Dacom 450/500 Facsimile Data Transcoding

A. Agarwal, M. J. O'Connor and D. L. Mills

2 November 1981

1. Introduction

As part of our effort in support of the DARPA Internet Program, software modules to encode and decode facsimile data for the Dacom 450 and 500 models Computerfax facsimile machines have been constructed. The purpose of these modules is to map the data representations used by these machines to and from bit-map and run-length representations in programs for editing, transmission and archiving facsimile images. The modules are written in the PDP-11 MACRO-11 assembly language and can be incorporated into programs for, among others, the RT-11 operating system and the DCNET BOS or VOS operating systems.

The first part of this report describes in detail the Dacom 450 data compression algorithm and is an update and correction to an earlier memorandum [2]. Following this, the encoding and decoding algorithms are described along with the supporting programs and file formats. Reference [3] describes another implementation of the decoding algorithm. Grateful acknowledgment is made to E. A. Poe of Rapicom for his assistance in this effort.

The second part of this report describes briefly the Dacom 500 data compression algorithm as used by the INTELPOST electronic-mail network under development by the US Postal Service and several foreign administrations. These machines conform to the CCITT T.4 Draft Recommendation, described in [5]. Supporting programs and file formats are described.

2. Dacom 450 Data Compression Principles

The encoding algorithm for the Dacom 450 processes lines scanned by the machine to produce a two-dimensional run-length code described by Weber [1]; however, this article contains a number of errors and omissions, many of which were discovered only after considerable analysis and experimentation [2,3]. The machine operates over a coordinate space of l726 by approximately 2200 pels when in high-resolution (detail) mode. In normal (quality) mode the vertical resolution is halved, so that about 1100 lines are transmitted, while in express mode about 733 lines are transmitted (missed lines are filled in on playback by replicating previous lines).

Data are encoded two rows at a time using a two-dimensional run-length code. Each row-pair is scanned left-to-right and the line-pairs themselves processed top-to-bottom of the document. Figure 1 shows how the pels are represented.

```
    |          |          |
----+----------+----------+----
... |  x(1,j)  | x(1,j+1) | ...
```

```
----+----------+----------+----
... |  x(2,j)  | x(2,j+1) | ...
----+----------+----------+----
    |          |          |
      Direction of scan ->
```

Figure 1. Data Representation

For each j the vector (x(1,j),x(2,j)) represents the contents of the jth column, where x(i,j) can take on values of zero (white) or one (black). Each of the four possible vectors ranging over these values will be called a state (Dacom calls these "modes") with the succession of transitions between these states determined by the picture content of the particular line-pair. Scanning of the line-pairs follows one after the other with no special end-of-line code in the data itself. For the purpose of later discussion and comparison with the published data, the following conventions will be used (note: the pels read top-bottom):

Pels	Vector	State
W-W	(0,0)	0
B-W	(1,0)	1
W-B	(0,1)	2
B-B	(1,1)	3

The algorithm used by Dacom to generate the transmitted data as the columns are scanned can be described as the non-deterministic finite-state automaton (nfsa) shown in Figure 2. Conceptually, the nfsa starts at the beginning of a page in a designated state and at a point just after scanning the jth column in the jth state. It then scans the (j + 1)th column and enters that state while emitting the string of bits shown in the figure.

