V.42bis and Other Ziv-Lempel Variants

by

Clark Thomborson

April 1991

Technical Report 91-02

University of Minnesota
Duluth

Computer Science

V.42bis and Other Ziv-Lempel Variants

by

Clark Thomborson

April 1991

Technical Report 91-02

Department of Computer Science
University of Minnesota
Duluth, Minnesota 55812
U.S.A.

Research supported by the National Science Foundation, through its Design, Tools and Test Program under grant number MIP 8706139

V.42bis and Other Ziv-Lempel Variants

Clark Thomborson*
Computer Science Department,
University of Minnesota at Duluth
Duluth, MN 55812
U.S.A.
cthombor@ub.d.umn.edu

Abstract

The recently adopted V.42bis standard for data compressing modems is discussed from algorithmic, experimental, practical, and marketing standpoints. V.42bis is a conservative and economically-implementable scheme, one that compresses text about as well as the Lempel-Ziv-Welch algorithm of the Berkeley Unix "compress" utility. Other Ziv-Lempel variants are discussed briefly. Even though better compression ratios can be obtained by these variants, V.42bis is shown to be eminently suitable for implementation on a contemporary modem with an 8-bit microprocessor, 40 Kbytes of RAM, 32 Kbytes of ROM, a 9.6 KBaud V.32 modem-modem connection, and a 19.2 KBaud EIA-232-D modem-terminal connection.

1 Introduction

It is now possible to buy a modem with the following characteristics.

- Price: $650 (discounted) to $1150 (list)
- Modem to terminal or computer connection: EIA-232-D (300 to 19,200 Baud)
- DCE to telephone network connection: V.32 (300 to 9600 Baud)
- Error correction: V.42 (16-bit CRC with retransmission, based on HDLC) [19]
- Data compression: V.42bis (modified Ziv-Lempel) [18]
- Implementation: 8-bit microprocessor (10 MHz Z80) with 40 Kbytes RAM and 64 Kbytes ROM.

I will call such a device a V.42bis modem, although I caution the reader that other uses of the V.42bis standard are possible. The implementation and pricing information correspond to a MultiTech MT932EAB, introduced in late July 1990. Six similar products are currently being offered by other manufacturers, according to an informative set of product reviews in PC Magazine [7].

Readers who are unfamiliar with the communication standards in the list above will find some information in Section 2 below. Should that discussion prove insufficient, I heartily encourage

*Research supported by the National Science Foundation, through its Design, Tools and Test Program under grant number MIP 8706139.

you to read the PC Magazine reviews [7] and to obtain a copy of Black's excellent compendium and discussion of the V-, EIA-, and X- series standards [5].

It is worth noting at the outset that the V.42bis modem's throughput is often limited by its EIA-232-D connection to the terminal or computer, or by the terminal or computer itself. At the 3:1 compression ratio observed on some transmissions, a V.42bis modem has a potential throughput of 3200 characters per second. Unfortunately, throughputs of this magnitude are rarely observed in practice, for the reasons listed in Section 2.

In Section 3, I describe the data compression method of V.42bis. Briefly, it is a textual substitution scheme based on Ziv and Lempel's second algorithm [27, 12]. V.42bis incorporates Welch's coding scheme [25, 11], as well as Miller and Wegman's (and, independently, Storer's) idea for incremental modification of a full dictionary using a least-recently-used (LRU) heuristic [16, 17, 22, 8, 23, 15]. V.42bis also has an efficient "cleartext" escape mechanism, differing in implementation but not in spirit from that proposed by Fiala and Greene [13].

Section 4 lists the compression ratios obtained by a sample V.42bis implementation, as well as the ratios obtained on the same files by the 4.3 bsd Unix "compress" utility, and by state-of-the-art Ziv-Lempel codes by Miller and Wegman [16], Rogers and Thomborson [20], and Fiala and Greene [13]. My data shows that a well-implemented V.42bis modem will acheive 2.3:1 compression on English text, 1.6:1 compression on binaries, and 3.2:1 compression on source code. A pessimally-implemented V.42bis-compliant modem will not compress anything.

Section 5 concludes the article with a brief summary and some comments on the future of data-compressing modems.

2 Communication Standards and the Modem Market

EIA-232-D An external modem for a terminal or a personal computer is universally connected in accordance with the EIA-232-D signalling standard. This standard is the most recent update of the venerable RS-232C convention. In a fully-compliant EIA-232-D implementation, the data rate is limited to to 19.2 KBaud, where a Baud is one bit-frame per second. However, most modern terminals and some personal computers will communicate over an EIA-232-D run at 38.4 KBaud.

