that has been put into another fundamental problem in computing, that of sorting. Just as there are dozens of sorting algorithms, so too there are dozens of source coding algorithms, each with its own distinctive features and applications. There is also, again as is the case with sorting, a gaggle of specialized coding sub-

problems that can be handled elegantly and economically by correspondingly specialized techniques. And there are some truly beautiful structures and analyses. To say that "Huffman coding is how we do coding" (an assertion implicit in the treatment of compression given by a number of textbooks) is as false as saying "Bubblesort is how we do sorting". And, moving forward by thirty odd years, to say that "arithmetic coding is how we do coding" is only slightly less naive than saying "Mergesort is how we do sorting". Just as a computing professional is expected to have a detailed understanding of Bubblesort, Heapsort, Quicksort, and Mergesort, together with an appreciation of the applications in which each should be preferred, so too should a computing professional – and certainly one professing knowledge of data compression techniques – have an appreciation of a range of coding mechanisms. The one or two coding mechanisms described in most texts should be regarded as a start, nothing more.

Hence this book. It examines in detail a wide range of mechanisms that have been proposed for coding, covering nearly fifty years of methods and algorithms, with an emphasis on the practicalities of implementation and execution. The book includes descriptions of recent improvements to widely-known methods such as minimum-redundancy (Huffman) coding and arithmetic coding, as well as coding problems with additional constraints, such as length-limited coding, alphabetic coding, and unequal letter-cost coding. It concludes with a chapter that examines three state-of-the-art compression systems, describing for each the type of coder employed and the reasons for that choice. Our intention is to be practical, realistic, detailed, and informative. Most of the techniques described have been tested, with compression and speed results reported where it is appropriate to do so. Implementations of a number of the principal mechanisms are available on the Internet, and can be used by those who seek compression, but are willing to forgo the details.

We also believe that this book has a role as a text for graduate and advanced undergraduate teaching. A suggested lecture schedule covering a 24 lecture-hour subject is available on the book's website at `www.cs.mu.oz.au/caca`; and we have included sufficient advanced material that even the keenest of graduate students will be challenged.

The remainder of this preface gives more details of the contents of the chapters that follow. So if your curiosity has already been pricked, feel free to go now to Chapter 1, it is where you will be shortly in any case. If you are not yet sold – if you are sceptical of the claim that coding is as fascinating as sorting, and almost as important – read on.

Outline of the book

Chapter 1 further motivates the need for compression, and explains in more detail the distinction between coding and modeling. It then defines the coding problem, and gives a number of simple examples of compression systems and how they might employ different coders. Chapter 1 concludes with a section that contains a modest amount of mathematical background that is required for some of the later analyses.

Chapter 2 then investigates fundamental limits on what can be achieved by a coder. These are the unbreakable rules that allow us to gauge how good a particular coder is, as we need only compare the actual length of its output with the entropy-based lower bound. And, perhaps surprisingly, these fundamental limits can also guide us in the implementation of effective coders. Ramamohanarao (Rao) Kotagiri, a colleague at the University of Melbourne, has observed that one of the enduring (and perhaps also endearing) desires of the human race is to condense multi-dimensional data sets into a single scalar value, so that alternative methods can be compared. In the field of compression, this desire can be sated, as if we ignore any resource implications, all behavior is measured in bits and bytes, and the same is true of the lower bounds.

The class of static coders – those that make no attempt to manage a probability distribution on alphabet symbols – is described in Chapter 3. While it seems counter-intuitive that these mechanisms can possibly be useful, they are surprisingly versatile, and by virtue of their simplicity and lack of controlling parameters, can sometimes yield better compression effectiveness than their more principled stable-mates. They also tend to be robust and reliable, just as a family sedan is in many ways a more practical choice for day-to-day motoring needs than is an exotic two-seater sports car.

Nevertheless, more principled methods result in better compression effectiveness whenever the cost of sending the symbol probabilities can be spread over sufficiently many message symbols. Chapter 4 examines the family of minimum-redundancy codes: those that assign a discrete bit-pattern to each of the symbols in the alphabet, and do so in a systematic manner so as to minimize overall message length. The best-known of all compression methods – Huffman coding, so named because of the famous paper authored by David Huffman in 1952 – is one example of a coding algorithm in this category. Chapter 4 gives details of the implementation of Huffman coding, and shows that minimum-redundancy coding is a far more efficient process than the follow-pointers-through-a-code-tree approach suggested by most textbooks.