In the states corresponding to W-W (0) and B-B (3) a special run-length encoding techniques is used. There are two state variables associated with each of these two states, one variable used as a run-length counter and the other the field length (in bits) of this counter. Upon each entry to either of these two states the counter is initialized at zero and counts up for every additional column of the same state. At the end of the run the value of this counter is transmitted extending with high-order zeros, if necessary, to fill the field length specified. If, however, the counter equals 2**n - 1, where n is the field length, then a sequence of n one-bits is emitted and the counter re-initialized at zero with a field length of n + 1. Thus, if n = 3, a run length of three is transmitted as {010} and a run length of seven as {110}, while a run length of eight as two words, {111} followed by {0000}. The field-length variables are maintained separately for both the W-W and B-B states, and at each re-entry to either of these states the previous values are used.

```
                      0100
.--------------------->---------------------------------.
|                                                       |
|   .----------------<-----------------------------.   |
|   |                1                              |   |
|   V                                               |   |
     .--------------.                     .---------------.     |   |
     |              |                     |               |     |   |
     |              |        010          |               |     |   |
  .->|      1       |-------------------->|       2       |->.  |   |
  |  |              |                     |               |  |  |   |
 0|  |     B-W      |        101          |      W-B      |  |1 |   |
  \<-|              |<--------------------|               |<-'  |   |
     |              |                     |               |     |   |
     |              |               .---->|               |     |   |
     \--------------'               |     \---------------'     |   |
         |   A                      |      |     |   A          |   |
         |   |     .--------->------'      |     |   |          |   |
         |   |     |         1             |     |   |          |   |
         |   |     |                       |     |   |          A   V
         |   |     |                       |     |   |          |   |
     0111|   |1    |                       | 1000|   |1         |   |
         |   |     |                       |     |   |          |   |
         |   |     |                       |     |   |          |   |
         |   |     |                       |     |   |          |   |
         |   |     |          1011         |     |   |          |   |
         |   |     |    .-------<----------'     |   |          |   |
         V   |     |    |                        V   |          |   |
     .--------------.   |                 .---------------.     |   |
     |              |<--'                 |               |     |   |
     |              |        0            |               |     |   |
     |      3       |<--------------------|       0       |-----'   |
     |              |                     |               |         |
     |     B-B      |                     |      W-W      |         |
     |              |-------------------->|               |<--------'
     |              |        0            |               |
     |              |                     |               |
     \--------------'                     \---------------'
         |    A                          |    A
         |    |                          |    |
         \----'                          \----'
          run                             run
```

Figure 2. NFSA Model of Encoding

Field-length values are constrained not to exceed seven, so that runs exceeding l27 with n = 7 will be encoded as a separate 7-bit word of one-bits for each run of l27 except the last, which must always contain at least one zero-bit. The field length n is decreased by one under the following circumstances: the current run has been encoded as a single n-bit field, and for n in the range four through seven the two high-order bits are zero or for n equal to three the single high-order bit is zero. The field length is not allowed to be reduced below two bits.

The encoding algorithm starts in state 0 with both field lengths set to 7.

2.1. Dacom 450 Decoding Algorithm

For reasons of speed and simplicity it is desirable that the Dacom 450 decoding algorithm be modeled on the basis of a deterministic finite-state automaton (dfsa). Using straightforward formal procedures, the dfsa of Figure 3 can be constructed. This machine makes one state transition for every bit, except for the W-W (0) and B-B (3) states, which must be treated specially in any case. The states are labeled in such a way as to correspond to those of Figure 2 for states numbered from zero to three.