The asynchronous signalling convention used by EIA-232-D encodes an 8-bit character into 10, 10.5, or 11 bit-frames, depending on the number of "stop bits" selected. A V.42bis modem's throughput is therefore upper-bounded by 1920 characters per second (1920 cps), if its EIA-232-D connection is run at 19.2 KBaud. Even at 38.4 KBaud, this interface provides at most 3840 cps, where this limiting behavior can be observed only if both the terminal and the modem have fast enough microprocessors, and enough internal buffering, to run without invoking a flow-control mechanism. A more assured way of obtaining higher throughput would be to replace the EIA-232-D connection with a high-bandwidth serial port such as the 10 MBaud EIA-422-A or the 100 KBaud EIA-423-A. Unfortunately, these high-bandwidth standards show no immediate promise of displacing the ubiquitous EIA-232-D.

V.32 The V.32 standard offers the highest bandwidth, 9600 Baud, for a dial-up connection in the general switched telephone network. (Leased lines can give even higher bandwidths.) Unlike the asynchronous EIA-232-D, V.32 is a synchronous protocol, sending characters in packets. If the packets contain 256 data characters apiece, the control overhead is approximately 12%; 9600 KBaud thus translates into about 1060 cps.

Comparing the 1060 cps bandwidth of V.32 with the 1920 cps offered by EIA-232-D, we see

the attraction of a data-compressing modem. If the data from the DTE is compressed by a ratio lower than 1060/1920 (= 0.55) before being packetized by V.32, a 19.2 KBaud EIA-232-D will replace V.32 as the limiting factor. By a similar reasoning process, if the EIA-232-D is run out-of-spec at 38.4 KBaud, the smallest useful compression ratio is 1060/3840 (= 0.28).

Mike Byrd of PC Magazine has acheived a throughput of 3440 cps when sending a highly-compressible file between two high-speed personal computers, showing that it is possible to use up to 90% of the bandwidth available on a 38.4 KBaud EIA-232-D connection to a V.42bis modem [7].

Future Standards The V.32 specification is currently being revised to permit data rates up to 14.4 KBaud. The higher-bandwidth V.32bis is expected to receive final approval from CCITT in February 1991 [7]. Within months of V.32bis approval, it will probably be incorporated into a new generation of V.42bis modems. The data compression feature in such modems will have less range of application than in the current V.42bis modems, however, since an (out-of-spec) 3840 cps EIA-232-D connection will be a bottleneck whenever the compression ratio is less than 1600/3840 (= 0.42).

Another development in telephone network signalling, hovering on the "near-future horizon" for many years, is the Integrated Services Digital Network (ISDN). When and if your local telephone loop is digitized, your dialup connections will run at 8000 cps, without a modem. This is four times the maximum in-spec bandwidth of the EIA-232-D standard, and twice its out-of-spec bandwidth, so it will finally be obsolete. Instead, one might expect to make a V.42bis ISDN file-transfer connections with one's personal computer, with an effective bandwidth of 20,000 cps for English text and 13 Kbyte/sec for executables.

V.42 The V.42 specification is a standardized method for error control, based on the HDLC protocol used in some computer networks. The V.42bis standard is, formally, a revision to V.42. Hence its "bis" suffix.

One can operate a V.32 modem at 9600 Baud without V.42 error correction, but its utility is limited by the frequent transmission errors on a typical dialup telephone connection. With the addition of V.42 error control, a V.32 modem is essentially error-free, but its apparent bandwidth drops proportionally with its transmission error rate.

Transmission errors are detected by V.42 with a 16-bit CRC code computed on the content of each packet. When a modem receives a packet whose locally-computed CRC does not agree with the CRC value received in that packet, the receiving modem sends a special control packet requesting a retransmission. Several packets may be outstanding at a time, so that it is not necessary for the transmitter to wait for an acknowledgement before proceeding to transmit the next packet. Thus, a telephone connection introducing one bit-error per minute will result in one extra data packet transmission each minute. This has a negligible impact on throughput, even when the packets have 256 characters. Somewhat higher error rates can be accomodated with small bandwidth degradation by limiting the maximum packet size to 32 characters.

The market for V.42bis modems The price and implementation of a V.42bis modem are strongly coupled. At present, the market price for a V.42 modem is about $600, with approximately a $50 additional charge for the V.42bis feature. Both products are typically built around an 8-bit microprocessor, perhaps 8 Kbytes of RAM for packet buffering, twenty or thirty Kbytes of ROM for the firmware, and perhaps eight Kbytes of RAM for V.42 buffering. When the V.42bis option is selected, an additional three Kbytes of RAM are required for each (minimal-sized) V.42bis data compression table. Two such tables are required for bidirectional compression; some modems may implement asymmetric links using just one table.