If the restriction that all codewords must be discrete bits is lifted, we get the family of arithmetic codes, the subject of Chapter 5. The singular advantage of arithmetic codes is that they can very closely approximate the lower bound

on the length of the compressed representation that was mentioned earlier in connection with Chapter 2. Amazing as it may seem at first, it is possible for symbols in a message to contribute less than one bit to the compressed output bitstream. Indeed, if the probability of a symbol is sufficiently close to 1 as to warrant such a short codeword, it might contribute only 0.1, or 0.01, or 0.001 bits – whatever is appropriate. Chapter 5 also considers in detail the implementation of arithmetic coding, and describes variants that trade a small amount of compression effectiveness for increased execution speed.

Chapter 6 examines the problem of adaptive coding. The preceding two chapters presume that the probabilities of symbols in the source alphabet are known, and that all that is necessary is to calculate a code. In fact, in many situations the symbol probabilities must be inferred from the message, and moreover, inferred in an on-demand manner, in which the code for each message symbol is finalized before that symbol contributes to the observed symbol probabilities. Coding in this way allows one-pass compression, an important consideration in many applications. Considerable complexity is introduced, however, as the codewords used must be maintained on the fly. Chapter 6 examines algorithms for manipulating such codeword sets, and considers the delicate issue of whether static codes or adaptive codes yield better compression effectiveness.

Chapter 7 broadens the quest for codes, and asks a number of what-if questions: what if no codeword may be longer than L bits for some limit L; what if the codewords must be assigned to symbols in lexicographic order; what if the channel alphabet (the set of output symbols that may be used to represent the message) is non-binary; and what if the symbols in the channel alphabet each have different costs of transmission. Unsurprisingly, when constraints are added, the codes are harder to find.

The last chapter closes the circle. Chapters 1 and 2 discuss compression systems at a high level; Chapter 8 returns to that theme, and dissects a number of recent high-performance compression techniques, describing the models that they embody, and then the coders with which those models are coupled. The intention is to explain why the particular combination of model and coder employed in that product is appropriate, and to provide sufficient explanation of the model that the interested reader will be able to benefit. The mechanisms covered include the LZ77 sliding window approach embodied in GZIP; the Prediction by Partial Matching mechanism used in the PPM family of compression systems; and the Burrows-Wheeler Transform (BWT) approach exploited by BZIP2. In dealing in detail with these complete compression systems, it is hoped that the reader will be provided with the framework in which to design and implement their own compression system for whatever application they have at hand. And that they will enjoy doing so.

Acknowledgements

One of the nice things about writing a book is getting to name names without fear of being somehow unacademic or too personal. Here are some names, people who in some way or another contributed to the existence of this work.

Research collaborators come first. There are many, as it has been our good fortune to enjoy the friendship and assistance of a number of talented and generous people. Ian Witten has provided enthusiasm and encouragement over more years than are worth counting, and lent a strategic nudge to this project at a delicate moment. Lang Stuiver devoted considerable energy to his investigation of arithmetic coding, and much of Chapter 5 is a result of his efforts. Lang also contributed to the interpolative coding mechanism described in Chapter 3. Justin Zobel has been an accomplice for many years, and has contributed to this book by virtue of his own interests [Zobel, 1997]. Others that we have enjoyed interacting with include Abe Bookstein, Bill Teahan, Craig Nevill-Manning, Darryl Lovato, Glen Langdon, Hugh Williams, Jeff Vitter, Jesper Larsson, Jim Storer, John Cleary, Julien Seward, Jyrki Katajainen, Mahesh Naik, Marty Cohn, Michael Schindler, Neil Sharman, Paul Howard, Peter Fenwick, Radford Neal, Suzanne Bunton, Tim C. Bell, and Tomi Klein. We have also benefited from the research work undertaken by a very wide range of other people. To those we have not mentioned explicitly by name – thank you.

Mike Liddell, Raymond Wan, Tim A.H. Bell, and Yugo Kartono Isal undertook proofreading duties with enthusiasm and care. Many other past and present students at the University of Melbourne have also contributed: Alwin Ngai, Andrew Bishop, Gary Eddy, Glen Gibb, Mike Ciavarella, Linh Huynh, Owen de Kretser, Peter Gill, Tetra Lindarto, Trefor Morgan, Tony Wirth, Vo Ngoc Anh, and Wayne Salamonsen. We also thank the Australian Research Council, for their funding of the various projects we have been involved in; our two Departments, who have provided environments in which projects such as this are feasible; Kluwer, who took it out of our hands and into yours; and Gordon Kraft, who provided useful information about his father.

Family come last in this list, but first where it counts. Aidan, Allison, Anne, Finlay, Kate, and Thau Mee care relatively little for compression, coding, and algorithms, but they know something far more precious – how to take us away from our keyboards and help us enjoy the other fun things in the world. It is because of their influence that we plant our tongues in our cheeks and suggest that you, the reader, take a minute now to look out your window. Surely there is a nice leafy spot outside somewhere for you to do your reading?

Alistair Moffat,
Melbourne, Australia

Andrew Turpin,
Perth, Australia

January 2002