The decoded output symbols, in this case the columns corresponding to each of the states, are represented by the states themselves. Upon entry to state B-W (1) or W-B (2) a run-length counter is initialized to one. Each traversal of a loop back to the same state increments this counter and, upon exit to any other state, the value of this counter represents the number of columns to be produced. Upon entry to state W-W (0) or B-B (3) the run-length counter is initialized to zero and the associated field-length state variable n established. For each successive n bits of all-ones, the counter is increased by 2**n - 1 and then n itself increased by one, but not above seven. If the next n bits are not all ones, then the counter is increased by the value represented by the n-bit field plus one. Finally, if upon entry to either state the next n bits are not all ones, n is decreased by one according to the rule mentioned in the preceding section.

```
        .-----------.                     .-----------.
  .-----|           |                     |           |-----.
  |     |     9     |                     |     6     |     |
  |   .-|           |<--.             .-->|           |-.   |
  |   | \-----------'    \           /    \-----------' |   |
 1|  0|                   \         /                   |1  |0
  |   |    .->Error        \       /       Error<-.    |   |
  |   |   0|                \     /               |1   |   |
  |   | .-----------.        \   /        .-----------. |   |
  | 1 | |           |         \ /         |           | | 0 |
  | .---|     7     |          \          |    10     |---. |
  | | | |           |         / \         |           | | | |
  | | | \-----------'        /   \        \-----------' | | |
  | | |       A             /     \             A       | | |
  | | |       |            /       \            |       | | |
  | | |      1|           /         \           |0      | | |
  | | | .-----------.  0 /           \ 1  .-----------. | | |
  | | | |           |---'             \---|           | | | |
  | | | |     5     |                     |     8     | | | |
  | | | |           |                     |           | | | |
  | | | \-----------'                     \-----------' | | |
  | | |       A                                 A       | | |
  | | |       |                                 |       | | |
  | | |      1|                                 |0      | | |
  | | | .-----------.                     .-----------. | | |
  | | ->|           |                     |           |<- | |
  | |   |     1     |                     |     2     |   | |
  | |   |    B-W    |<-----.       .----->|    W-B    |   | |
  | |   \-----------'      |       |      \-----------'   | |
  | |      |     A         |       |         A     |      | |
  | |      |     |         |0     1|         |     |      | |
  | |      \-----'         |       |         \-----'      | |
  | |         0          .-----------.          0         | |
  | |                    |           |                    | |
  | |                    |     4     |                    | |
  | |        RUN         |           |         RUN        | |
  | |      .-----.       \-----------'       .-----.      | |
  | |      |     |         A       A         |     |      | |
  | |      |     V         |       |         V     |      | |
  | |   .-----------.   1  |       |  1   .-----------.   | |
  | \-->|           |------'   0   \------|           |<--' |
  |     |     3     |<--------------------|     0     |     |
  \---->|    B-B    |-------------------->|    W-W    |<----'
        \-----------'          0          \-----------'
```

Figure 3. DFSA Model of Encoding

2.2. Formatting Considerations

Data are encoded for transmission by the Dacom 450 in 585-bit frames, consisting of a 24-bit synchronization code, 37-bit leader, 512-bit information area and l2-bit checksum. There are two kinds of frames distinguished by leader format, one for setup or initialization and the other for the data itself. Serial binary image data are placed in the data area of succeeding data frames.

The header of each frame is shown in Figure 4. The various fields are defined in Table 1 following the Figure.

```
   +-----------+--------+-------------------+----------+
   | Sync Code | Leader |        Data       | CRC Code |
   +-----------+--------+-------------------+----------+
        24    /    37    \       512             12
     .-------'            \----------------------.
    /                                             \
   +-------+-------+-------+-------+-------+-------+
   | Flags | Count | X Pos | Black | White | State |
   +-------+-------+-------+-------+-------+-------+
   |   7    \ 10      12       3       3       2
   |         \--------------------------.
   |                                     \
   +-----+-----+------+-----+-------+-----+
   | Seq | RUN | COFB | RPT | Spare | SUB |
   +-----+-----+------+-----+-------+-----+
      2     1     1      1      1      1
```

Figure 4. Frame Format

Table 1. Header Field Definitions

Field	Width (bits)	Function	Setup Block	Data Block
Sync Code	24	Synchronization	30474730 (octal)	
Seq	2	Sequence number	00	00,01,10,11
RUN	1	Initialize-start	0	1
COFB	1	Unknown	0	0
RPT	1	Unknown	1	0
Spare	1	Unknown	0	0
SUB	1	Indicates setup frame	1	0
Count	10	Number of bits in data field (0 - 512)	All 1's	
X Pos	12	Current position on scan line (0 - 1725)	All 1's	
Black	3	Current black field length (2 - 7)	All 1's	
White	3	Current white field length (2 - 7)	All 1's	
State	2	Current state (0 - 3)	All 1's	
Data	512	Data (0 - 512 bits)		
CRC Code	12	CRC checksum. Uses polynomial x**12 + x**8 + x**7 + x**5 + x**3 + 1		

Setup frames have additional information in the data field; the various fields and their functions are described in Table 2.

Table 2. Field Definitions for Setup Frame.