In computer-to-computer communications, the data compression feature of V.42bis must compete with low-cost, higher-performance software. The transmitting computer can run a compression algorithm on the data before sending it to a non-compressing V.42 modem; the receiving computer can run the corresponding decompression algorithm. This approach permits one to select the compression method most suitable for the data being transmitted. If one is transmitting scanned image data, for example, one can do much better (10:1 compression is routine) if the V.42bis feature is bypassed. Another advantage of compressing data, before transmitting it to the modem, is that a 19.2 KBaud EIA-232-D connection is no longer a bottleneck.

It is easy to understand why there is such a small price differential between V.42 and V.42bis modems, despite the novelty and increased performance of V.42bis. The principal market for V.42bis modems is in computer-to-terminal communications, not in computer-to-computer communications. Users with high-speed terminals will be willing to pay a premium for obtaining 1920 cps throughput on V.42bis instead of 1100 cps on V.42. Many users, however, are currently satisfied to obtain 270 cps with a $100 2400 Baud modem, so the V.42bis product is aimed at the top end of the market.

Applications at UMD If my experience is any indication, V.42bis modems are useless at present, but promise some near-term future utility.

My Zenith Z-39 and Z-29 terminals do not have a 38.4 KBaud setting. They do not run noticeably faster at 19.2 KBaud than at 9600 Baud. Furthermore, their effective display bandwidth seems to be limited to 1000 cps, so an (uncompressing) V.42 modem gives the same performance as a more expensive V.42bis modem.

I have yet to see a terminal emulator on a personal computer that can display at a rate exceeding 1000 cps, so I can not recommend the purchase of a V.42bis modem for this application either.

As argued above, when transferring files from one computer to another, a software package can do at least as well (and sometimes much better!) than V.42bis.

The fastest terminal currently on my campus, a Wyse, can keep up with a full-bandwidth 19.2 KBaud input stream, but not 38.4 KBaud. Its effective input bandwidth therefore lies between these two data rates, perhaps 2500 cps. Unfortunately, the Telnet/Ethernet connection used between our modem "Annex" box and the UMD mainframes is limited to about 1000 cps, so a Wyse/V.42bis setup does not give any better performance than does a Z-39/V.42 setup.

V.42bis modems are nonetheless a good investment for our campus because their incremental cost over a V.42 modem is small, and because terminals, home computers, and inter-campus connections should soon be fast enough to take advantage of the added speed of V.42bis.

3 V.42bis Compression

V.42bis is a textual substitution scheme based on Ziv and Lempel's second algorithm [27]. Whenever a codeword is transmitted, the dictionary is augmented by a new codeword consisting of the codeword being transmitted, concatenated with the next character in the input stream. Initially, the dictionary consists of all one-character strings.

Like Welch's algorithm [25], V.42bis transmits only previously-created codewords, expressing these as binary strings whose length is the logarithm of the current dictionary size. (The 4.3 bsd Unix "compress" utility is a highly-tuned implementation of Welch's algorithm.) Note that Welch's algorithm is a significant improvement over the Ziv-Lempel algorithm [27], since Ziv and Lempel suggest transmitting a just-created codeword by sending a previously-created codeword

index followed by an unencoded character.

Unlike Welch's algorithm, the most-recently-created codeword is *not* a candidate for transmission in V.42bis. This simplifies and accelerates the decoding process slightly, with little cost in compression or encoding speed.

Like Storer's LRU algorithms [23, 10] and the Miller-Wegman codes [16], V.42bis adds new codewords to a full dictionary by deleting some codeword that is not a prefix of any other codeword. Candidates for deletion in V.42bis are found by cycling through the dictionary, a process that is a time- and space-saving approximation to the exact least-recently-used (LRU) deletion heuristics proposed by Storer and Miller-Wegman.

V.42bis provides a short escape-code sequence to shift into "transparent mode" whenever this would provide greater compression (or less expansion). This is somewhat like the literal mode of Fiala and Greene's Ziv-Lempel variant [13]. However, Fiala and Greene propose sending the length of the literal string along with the literal escape code. In V.42bis, transparent mode is entered with an escape sequence, and exited with another escape sequence, so that a length determination is unecessary.

The V.42bis dictionary is updated during transparent mode, just as it is during compressed mode, so that the dictionary is constantly adapting to the modem input.

To avoid undue expansion during transparent mode, V.42bis changes the identity of the escape character <esc> every time this code is encountered in the input stream, using the formula <esc> = (<esc> + 51) mod 256. This clever scheme will find an unused character in the input stream to serve as an escape, if there is such a character. Thus a file that doesn't contain all 256 character codes can be tranparently encoded with at most 255 extra characters, yielding a worst-case expansion of 1.0 for long files.

If the input does contain all 256 character codes, but if the input characters are uncorrelated with the escape character update function, then any given character transmission has only one chance in 256 (on long-term average) of being encoded as the two-character sequence <esc><eid>. The expected expansion for uncorrelated input in transparent mode is thus $1 + 1/256 = 1.005$. Of course, a pathological input can be constructed in which the character sequence tracks the escape code update function, leading to a worst-case expansion of 2.0. This last case is highly unlikely to occur in any application, so I conclude that essentially no expansion occurs in the transparent mode of V.42bis.

