DATA COMPRESSION AND COMPUTATION SPEED

Catherine R.W. Duff

Internal Report 86-17

Geological Survey of Canada

Energy, Mines and Resources

Ottawa, April, 1986

Geophysics Division

Data compression and Computation Speed

Introduction

It has been proposed that Yellowknife Array data should be compressed so as to take up less digital storage space. This might be useful for satellite transmission and/or for storage of continuous data on optical disk. A proposed method involves taking the differences between consecutive data points, (since the differences are usually small numbers), and storing most of them as 8-bit numbers. Where the differences are too large, 2-byte or 4-byte numbers would be used. For this method to work efficiently, most of the differences must be small enough to fit into 8-bit numbers. This report investigates what percentage of differences of the YKA data are that small. For comparison, ECTN, GAC, RSTN, SRO and DWWSSN data are also examined. A set of computation-speed experiments is carried out on the μVAX to test how much time would be required to translate into and out of compressed format and to beamform.

Data Samples

For this study it was desirable to obtain a random sample of YKA data. This is difficult, because data are saved only when triggers occur. If the trigger is on noise, that noise is likely to be higher than average; if the trigger is on a signal, the noise preceding the signal may tend to be lower than average. This problem was solved in the following way: files with large amplitudes were chosen from the trigger list. These were taken two from each month of the 13-month period Jan. 1985 to Jan. 1986, to obtain a good sampling of various seasons. All but one of these were seismic events with high amplitude impulsive onsets. The one with emergent onset was discarded, and the noise preceding the onset was taken in each of the others. This set of 25 noise samples, about 16 seconds long each, should be a good random sample of noise.

A random sample of triggered files (whether the trigger was on noise or events) was also desired, to get a sample of the larger amplitudes which occur. For this purpose, The first triggered file was taken from one tape in each month the 13-month period.

For ECTN and GAC, five triggered files were chosen. Several of the stations were very spikey; transmission problems had garbled the data and produced very large numbers. The data were separated into spikey and non-spikey stations for each file (the spikey stations were not always the same between files). For GAC, both vertical and horizontal short period data are used.

The RSTN, SRO and DWWSSN data are the stations which triggered on the Dec. 23, 1985 Nahanni event. For three stations, it was not obvious whether the first arrival was before the beginning of the file. These stations were not used. Four stations which triggered on the event remained in each network. The short period vertical noise between the beginning of each file and the first arrival was used. The noise samples thus obtained were 11-90 seconds long.

Distribution of Differences

The distribution of sizes of differences for various stations and networks are shown in Figures 1 to 7. In these histograms, the x-axis represents a function f of the differences Δ_i between consecutive data points $(x_i - x_{i-1})$ in the seismograms, where

$$\begin{aligned} &f(\Delta_i) = 0 && \text{if } \Delta_i = 0 \\ &log_2 |\Delta_i| < f(\Delta_i) \leq log_2 |\Delta_i| + 1 && \text{if } \Delta_i > 0, \\ &\text{and } f(\Delta_i) \text{ is always an integer.} \end{aligned}$$

Thus, for example, $f = 3$ represents $\Delta_i = \pm 4, 5, 6$, or 7. f can be thought of approximately as the number of bits required to represent the number Δ_i. There are 256 possible combinations of 1's and 0's which fit into 1 byte (8 bits). Of these, 255 can be used to represent the numbers from -127 to +127; the remaining combination can be a flag to signal that the next few bytes will contain a number in expanded format. All the numbers with f from 0 to 7 inclusive (Δ_i from -127 to +127) can be represented in one byte, with one code left over to signal that a larger number is to be represented in the following bytes. On each plot is marked the percentage of data points for which $f \geq 8$, i.e. the percentage which must be represented by more than 1 byte. The y-axis is the number of occurrences of differences with the given $f(\Delta_i)$; the absolute size of these numbers is dependent on the number of data samples processed, and can be ignored, but the relative heights of the bars on each plot gives the distribution of the differences.

Table I gives the percent of samples with $f \geq 8$ (i.e. the percent of samples requiring more than 1 byte of storage) for the various stations and networks examined.

Evaluation of Storage Efficiency

For the YKA noise, a very low percentage (.005%) of samples require more than 1 byte. For the triggered files, the percentage is much higher (3.25%) but

still small. Let us do a rough calculation to see how efficient it would be to store YKA data in compressed form.

Suppose there are 20 triggers per day, with the triggered files 2 minutes long, and suppose that outside these files the data has the characeristics of the random noise sample in this study (only .005% of data requiring more than 1 byte), and inside those triggered files it is like the triggered files studied here (3.25% requiring more than 1 byte).

$$(40 \text{ minutes/day})\ (3.25\%) + (23\text{h. } 20\text{min})\ (.005\%) = .095\%$$

So, for .095% of the data, an extra 4 bytes are required. The average number of bytes per sample is therefore $1+4\times.095\% \simeq 1.004$ bytes. This is very nearly as good (from the point of view of storage space) as if all the samples took only 1 byte each. Even if 8 bytes were used for the larger numbers, the storage would still average only 1.008 bytes per sample. Hardly any savings would be achieved by using only 2 extra bytes for the large numbers: the average would be 1.002 bytes per sample. In conclusion, under these conditions, YKA data could be compressed to very nearly half of its former storage requirements.

For ECTN, the non-spikey stations are about as good as YKA, looking just at the number of samples requiring extra storage space, though the sampling rates, instrument response, and distribution of the small numbers are different. However, with the spikey stations, about 5% of the differences are numbers too large for 1 byte. This suggests that storage efficiency would not be as good if data with spikes and transmission problems are compressed. The spikey stations have a very different distribution from the non-spikey: a preponderance of zeroes and of very large differences. Assuming that the real, uncorrupted data are similar to the good stations, this suggests that most or all of the data from those stations was corrupted, since merely the occasional spike would not change the distribution much. Excellent savings could still be made, however. Even if the spikey condition held continuously at all stations, the data would only average 1.2 bytes per sample (assuming 4 extra bytes for large numbers) or 1.1 bytes per sample (assuming 2 extra bytes for large numbers).

GAC has different instrument response from ECTN and 30 sps sampling rate compared to ECTN's 60 sps. It is therefore not surprising that its distribution of differences differs from that of the ECTN stations.

It appears that data compression might be feasible for ECTN and GAC, but no definite conclusions can be made from the small data sample used here.

For RSTN and DWWSSN data, compression would be feasible.

For SRO, the proposed data compression scheme would be a dismal failure. There are very few small differences which could fit into 1 byte. This is because

of the digitization scheme used by that network, which represents a very tiny amount of ground motion in one digital count.

The percentage requiring more than 1 byte per sample depends not only on the ground motion, but also on the digitization level, that is, the amount of ground motion represented by one digital count. With the present instrument response, digitization level and 20 sps sampling rate, YKA short period data could be compressed to take only very slightly more than half its present storage requirement: that is, it could take slightly more than 1 byte per sample on average.