Field	Width	Function
Start bit	1	Always zero
Speed bit	1	Set if express mode
Detail bit	1	Set if detail mode (speed and detail bits both zero for quality mode)
14 inch	1	Set if 14-inch paper

5 inch	1	Set if 5-inch inch paper (14-inch and 5-inch inch paper bits both zero for 11-inch paper)
Paper present	1	Set if paper present in scanner
Spare	5	Can have any value
Multi-page	1	Set if multi-page mode
	20	All 0’s
	480	Alternate 1’s and 0’s

The tailing setup frames differ from the leading setup frames only in bits which indicate whether the system is operating in single or multiple page mode and whether paper is present in the scanner.

All n-bit numeric fields (except Seq) are transmitted by the Dacom 450 machine least-significant-bit (LSB) first (i.e. Count, X Pos, Black, White, State, CRC, and run length words in the data field). All other fields are transmitted left-most bit first.

There are a few important points to be considered in regard to the header of a data frame. The header contains enough information about the state of the decoding algorithm to be able to re-establish correct decoding in the event of loss or mutilation of a data frame. The decoding algorithm resets its state variables to those in the header each time it begins decoding a new data frame. One of the most difficult problems encountered while constructing the decoding algorithm was the correct synchronization of the algorithm as it proceeds across the frame boundary with respect to the header information. In order for synchronization to be maintained, the operation of the algorithm must

follow exactly that described in the previous section.

This requirement for every data frame to be self-synchronizing, leads to a few subtleties in the encoding algorithm which seem quite natural, but were not very obvious in the beginning.

1. Transition bits(s) labeling the arcs on the state transition diagram in Figure 2 are not broken across frames. Similarly, individual run-length words are not broken across frames.

2. If a frame ends with a transition, the header of the next frame contains the state to which the transition is made.

3. If a frame ends with a transition out of state 0 or 3, then the transition bit (0 or 1) is inserted at the end of the current frame (not at the beginning of the next frame).

4. The field lengths for black and white runs in the header include changes that may have been caused at the end of the previous frame.

5. If a frame begins with a white or black run, then this run is treated (for purpose of decreasing its field length) as if it were

the beginning of a new run, since there is no information in the header to indicate otherwise.

The decoding algorithm is initialized at the first data frame received after the sequence of setup frames at the beginning of transmission. The first data frame has a count of zero, indicating no data bits are in the frame. The second data frame begins the actual document; however, its X position appears to be irrelevant. Instead, we assume the initial X position at this time is one pel to the left of the right margin (-l mod l726). With these assumptions succeeding X positions of the algorithm and the frame headers agree.

2.3. The Decoding Program

The decoding algorithm described above has been implemented in the PDP-11 MACRO-11 assembly language for the RT-11 operating system. This program contains extensive features for selectively dumping frames and tracing the operation of the algorithm. It is designed to operate on a file containing the raw data generated by the machine and does not depend upon any prior reformatting of the data. However, it will operate also on files in the so-called UCL format [4], which has been adopted as the standard for use in the Internet Program. The existing DCNET supporting software for the Dacom 450 uses the UCL format and operates normally to copy data directly between the machine and the file, with decoding operations done at a later time. However, there is no intrinsic factor, except processing-rate limitations, why input data could not be decoded directly from the machine.

In operation, the program scans the input data one bit at a time and searches for the synchronization pattern. Note that all data processed are inverted from the natural interface conventions. When a

synchronization pattern is found, the header and data portions are extracted and the various state variable checked and reset, if necessary. Checksum verification is performed according to the polynomial 1 + x**3 + x**5 + x**7 + x**8 + x**12. In the case of setup frames the format (detail, quality, express), page length (14, 8-l/2, 5-l/4) and multiple-page indicators are extracted from the data area. Finally, under control of specified options, the header and data portions of the frame are printed with appropriate headings.

The decoding algorithm itself is called for each data frame. It produces an output consisting of a sequence of run-length pairs which can be used to form bit maps and other representations of the data. Optionally, a printed trace of the operations performed by the algorithm can be produced.