V.42bis also provides an escape-code sequence that causes the current dictionary to be discarded. This sequence, as well as the transparent-mode escape, can be transmitted by the compressor at will, presumably when this would improve the compression ratio.

Algorithmically, V.42bis is underspecified, as indicated in the previous paragraph. A V.42bis-compliant modem need not provide any data compression at all, when it transmits: it might never leave transparent mode, or it might enter compressed mode only when *expansion* would result. A more likely scenario is that some V.42bis modems will have better compression characteristics than others, due to differences in their heuristics for sending escape codes.

Another area of algorithmic underspecification in V.42bis lies in its openness regarding two key parameters, the maximum dictionary size and the maximum codeword length. When a V.42bis connection is first established, the calling modem and the answering modem negotiate these values. A fully-compliant, but underconfigured, V.42bis modem can insist on the minimum values of 512 codewords with at most 6 characters apiece.

In a typical V.42bis implementation, a dictionary is implemented as a pointer-list trie with one node per codeword. A node in a pointer-list trie contains two pointers (lchild/rsibling) and a character; a pointer requires two bytes on an 8-bit microprocessor with a 16-bit address space. The trie nodes should be stored in a dense linear array, so the string corresponding to a node can

be uniquely specified by transmitting the index of that node in the array. Note that 16-bit node indices could be used instead of pointers, if this is more efficient, transforming the structure into a cursor-list trie.

To allow rapid deletion of leaf nodes, a third "parent" pointer could be added to each record. Alternatively, an additional 8-bit unsigned integer would suffice. This could be used to indicate the depth of a trie node, since the V.42bis standard limits code words to a length of at most 250 characters. The rsibling field of the rightmost child could then contain a parent pointer. This pointer could be disambiguated from a normal rsibling pointer by a change in the depth of the node being referenced. A pointer-list trie can thus be implemented with just 6 bytes per node.

A minimally-compliant (512 codewords/dictionary) V.42bis modem thus requires just 3 Kbytes of data space to hold its dictionary. Such a modem would only be able to compress in one direction of communication, as would be appropriate on a home terminal: the display data can be at high bandwidth, but keyboard input is always low bandwidth.

The MultiTech MT932EAB on my desk does not seem to allow unidirectional compression, so it must maintain two dictionaries in its 40 Kbytes of RAM. Some of this RAM must be used for V.32 buffering. I thus deduce that it is unable to operate with two 4096-codeword dictionaries. Instead, it probably runs the V.42bis compression algorithm with at most 2048 codewords in each of its dictionaries.

A closed hash trie, along the lines suggested by Welch [25], is another method that could be used to store the dictionaries in a V.42bis modem. In order to support V.42bis-style incremental updates, an 8-bit count field must be added to each hash-trie node, to keep track of the number of its descendants. A leaf node has a 0 count; whenever a leaf is deleted, its parent's count field should be decremented. Total storage per entry in the hash table would then be 6 bytes: a 1-byte suffix character, a 2-byte pointer to the prefix (its parent node), a 1-byte count field, and a 2-byte code number. If the hash table is 90% full, then we have 6.6 bytes per trie node, making this structure slightly larger than the 6 bytes per node in a minimal pointer-list trie. Also, the decompressor requires another 2 bytes per dictionary entry to maintain a mapping from codes onto hash indices. Closed hash tries are therefore at a significant disadvantage in the present memory-limited implementations of V.42bis.

If 64K-node dictionaries ever became prevalent, a random or otherwise incompressible input file would expose a weakness in a pointer-list trie: the average search path would be of length 128 (approximately), as opposed to a small constant in a hash trie. The worst-case search path in the pointer-list trie would only be about 256, however, as opposed to 64K in the hash trie. (V.42bis, unlike Welch's algorithm [25], will *not* limit the length of a collision chain in a hash search.) So pointer-list tries will be a preferred implementation for V.42bis, due to better worst-case performance, even on the largest dictionaries.

Tim Bell is currently making a comparative study of seven methods of organizing the data structure for a Ziv-Lempel compressor [2].

4 Experimental Results

John Copeland, of the V.42bis study group, was kind enough to send me C-language source code for an implementation of the V.42bis standard. The comments and experimental results in this section are obtained from his version 6.83, dated 7/16/89, of hv42b3.c. Hayes Microcomputer Products, Inc. owns the copyright to this code, which was "distributed for experimentation aimed at developing a CCITT V.42 data compression standard."

The dictionaries in hv42b3.c are implemented as rchild/lsibling/rsibling/parent pointer-tries. The sibling lists were sorted by their suffix character.