Computation Speed on the MicroVAX: Data Compression

In order to test how much computation time would be required for a compressed data format, two subroutines were written to translate into and out of a compressed format. A description of the compressed format, and a copy of the program are in the Appendix. A YKA short period data file of 2432 samples in each of 18 channels (about 122 seconds of real-time data) was read into the μVAX and translated back and forth 100 times. It took 10 seconds to read in the data, 78 seconds to translate forwards 100 times, and 93 seconds to translate back 100 times. From this we can conclude that it took .93 seconds to translate all the data back once. (That is 0.76% of real time.) Since reading the data off disk took about 10 times as long as translating, it is reasonable to guess that it might take less time to read in compressed data and translate than to read in full format data. However, since the compressed data varies in length, it may take more time searching for the right block of data. The computation time for forward translation was 0.64% of real time. Therefore compression in real time of all the data would take only a small part of the CPU time of a μVAX.

Computation Speed on the MicroVAX: Beamforming

In order to test the time it would take to beamform on the μVAX, a program was written which forms all 121 beams from 18 substations. It was tested on the same file (2432 samples per channel) used in the previous section, and took 108 seconds to form all beams 3 times. That is 29.5% of real time. That is a large part of the CPU time of a μVAX to dedicate full-time. It may be possible to write a more efficient beam-forming program. The program used here has several addition operations to manipulate array indices for every addition operation which is used directly in the beam-forming process.

Comments on Data Compression

If a data compression scheme is considered, we must look at the following questions.

(1) How much data would become indecypherable if a few bytes were somehow lost?

(2) Would the rules for translating back ever become lost or forgotten?

(3) Would there be only a few people who would know how to access the data?

(4) Would newer computers or compilers be unable to support the required byte manipulation at a reasonable speed?

(5) How much time would be spent writing programs to translate one way or the other? Would the translation program need to be rewritten for various different computers and compilers? It took me about 3/4 of a day to write, but not fully test, a FORTRAN program for the μVAX to translate into and out of compressed format. One for actual use would be much more complex, involving, for example, times inserted at regular intervals.

(6) Would translation be reasonably fast on all computers – even future ones which might not be designed to handle manipulation of single bytes efficiently?

(7) Since the compressed data varies in length, times must be written in more often than with full format data.

(8) In the worst case, during a large event, if all the numbers were very large, over 150% of the previous storage space would be required.

(9) In the worst case, if we didn't debug our programs properly, we would end up with an optical disk filled with a gigabyte of meaningless 1's and 0's.

I recommend that if some such compression scheme is used, then an ASCII file should be written in the beginning of every optical disk (or other storage medium) containing compressed data. This file should explain the compression scheme in English (and French?) and contain a copy of a program in FORTRAN for translating back. Other information, such as date, instrument response, etc., might also be written there.

Table I

	Percent of data samples with $f \geq 8$		Station or network
Figure 1	.005%	25 files	YKA, random sample of noise
Figure 2	3.25%	13 files	YKA, triggered files
Figure 3	5.25%	25 files	YKA, large events
Figure 4	0.01%	5 files	ECTN, non-spikey stations, triggered files
Figure 5	4.9%	5 files	ECTN, spikey stations, same files
Figure 6	0.22%	5 files	GAC, same files as ECTN
Figure 7	90%	4 stns.	RSTN, Dec. 23 /85 noise
Figure 8	99.4%	4 stns.	SRO, Dec. 23 /85 noise
Figure 9	26%	4 stns.	DWWSSN, Dec. 23 /85 noise

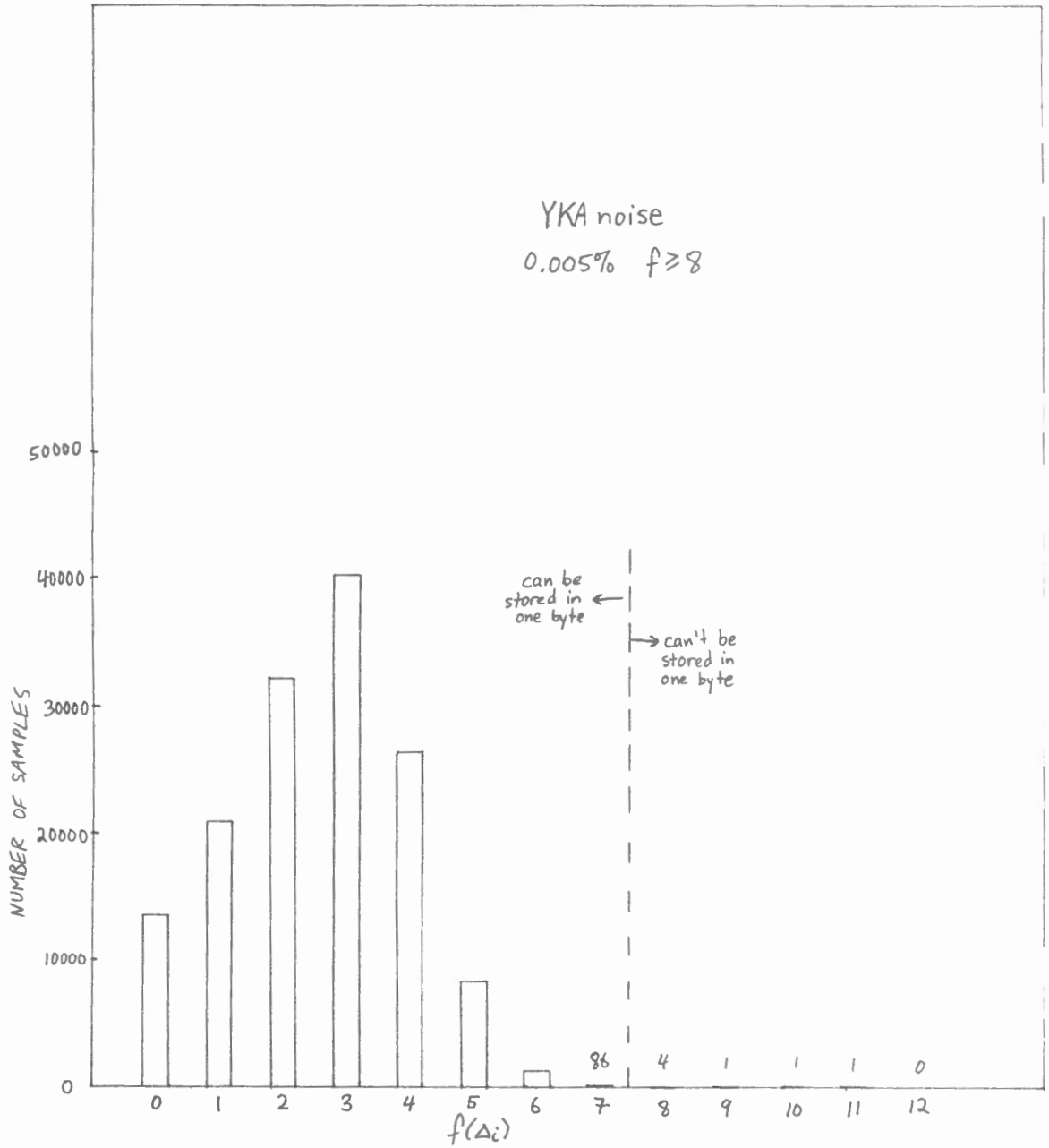


Fig. 1. Histogram of differences between consecutive data points, for random sample of noise at YKA. The function f is explained in the text. $f \geq 8$ for 0.005% of the data.