2.4. The Encoding Program

The encoding algorithm has been implemented in the PDP-11 MACRO-11 assembly language for the RT-11 operating system. The program accepts facsimile data in 16-bit run-length format or bit-map format. The input data would normally be in a file, possibly obtained by translating some other representation (e.g., T.4 format) to run-length or bit-map format. The program produces an output consisting of data compressed in Dacom 450 format and packed in 585-bit frames along with the corresponding header and checksum information.

The encoding program needs to be careful about how to break data across frames and how many bits of data to insert in each frame. The rules mentioned in section 2.2. help to solve the first problem. The second problem is a little less understood. The problem arises because data bits are required by the printing mechanism at a constant rate, but successive frames transmitted at the line rate can contain different amounts of decoded information, leading to buffer overrun in extreme cases.

In order to compensate for the rate mismatch, it has been found sufficient to control the size of the data portion of the frame according to a simple set of empirical rules which produce results quite similar to the scanner iteslf. According to these rules, a frame is "full" when it contains more than 500 bits of data or when the data represents more than 4800*X pels (or columns) of information,

```
where   X = 2 for transmission rate 2.4 kbs,
        X = 1 for transmission rate 4.8 kbs,
        X = 1/2 for transmission rate 9.6 kbs.
```

2.5. Dacom 450 File Formats

Dacom 450 facsimile data is ordinarily stored as an RT-11 file in the so-called UCL format [4]. In this format, each 585-bit frame is stored in a 76-byte record. The first byte specifies the length of the record, the second specifies a command and the remaining 72 bytes contain the 585 bits of the original Dacom 450 frame zero-filled at the

end. The command byte is coded as follows:

a. 56 (70 octal): The data field contains a setup frame (the first record of the file). The length byte is 76 (114 octal).

b. 57 (71 octal): The data field contains a data frame (the remaining records in the file except the last one). The length byte is 76 (114 octal).

c. 58 (72 octal): End of file (the last frame of the file). There is no data field and the length byte is 2.

2.6. Run-Length and Bit-Map File Formats

The decode program produces 16-bit run length words as its output. Each run is encoded in a 16-bit word, with white in positive and black in negative two’s complement values. A zero word terminates each line, with the trailing white run suppressed if present. An all-white line is encoded as a single run of length one followed by a zero word. The file is terminated by a line of length zero, that is, a single zero word.

Bit-map files consist of a four-byte header followed by the data. The header consists of two 2-byte quantities, the first of which represents the number of pels in a line and the second the number of lines in the page. Each scanning line of data is represented in an integral number of bytes, the last byte of a line zero-filled if necessary.

3. Dacom 500 Data Compression Principles

The Dacom 500 machines are high-speed versions of the Dacom 450 machines and operate in the 50-Kbps range using the T.4 compression algorithm. This algorithm, described in the [5], is a one-dimensional one, rather than the two-dimensional one used in the Dacom 450 and described in previous sections. Since this algorithm is well known and the subject of an international standard, it will not be further discussed here.

3.1. Dacom 500 Decoding Algorithm

The decoding program has been implemented in the PDP-11 MACRO-11 assembly language for the DCNET and RT-11 operating systems. It operates on a file containing facsimile data encoded using the T.4 algorithm and produces a file in bit-map format.

The decoding program scans the input data bit-by-bit and recognizes sequences of bits which form valid run-length codes (see the tables in [5]). The table of Huffman codes can be represented as a binary tree with the values of the run lengths (e.g. 1, 2, 64, 1728, etc.) at the terminal nodes and each branch labeled 0 or 1. The code for any run length then is the sequence of branch labels on the path from the root to the terminal node representing this length.

The tables for black and white run-length codes are stored as separate binary trees in the decoding program. The decoding algorithm starts by initializing an accumulator at zero. It then begins at the root of the corresponding tree and traverses the tree as it consumes bits one-by-one from the input. When a terminal node is reached, the value stored at that node is added to the accumulator. If a make-up node is reached, the value at that node is added to the accumulator and the search is resumed with the same tree to obtain the terminating value; otherwise, the accumulator represents the current run length and the search resumes with the alternate tree.