	Dictionary Size			
	512	1024	2048	4096
csh	.65	.62	.62	.62
essays	.57	.49	.45	.43
news	.70	.64	.60	.56
pascal	.52	.41	.35	.31
tex	.59	.50	.45	.42

Table 1: Effect of dictionary size on V.42bis compression ratio, when maximum codeword length is 16.

When I ran hv42b3.c on my Sun-3/50, my 100 Kbyte testfiles were compressed at rates of 6 to 10 Kbytes of input data per second. On a 10 MHz Z80, I estimate that hv42b3.c would run 2 to 4 times slower, due to the lower MIPS rate and the narrower data bus. Some code optimization will thus be necessary to keep the CPU from becoming the bottleneck in a V.42bis modem, since the CPU in such a modem has a number of other tasks. The CPU must handle V.42bis compression on the transmission channel, V.42bis decompression on the receiving channel, as well as V.42 error correction and V.32 packetizing on both channels. Perhaps the ATI 9600etc/e modem has a CPU bottleneck of this sort, which would explain why it was unable to deliver more than 1977 cps throughput, on an experimental setup where its fastest competition ran at 3440 cps [7].

I have run a number of compression measurements on the V.42bis implementation used by the study group. All measurements reported in this paper are for the following five files. The first, labelled "csh," is the 122880-byte executable c-shell in my Sun 4.2 Unix. The second test file, labelled "essays," consists of 176369 bytes of concatenated freshman essays. It is cleartext English, containing no text formatting commands. The third file, labelled "news," is 210931 bytes of articles representing a (small) portion of a day's Usenet feed. This file is a good test of a compression algorithm's ability to respond quickly to a rapidly-changing source, since the article header fields are quite different from the article bodies, and the article bodies are also variable (ranging from cleartext English to source code). The fourth file, labelled "Pascal," is 119779 bytes of source code to a student-written program. It is highly compressible, due to its repeated use of long identifiers. The fifth and final test file, labelled "tex," is the 201746-byte LaTex manual, vintage 1984, containing both cleartext and text formatting commands.

Table 1 shows the compression ratios obtained by hv42b3.c on the five test files, when the length of the longest codeword is 16 and the maximum length of the dictionary was varied between 512 and 4096. In this table, and throughout this section, we define compression ratio to be the length of the compressed file divided by the length of the input file. A compression ratio of 0.50 is thus the "2:1 compression" mentioned earlier in this paper as being typical of V.42bis when applied to a text file.

Note that increasing the size of the dictionary has a dramatic effect on compression ratio for all files save "csh." Perhaps the most striking feature of Table 1, however, is that even a 512-codeword dictionary is sufficient to obtain a fairly high level of compression on our long test files.

Table 2 is identical to Table 1, except that the length of the longest codeword is restricted to 6. Note that these figures are only slightly larger than those of Table 1, indicating that the codeword-length parameter is not very important to V.42bis performance, at least for our five test files.

Table 3 shows the compression ratios obtained by the Unix "compress" utility [25] on our five test files, as a function of dictionary size. Note that "compress" is a poor algorithm for

	Dictionary Size			
	512	1024	2048	4096
csh	.65	.63	.63	.64
essays	.57	.49	.46	.43
news	.70	.64	.60	.57
pascal	.53	.43	.38	.35
tex	.60	.51	.46	.44

Table 2: V.42bis compression ratio, for various dictionary sizes, when maximum codeword length is 6.

	Dictionary Size				
	512	1024	2048	4096	65536
csh	.98	.82	.89	.75	.64
essays	.67	.53	.49	.46	.40
news	.91	.73	.71	.67	.54
pascal	.70	.50	.43	.37	.30
tex	.85	.61	.53	.48	.39

Table 3: Unix-Welch compression ratio, for various dictionary sizes.

small dictionaries. Even with a 64K dictionary, it is only slightly better than V.42bis with a 4K dictionary.

Tables 4 and 5 compare the performance of hv42b3.c with a 4K codeword dictionary (max codeword length = 16) with that of a number of other Ziv-Lempel variants. Where possible, I use the abbreviations of the Fiala-Greene paper [13].

Table 4 consists of those algorithms which, like V.42bis, send codeword indices in straight binary form. The algorithms of Table 5 gain compression, at some penalty in speed, by using Huffman or arithmetic codes to compress the codeword indices.

UW is the Unix-Welch Berkeley "compress" utility, with 16-bit codes (64K dictionary entries)[25]. As noted previously, UW is a variant of Ziv and Lempel's second algorithm [27].