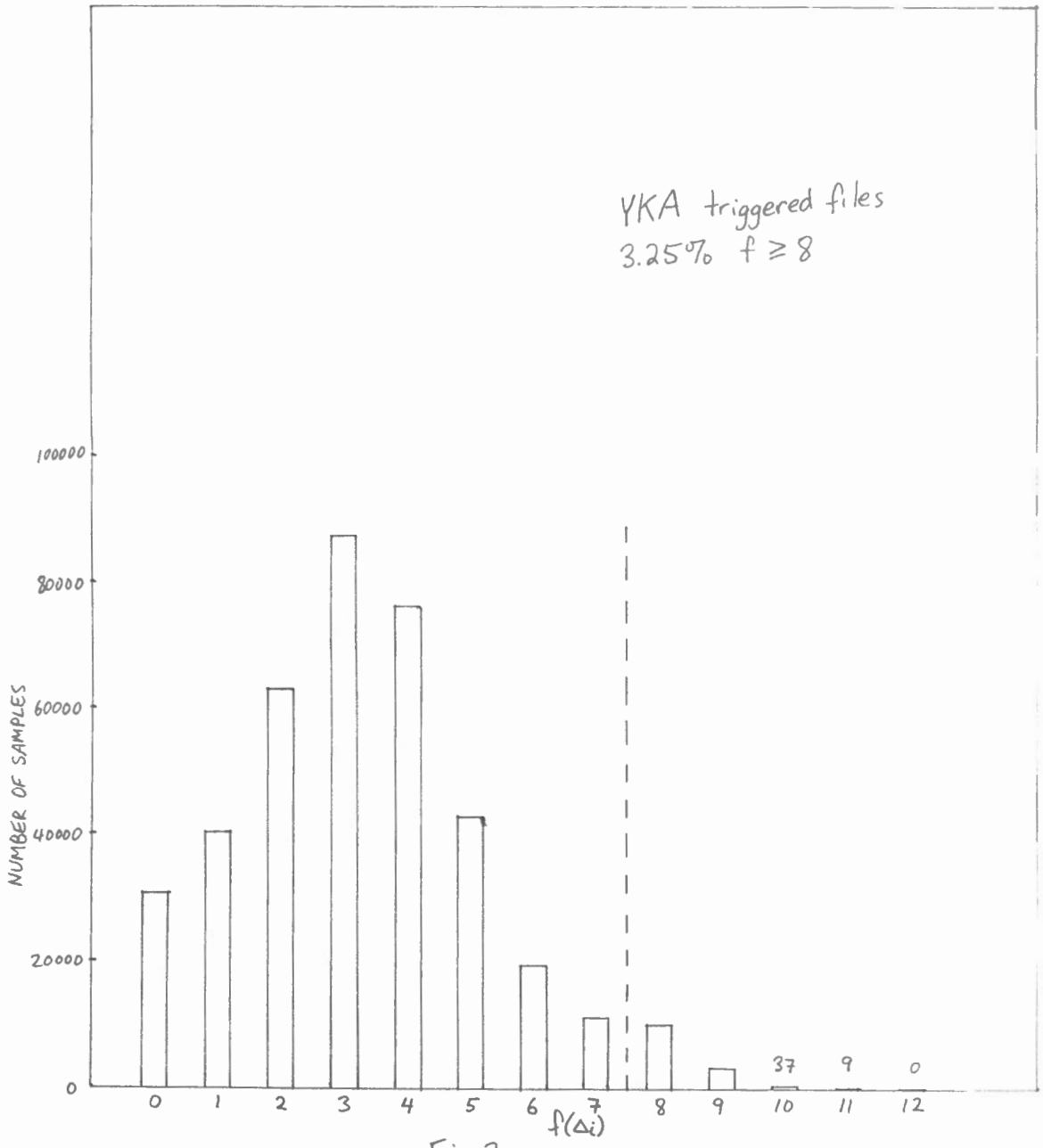

YKA triggered files
3.25% f ≥ 8
NUMBER OF SAMPLES
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9 10 11 12
f(Δi)
37
9
0

Fig. 2.