3.2. Dacom 500 Encoding Program

The encoding program is also implemented in the PDP-11 MACRO-11 assembly language for the DCNET and RT-11 operating systems. It scans the bit-map input and encodes each run of black or white pels by a simple table lookup of the Huffman codes. It operates on a file containing facsimile data in bit-map format and produces a file in T.4 format. The T.4 specifications [5] require a minimum transmission time per scan line of 4.3 milliseconds, which at 50-Kbps corresponds to 242 bits (DATA bits plus any required FILL bits plus the EOL bits equal 242 bits minimum).

3.3. Dacom 500 File Formats

The file consists of a number of 512-byte blocks, the first of which is the header. The header contains a list of two-byte entries, the first of which contains the number of pages and the remaining the lengths (in blocks) of each page in turn. The remaining blocks of the file contain the data for each page in T.4 format. The data for each page is preceded by a page-setup command and succeeded by a page-end-of-record command, as transmitted by the Dacom 500. The format of both commands consists of the 12-bit T.4 EOL code (000000000001) repeated six times and followed by a special 4-bit code word also repeated six times. The special code word consists of bits B1 through B4 as defined below.

```
B1: VERTICAL RESOLUTION
    0 = 7.7 lines per millimeter
    1 = future option, not implemented

B2: OUTPUT PAPER LENGTH
    0 = short length (Letter size)
    1 = long length (Legal size)

B3: DOCUMENT IN SCANNER
    0 = no document present (end of page)
    1 = document present (beginning of page)

B4: ODD PARITY OVER B1-B4
```

3.4. Comparison of Facsimile Formats and Transcoding Speeds

Four different file formats are presently used in our system for facsimile data storage: Dacom 450, Dacom 500 (T.4), 16-bit run-length and bit-map. The sizes of typical files (in megabits) in these formats are shown below for comparison.

File	Dacom 450	Dacom 500	16-bit run-length
PNGUIN	0.22	0.5	0.27
INTELP	0.62	0.77	3.3
PANDA	1.02	2.03	6.41

The file called PNGUIN is a block drawing of dancing penguins and represents a "small" file. The file called INTELP is a composite INTELPOST test image with text and graphics and represents a "medium" file. Finally, the file called PANDA is a half-tone newspaper photograph of a giant panda and represents a "monster" file (this file was recorded on the Dacom 450 in quality mode and is therefore about half the size it would be in detail mode). The size of the bit-map file for all these images is 3.8 megabits.

Figure 5 shows the file sizes (in 512-byte blocks) and transcoding times (in seconds) for the INTELPOST test page. The file sizes are indicated on the boxes, while the transcoding times are indicated on the arrows. All times were obtained on the LSI-11/23 processor.

```
         193                        925
    +-----------+     95     +-----------+
    |           |----------->|           |
    |    T.4    |            |  Bit-map  |
    |           |<-----------|           |
    +-----------+    165     +-----------+
                                 A    |
                      60     |    |
          .----------------------'    |110
          |                           |
          |                           V
    +-----------+     89     +-----------+
    |           |----------->|           |
    |Run-length |            | Dacom 450 |
    |           |<-----------|           |
    +-----------+    153     +-----------+
         413                      155
```

Figure 5. File Sizes and Transcoding Times

4. References

1. Weber, D.R. An adaptive run-length encoding algorithm. ICC-75, IEEE, San Francisco, California, June 1975.

2. Palmer, L.C. Final Report, COMSAT Participation in the DARPA Packet Satellite Internetworking and Speech Applications Program. COMSAT Laboratories, July 1980.

3. Katz, A. Decoding Facsimile Data from the Rapicom 450. DARPA Network Working Group Report RFC-798, USC/Information Sciences Institute, September 1981.

4. Postel, J. Rapicom 450 Facsimile File Formats. DARPA Network Working Group Report RFC-769, USC/Information Sciences Institute, September 1980.

5. Draft Recommendation T.4 - Standardization of Group 3 Facsimile for Document Transmission. CCITT Study Group XIV Contribution #25-E, December 1977. (Also in RFC-804).