A1, B1, A2, B2, and C2 are Fiala and Greene's algorithms. A1 is a ZL1 scheme [26] with a 4K-character buffer, augmented with a literal transmission mode, using a suffix trie to accelerate the buffer searches. Its compressed output contains two types of codewords: "literal x," meaning that the decompressor should pass the next x characters directly to the output, and "copy x y," meaning that the decompressor should go back y characters in the output buffer and append the next x characters to the output buffer. B1 is similar to A1, but optimized for speed: it builds a Patricia trie with only some of the substrings contained in the last 4K characters of input. A2 and B2 are similar to A1 and B1, respectively, with 16 Kbyte buffers and a static Huffman "back end." C2 is designed for maximum compression. Its Patricia trie contains all the substrings contained in the last 16K characters of input. It gains compression efficiency over the usual ZL1 scheme, since it does not waste code space on duplicate substrings. Because A2, B2, and C2 use a Huffman code to send trie indices, I have listed their compression ratios in Table 5.

MW1 and MW2 are the Miller-Wegman algorithms[16, 17], as implemented by Dan Greene of Xerox PARC. MW1 is a ZL2 variant [27], with a 4096-entry dictionary, an LRU replacement algorithm, and Welch-style output coding [25]. MW2 is like MW1, augmented with "string extension." It is thus an ID-LRU, in Storer's terminology [23].

MWP, of Table 5, is a high-compression, high-speed implementation of MW2 by Roberto Pasqui of IBM Italy. It uses a static Huffman code with only three code lengths. The shortest

codes refer to the most-recently-used set of 128 codewords; medium-length codes refer to a set of less-recently-used 512 codewords; and the longest codes refer to the oldest 4096 codewords.

Y64 and Y512 are the "yabba" coders, placed into the public domain recently by Dan Bernstein [4]. Yabba is a ZL2 variant [27] that adds one string pc to its dictionary for each input character c, where p is the longest suffix of the already-processed input such that p is currently in the dictionary. The higher-compressing Y512 has a 512K-entry dictionary; Y64 has a 64K-entry dictionary.

Like MWP, Rogers-Thomborson code RT [20] uses a back-end coder to transmit frequently-used codewords with fewer bits than the rarely-used ones. Since RT uses an arithmetic code rather than a static Huffman code, however, it is very slow. RT grows its dictionary in a similar fashion to the MW2 and MWP algorithms, except that it does *not* add new dictionary entries ending with a blank space when doing character extension. In such cases, only string extension occurs. This heuristic tends to increase the fraction of dictionary entries that are complete English words or phrases, when the input is textual. Since there are usually few ASCII blanks in a binary file, their compression is not degraded much: note that RT compresses slightly better than MWP, on all but the csh binary. We believe that most of this improvement is due to the non-blank heuristic, not to the arithmetic coding, but this hypothesis deserves test.

LHarc (see Table 5) is a public-domain code from Japan that is in common use for distribution of personal computer software on floppy disks. To install such packages, users must typically wait minutes for the software to be decompressed onto their hard disk or alternate floppy drive. I obtained a Unix version of LHarc over the Internet [24]. It runs at about 2 Kbytes/second on my Sun 3, or about 15 times slower than the Unix "compress" utility UW. As may be seen from Table 5, however, LHarc has better compression ratios. No documentation was included with my version of LHarc, although it appears to run in two passes: a ZL1 coding [26] with a 4 Kbyte buffer, followed by a dynamic Huffman recoding of the Ziv-Lempel indices. This Huffman "back end" probably accounts for most of the compression improvement and speed degradation, relative to UW.

Freeze is a public-domain code that recently emerged from the Soviet Union [6]. Like LHarc, it is a ZL1 variant [26] with a dynamic Huffman back end, although it has an 8 Kbyte buffer. Since single-character literals appear in the output stream, Freeze is more precisely an LZSS variant [21, 1]. Its output coding appears to work a little better than that of LHarc, and it runs at about the same speed, so I will describe it here. Of a 512-symbol alphabet, the first 256 symbols are used to denote single-character literals; one symbol marks an end-of-file; and the last 255 symbols indicate the length l of the match ($1 < l \leq 256$). If a multiple-character match is selected, the next bits in the compressed stream indicate the most-significant 6 bits of the position of the match. These bits are sent with a static Huffman code of 1 to 8 bits. The 7 least-significant bits of the match position are sent in binary.

As may be seen in Table 5, the Bentley-Sleator-Tarjan-Wei code [3] achieves very good compression ratios for every file except csh. The data in Table 5 was obtained from Dan Greene's implementation. This data agrees with that from my own, independently-coded, implementation to within 0.01.