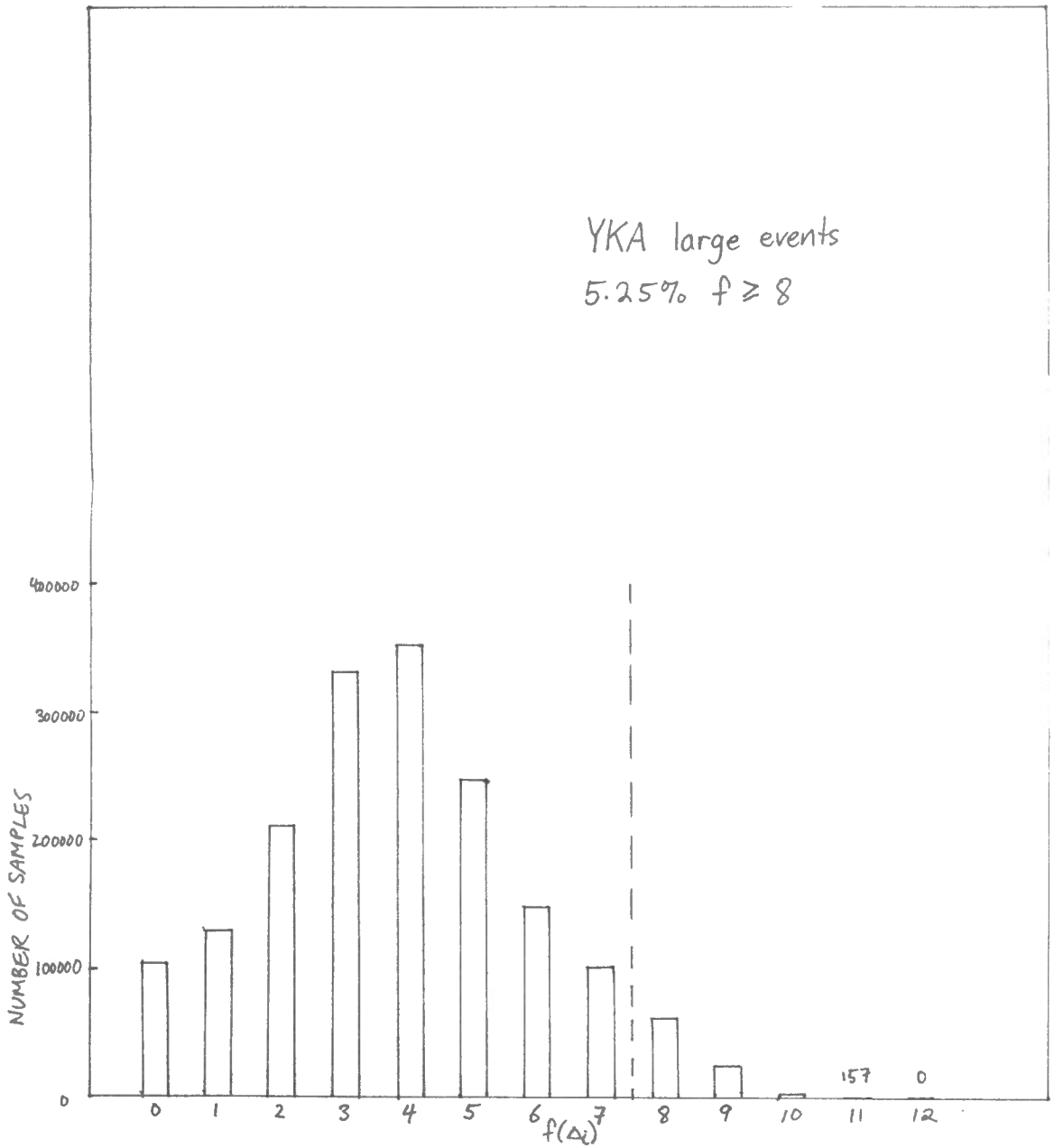

YKA large events
5.25% f ≥ 8
400000
300000
200000
100000
0
NUMBER OF SAMPLES
0
1
2
3
4
5
6
7
8
9
10
11
12
157
0
f(Δt)

Fig. 3.

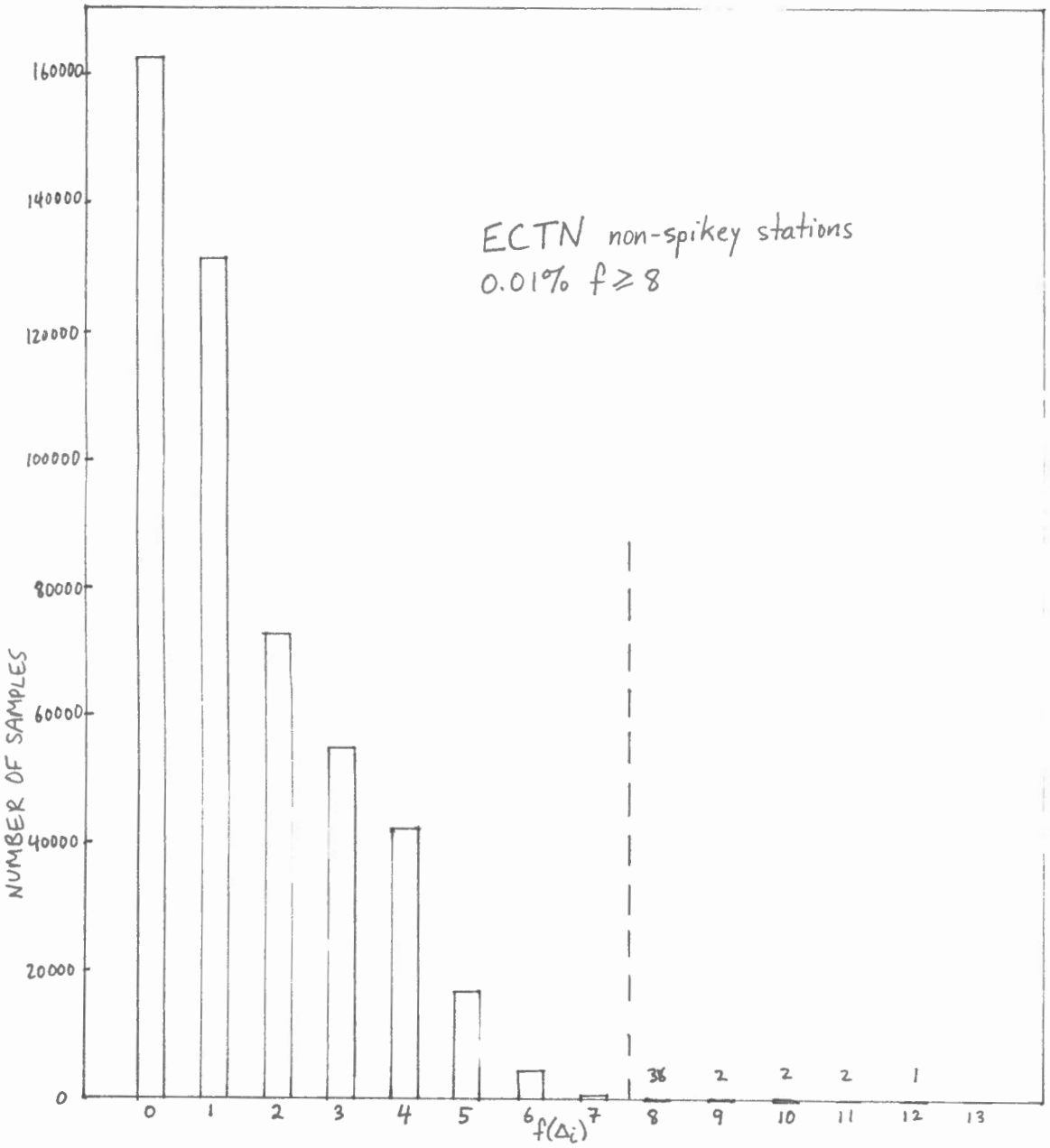

ECTN non-spikey stations
0.01% f ≥ 8
NUMBER OF SAMPLES
160000
140000
120000
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13
f(Δi)
38 2 2 2 1

Fig. 4.

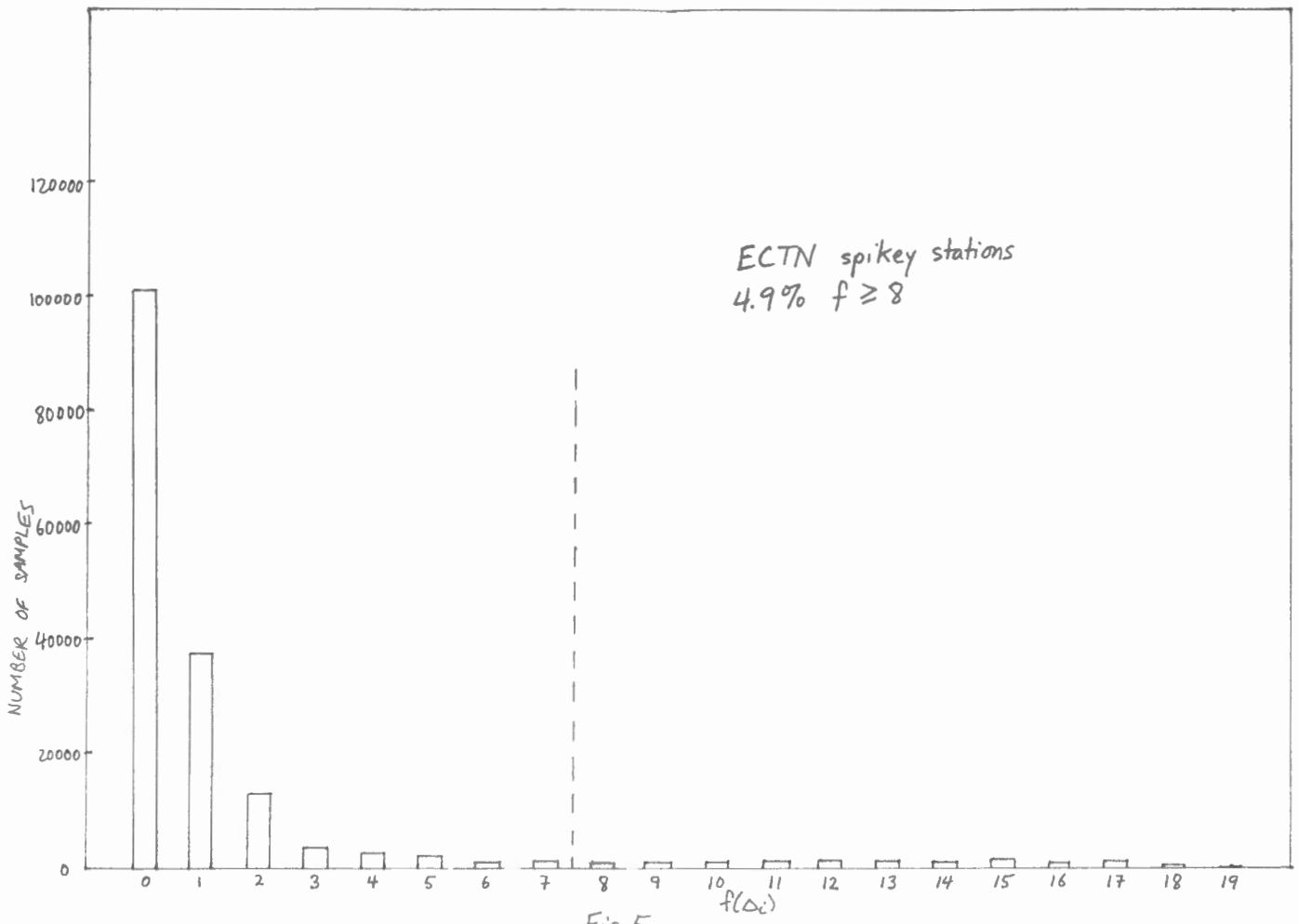

ECTN spikey stations
4.9% f ≥ 8
NUMBER OF SAMPLES
120000
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
f(Δt)

Fig. 5.

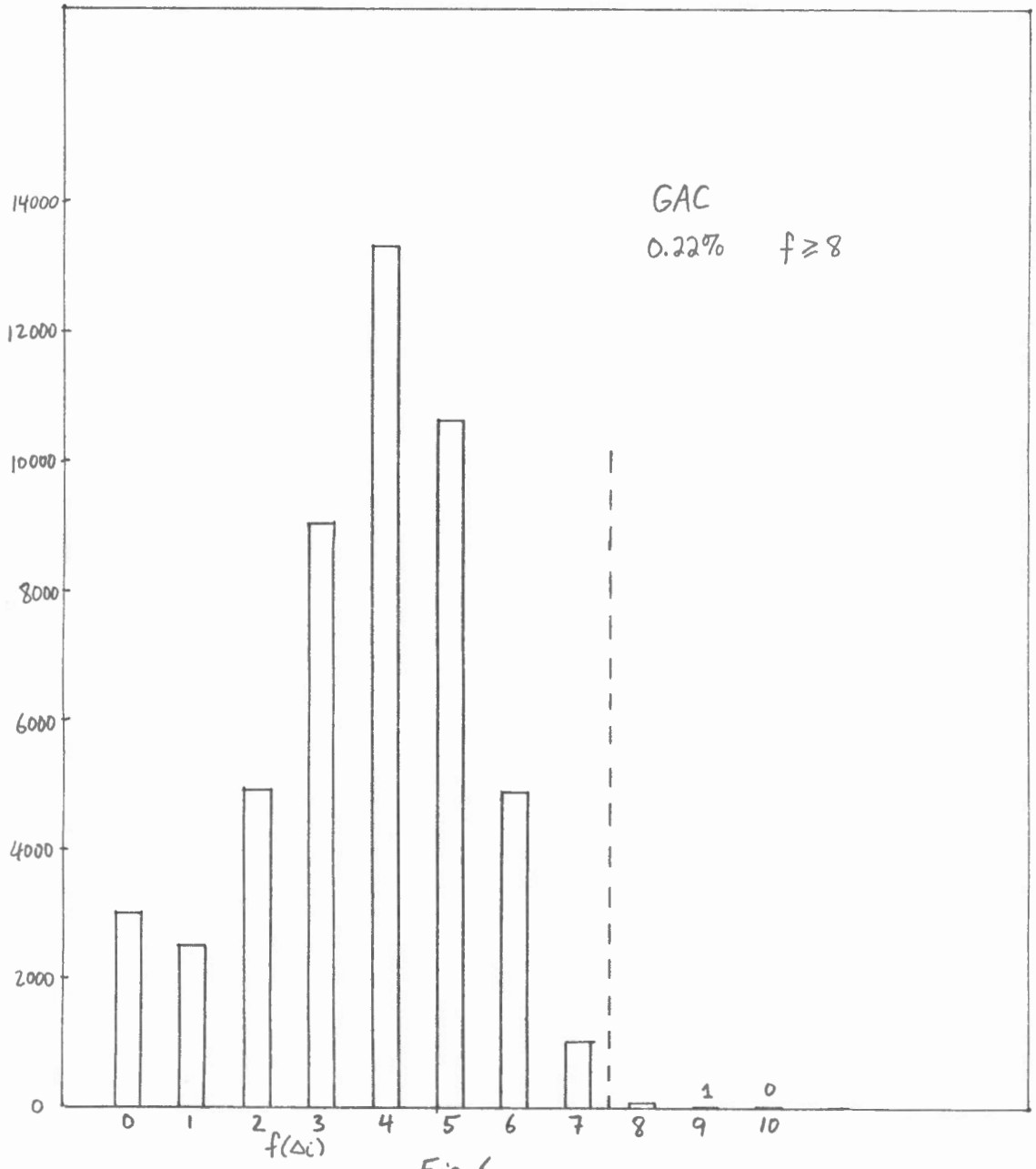

GAC
0.22%
f ≥ 8
14000
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
8
9
10
1
0
f(Δi)

Fig. 6.