BSTW's compression ratios are remarkable, it uses less than 6 Kbytes of space for its data structures. A BSTW compressor, in my implementation, uses two linked lists of 239 cells containing words of up to 16 characters apiece. One list contains only "alphanumeric words"; the other list contains "nonalpha words." The input file is parsed into an alternating sequence of maximal-length alpha- and nonalpha- words; 0-length words must be allowed in the lists to allow for the eventuality that the input contains alphanumeric (or nonalphanumeric) substrings of length greater than 16. A previously-encountered word is transmitted with an adaptive Huffman encoding of the list position, then moved to the front of its list. A new word is transmitted with

	V42	UW	A1	B1	MW1	MW2	Y64	Y512
csh	.62	.64	.61	.62	.62	**.57**	.64	.60
essays	.43	.40	.45	.44	.42	.40	.39	**.37**
news	.56	.54	.56	.55	.56	.53	.54	**.49**
pascal	.31	.30	.25	.25	.31	**.17**	.25	.22
tex	.42	.39	.41	.39	.41	**.35**	.40	.35

Table 4: Compression ratios of Ziv-Lempel algorithms that do not also employ Huffman or arithmetic coding. Bold-face entries are row minima.

	A2	B2	C2	MWP	RT	LHarc	Freeze	BSTW	ppmC
csh	.54	.55	.52	.51	.55	.51	.50	.69	**.47**
essays	.40	.39	.36	.36	.34	.40	.39	.39	**.29**
news	.48	.48	.44	.47	.45	.48	.46	.53	**.40**
pascal	.15	.14	**.13**	.17	.16	.19	.15	.24	.18
tex	.34	.33	.31	.32	.31	.36	.33	.38	**.27**

Table 5: Compression ratios of algorithms employing Huffman or arithmetic coding. Bold-face entries are row minima over Tables 4 and 5.

an escape code followed by an adaptive Huffman encoding of the cleartext.

The Bell-Cleary-Witten code ppmC [9, 1] of Table 5 is based on a variable-order Markov modelling of the source, with arithmetic coding used to communicate state transitions. It achieves impressive compression ratios, however it requires the largest amount of data space (730 Kbytes) of any algorithm in my tables.

As indicated in Tables 4 and 5, hv42b3.c obtains markedly worse compression ratios than many other algorithms. Below, each competing algorithm is discussed in turn, to see if it offers a significant advantage in the V.42bis application.

The UW algorithm achieves slightly better compression than V.42bis on the test files, at the expense of 16 times the data space. Since a low-cost modem is memory-limited, UW is not an appropriate choice. Furthermore, as we saw in Table 3, a memory-limited UW is not competitive in compression ratio with V.42bis.

Fiala and Greene algorithm's A1 requires 145 Kbytes to compress with a 16K-entry dictionary; their other algorithms require even more space. Further research is necessary to decide if any of these algorithms are competitive with V.42bis, when restricted to a 10 or 20 Kbyte dictionary and an 8-bit microprocessor. Still, the A2, B2, and C2 algorithms would be a promising starting point for research on a future data compression standard.

Despite its LRU-Huffman "back end," MWP runs at a respectable 10K to 20K bytes/second on my testfiles, on an IBM PS/2 mod 80, 16 MHz 80386 under DOS 4.0 [14]. By way of comparison, the Unix "compress" utility UW runs at 25K to 50K bytes/second on my testfiles, on my Sun 3/50, and my V.42bis implementation runs at only 6K to 10K bytes/second. Even though my V.42bis implementation is not optimized for speed, I believe that MWP would be at least as fast as V.42bis, even on an 8-bit microprocessor such as a Z80. Despite its impressive speed and compression ratios, however, MWP is not directly applicable to the design of today's low-cost V.42bis modems, since it requires 60 Kbytes of space for its data structures.

BSTW requires just 6 Kbytes of data space, so it would "fit" in a low-cost modem. However, its adaptive Huffman encoding would severely the bandwidth of any such modem's low-performance processor. Of course, a much more expensive modem could employ a special-purpose, high-bandwidth, Huffman- (or arithmetic-) encoder/decoder chip. The poor perfor-

mance of BSTW on non-textual data could also pose a problem in many applications, although one might envision a revision to V.42bis allowing a compressor to enter "BSTW mode" whenever suitable data was received.

As argued in Sections 1 and 2 of this report, the V.42bis modem is limited by its EIA-232-D input port, not by its compression ratio, for textual data. In less cost-sensitive applications, the EIA-232-D bottleneck could be removed, and other schemes would be markedly superior to V.42bis. Thus MWP (perhaps with the RT conditional dictionary entry heuristic), the Fiala-Greene algorithm B2, BSTW, and ppmC deserve attention in the next round of standards-making activity.

5 Conclusion

This examination of some of the practical, algorithmic, and marketing aspects of the V.42bis standard has not uncovered any serious flaws in its specification. On the contrary, the algorithm embodied in this standard is competitive with other Ziv-Lempel variants, if these are required to run in a small data space on 8-bit microcomputer.