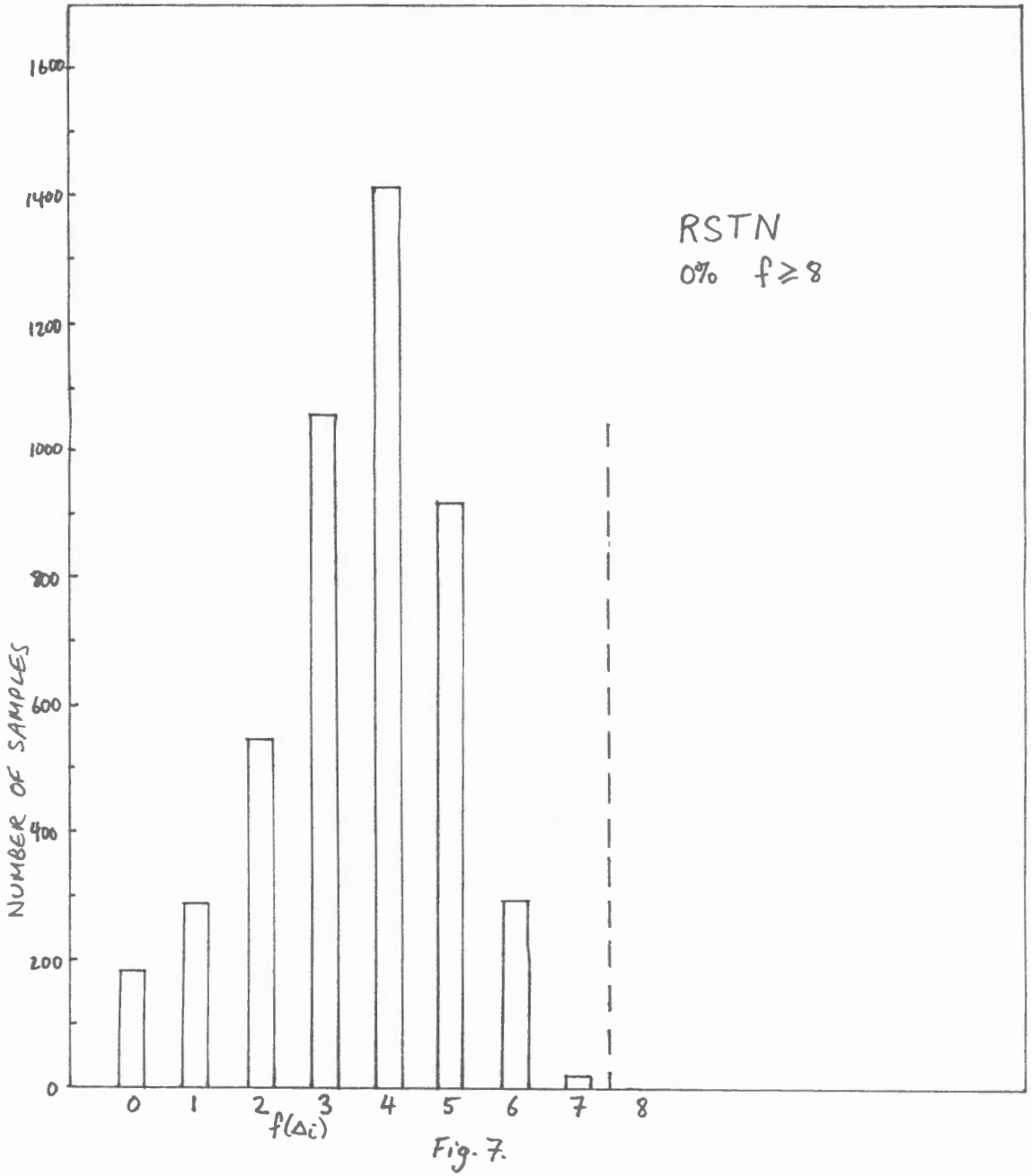

RSTN
0% f ≥ 8
NUMBER OF SAMPLES
1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7
8
f(Δi)

Fig. 7.

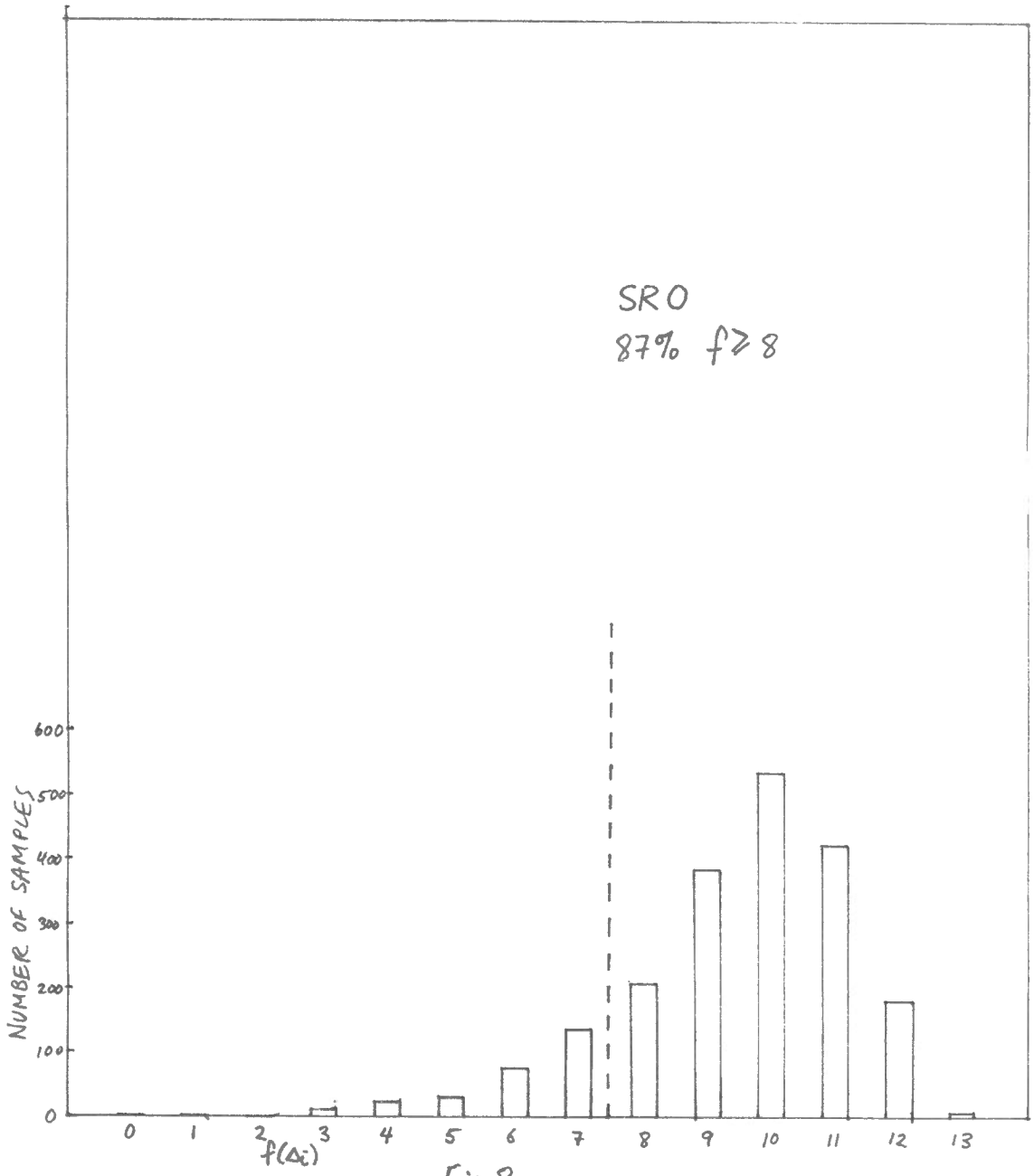

SRO
87% f≥8
NUMBER OF SAMPLES
600
500
400
300
200
100
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13
f(Δi)

Fig. 8.

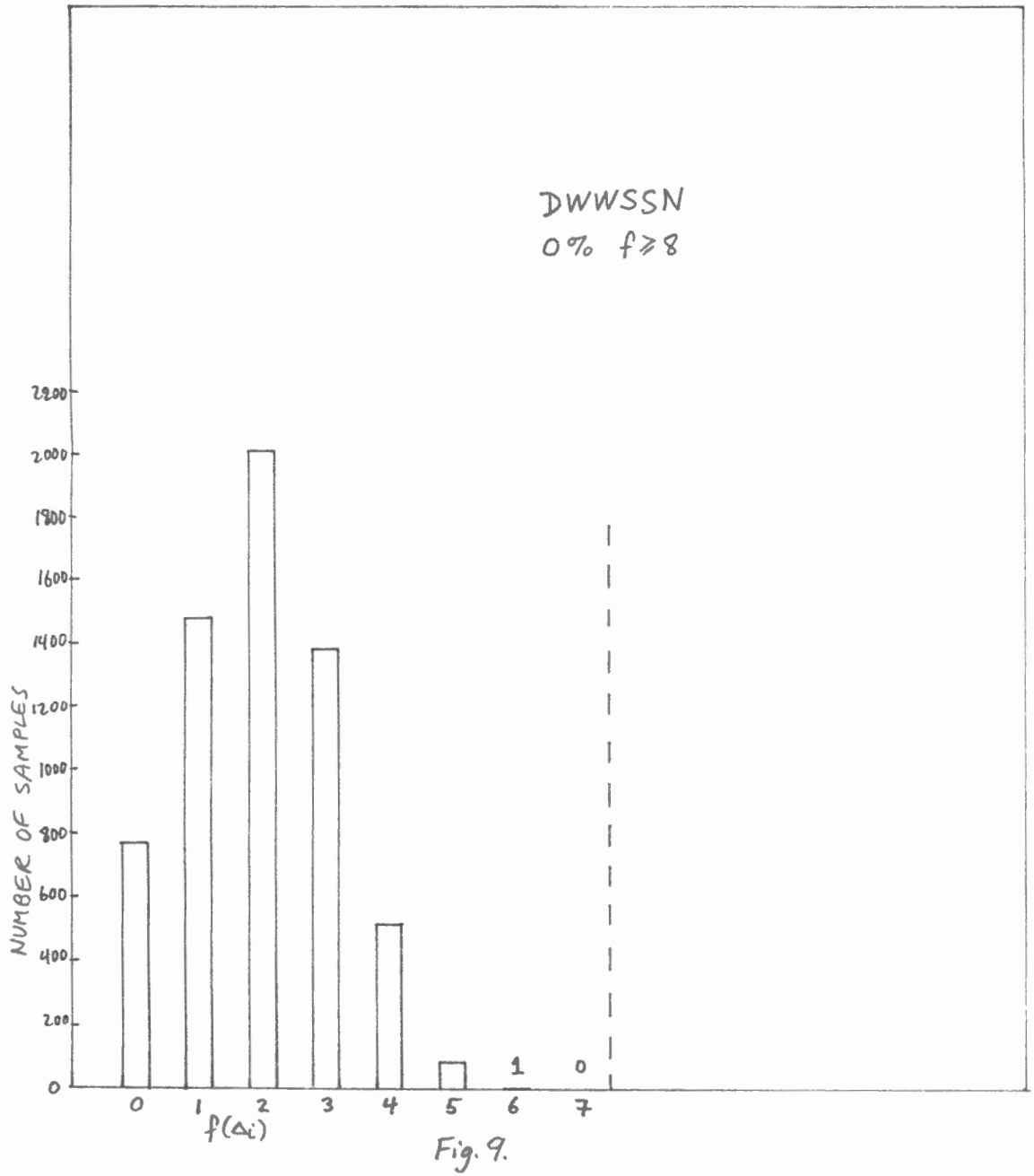

DWWSSN
0% f≥8
NUMBER OF SAMPLES
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7
1
0
f(Δi)

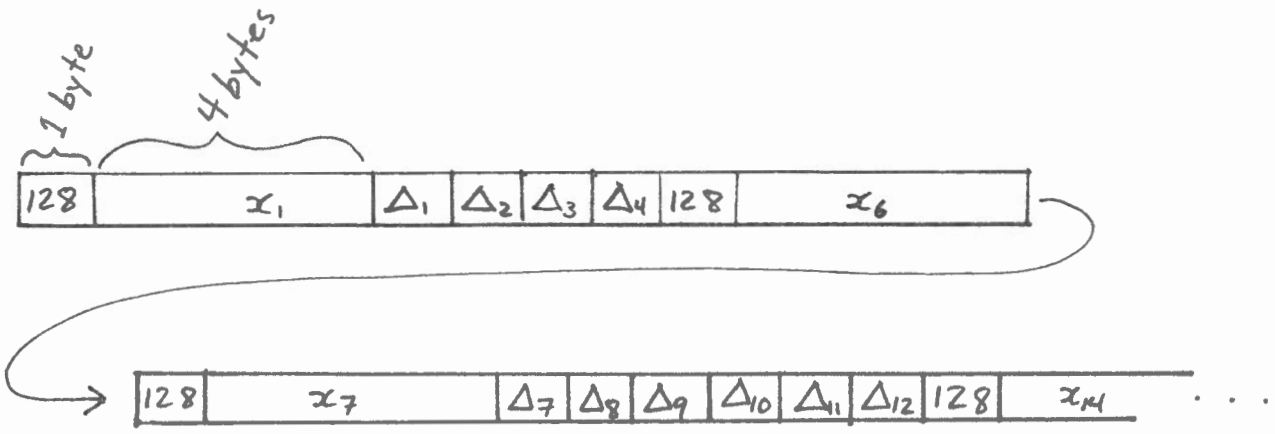
Fig. 9.