The future of data-compressing modems is uncertain. Microcomputers and their memory are rapidly becoming cheaper, by 20% to 30% per year. Thus the "quick-and-inexpensive" algorithmic choices embodied in V.42bis will soon be outmoded, and the "more computationally expensive" but better-compressing algorithms mentioned in this paper will become economical for text transmission. However, text transmission is only a fraction of current computer communication over the telephone network, and this fraction is rapidly decreasing. Source-specific compression algorithms (for voice, image, fax, etc.) can be run efficiently on a personal computer, acheiving much higher compression ratios than are offered by any of the algorithms in this paper, for these non-textual sources.

Following this line of reasoning, the main application of V.42bis modems will continue to be in low-cost computer-to-ASCII-terminal communications. I would therefore not expect to see many such modems in the year 2000, when a 10 MIPS personal computer should be only a little more expensive than an ASCII terminal. Such computers could run source-specific data compression algorithms at high bandwidth, not only for their communication lines, but also for their mass storage devices.

6 Acknowledgements

Many of the experiments described herein were run by the former UMD students Clyde Rogers and Julie Redland, or by Dan Burrows and Bill Marko of UMD Information Services. I am indebted to John Copeland, of Hayes Microcomputer Products, Inc., for a copy of the source code to a V.42bis-compliant compressor and for the test files used by the V.42bis study group. Last but not least, I'd like to thank Dan Greene of Xerox PARC, Tim Bell of the University of Calgary, and Victor Miller of IBM, for taking the time and trouble to run our test files through their compressors.

References

[1] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

[2] Timothy Bell. Longest match string searching for Ziv-Lempel compression. unpublished manuscript, 1991.

[3] Jon Louis Bentley, Daniel D. Sleator, Robert E. Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, April 1986.

[4] Dan Bernstein. Yabbawhap. *comp.sources.unix*, 24(73–76), March 1991.

[5] Uyless Black. *Physical Level Interfaces and Protocols.* IEEE Computer Society Press, Washington, DC, 1988.

[6] Leonid A. Broukhis. Freeze v2.2.1 alpha 3/27/91. *comp.sources.misc*, 17(68 and 74), March 1991.

[7] Mike Byrd. 9600-bps modems: Breaking the speed barrier. *PC Magazine*, pages 307–346, December 11 1990.

[8] A. D. Clark. *Analysis and Design of Source/Channel Codes for Noisy Communication Channels.* PhD thesis, Leicester Polytechnic, School of Electronic and Electrical Engineering, June 1987.

[9] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Computers*, COM-32(4):396–402, April 1984.

[10] James A. Storer (Data Compression Corporation). (title unknown). *United States Patent*, 4,876,541, 1989.

[11] Terry A. Welch (Sperry Corporation). High speed data compression and decompression apparatus and method. *United States Patent*, 4,558,302, 10 December 1985.

[12] Willard L. Eastman, Abraham Lempel, Jacob Ziv, and Martin Cohn (Sperry Corporation). Apparatus and method for compressing data signals and restoring the compressed data signals. *United States Patent*, 4,464,650, 7 August 1984.

[13] Edward W. Fiala and Daniel H. Greene. Data compression with finite windows. *Communications of the ACM*, 32(4):490–505, April 1989.

[14] Victor S. Miller. private correspondence, January 1991.

[15] Victor S. Miller and Mark N. Wegman (International Business Machines Corporation). Data compression method. *United States Patent*, 4,814,746, 21 March 1989.

[16] Victor S. Miller and Mark N. Wegman. Variations on a theme by Ziv and Lempel. Technical Report RC 10630 (#47798), IBM T. J. Watson Research Center, July 31 1984.

[17] Victor S. Miller and Mark N. Wegman. Variations on a theme by Ziv and Lempel. In A. Apostolico and Z. Galil, editors, *NATO ASI Series, Volume F12, Combinatorial Algorithms on Words*, pages 131 – 140. Springer-Verlag, Berlin, 1985.

[18] Omnicom, Inc., 115 Park St. SE, Vienna, VA 22180-4607 U.S.A., (703) 281-1135. *CCITT Draft Recommendation V.42bis*, 1989.

[19] Omnicom, Inc., 115 Park St. SE, Vienna, VA 22180-4607 U.S.A., (703) 281-1135. *CCITT Recommendation V.42*, 1989.

[20] Clyde Rogers and Clark D. Thomborson. Enhancements to Ziv-Lempel data compression. In *Proc. of the 13th International Computer Software and Applications Conference (COMPSAC-89)*, pages 324–330, September 1989.

[21] J. A. Storer and T.G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[22] James A. Storer. Textual substitution techniques for data compression. In A. Apostolico and Z. Galil, editors, *NATO ASI Series, Volume F12, Combinatorial Algorithms on Words*, pages 111 – 129. Springer-Verlag, Berlin, 1985.

[23] James A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, 1988.

[24] Yooichi Tagawa. Lharc v0.03 release 3. *comp.sources.misc*, 11(17–18), August 1989.

[25] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

[26] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.

[27] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT-24(5):530–536, 1978.