Appendix

Programs Trab and Beam, used in computation speed experiments on the μVAX, and a description of the data compression scheme used in the program TRAB.

Data Compression Scheme

The data compression scheme used in the program TRAB is as follows. The 1-byte number 128 (binary 10000000) is used as a flag to signal that the next four bytes contain a 4-byte integer which is an actual value of the data, not a difference between two values. If this flag does not occur, it is assumed that the data are stored as 1-byte integers which are differences between consecutive values. The first byte in the sequence of compressed data is always the flag 128, followed by the first value of the data, in 4-byte format. 1-byte differences Δ_i are recorded whenever $-127 \leq \Delta_i \leq +127$; otherwise the flag 128 and the 4-byte value are recorded.



$$\Delta_i = x_{i+1} - x_i$$

$$-127 \leq \Delta_i \leq 127$$

```
      PROGRAM TRAB
C
C     translates into compressed format and back.
C
      include 'tsf2.h'
      include 'otsf2.h'        !These two include files are used
C      by getind and getwvf, which read in SAM format files.
C
      character*80 filnam
      integer idata(3000,20) !full format data, 20 channels
      integer icdata(3000,20) !compressed data
      character*8 ctim         !time
      integer nic(20)          !number of bytes in compressed data
C
      call time(ctim)
      type '(1x,''The time is now  '',a8)',ctim
C
      type '('' input YKA filename:   '',$)'
      accept '(a80)',filnam
C
      call time(ctim)
      type '(1x,''The time is now(  '',a8)',ctim
C
      open(1,file=filnam,status='old',form='unformatted',
     +     access='direct',recl=512,readonly)
C
      call getind(1)
      idec = 1
      iunit = 1
      tlen = 500
      do icomp=1,18
C              get 18 substations
               call getwvf(idata(1,icomp),ns(icomp),icomp,start(icomp),
     +           tlen,idec,iunit,ierr)
C              now idata(ns) is time series (integer)
       end do
      nsamp = ns(1)
C
      call time(ctim)
      type '(1x,''The time is now  '',a8)',ctim
      type '('' (about to start trans)'')'
C
      do ii=1,100
      do icomp=1,18
C              ******* TRANSLATE INTO COMPRESSED FORMAT *******
               call trans(nsamp,idata(1,icomp),icdata(1,icomp),
     +           nic(icomp))
       end do
      end do
C
      call time(ctim)
      type '(1x,''The time is now  '',a8)',ctim
C
      do ii=1,100
      do icomp=1,18
C              ******* TRANSLATE BACK ******
               call transb(nsamp,idata(1,icomp),icdata(1,icomp),
     +           nic(icomp))
       end do
      end do
```

```
c
      call time(ctim)
      type '(1x,''The time is now  '',a8)',ctim
c
      i = i + 1
      stop
      end
c
c*****************************************************************
c
      subroutine trans(nsamp,iseis,icomp,ncomp)
c     compacts data iseis into icomp
c     nsamp is number os samples (input)
c     ncomp is number of bytes compressed data (output)
      integer*4 iseis(1)  !input
      byte icomp(1)   !output
      integer*4 idif
      byte ibdif(4)
      equivalence (ibdif(1),idif)
      common/tracm/icptr
c
c
      icptr = 1
c
c     first sample is recorded as 4 bytes
      call big(iseis,icomp,1)
c
      do i=2,nsamp
            idif = iseis(i) - iseis(i - 1)
            if (iabs(idif) .lt. 128) then
c                 one-byte difference
                  icomp(icptr) = ibdif(1)
                  icptr = icptr + 1
             else
c                 4-byte value
                  call big(iseis,icomp,i)
             end if
       end do
      ncomp = icptr - 1
      return
      end
c
      subroutine big(iseis,icomp,nn)
c     transfer one sample from iseis to icomp in
c     4-byte format, with '128' flag preceding it.
c
      integer*4 iseis(1)
      byte icomp(1)
      integer*4 i128
      byte ib128(4)
      equivalence (i128,ib128(1))
      integer*4 ibig
      byte ibbig(4)
      equivalence (ibig,ibbig(1))
      data i128 / 128/
      common/tracm/icptr
c
c     put in flag '128'
      icomp(icptr) = ib128(1)
c
```

```
c       transfer 4-byte number
        ibig = iseis(nn)
        do i=1,4
                icomp(icptr + i) = ibbig(i)
         end do
        icptr = icptr + 5
        return
        end
c
c*****************************************************************
c
        subroutine transb(nsamp,iseis,icomp,ncomp)
c       translates compressed data icomp back into
c       full format iseis
c
        integer*4 iseis(1)
        byte icomp(1)
        integer*4 idif
        byte ibdif(4)
        equivalence (ibdif(1),idif)
        common/tracm/icptr,isptr
        integer*4 i128
        byte ib128(4)
        equivalence (i128,ib128(1))
        data i128 / 128/
c
        icptr = 1
        isptr = 1
c
c       decode first sample ('128' + 4 bytes)
        call decbig(iseis,icomp)
c
        do while(icptr .le. ncomp)
                if(icomp(icptr) .ne. ib128(1)) then
c                       single byte difference
                        idif = icomp(icptr)
                        iseis(isptr) = iseis(isptr-1) + idif
                        isptr = isptr + 1
                        icptr = icptr + 1
                 else
c                       4-byte value
                        call decbig(iseis,icomp)
                 end if
         end do
        end
c
        subroutine decbig(iseis,icomp)
c       transfer 4-byte number from icomp to iseis
        integer*4 iseis(1)
        byte icomp(1)
        integer*4 isss
        byte ibsss(4)
        equivalence (isss,ibsss)
        common/tracm/icptr,isptr
        do i=1,4
                ibsss(i) = icomp(icptr + i)
         end do
        iseis(isptr) = isss
        isptr = isptr + 1
        icptr = icptr + 5

        return
        end
```

```
      program beam
c     reads in a yka file
c     forms 121 beams
c     does nothing with them
c     purpose:  check computation speed of MicroVAX
c
      include 'tsf2.h'
      include 'otsf2.h'
      character*80 filnam
      integer ioff(20,-5:5)
      integer ixbeam(6100,-5:5),iybeam(6100,-5:5),ibeam(6100,-5:5,-5:5)
      integer idata(6100,20)
      character*8 ctim
      common/bem/nsamp
c
      call time(ctim)
      type '(1x,a8)',ctim
c
c     initialize constant ioff, the delays for beam summing.
      do ich = 1,10
              do ixb = -5,5
                      ioff(ich,ixb) = - (ich - 5) * ixb
               end do
       end do
      do ich = 11,17
              do iyb = -5,5
                      ioff(ich,iyb) = - (ich - 18) * iyb
               end do
       end do
c     note:  these delays may not be correct
c
      type '('' input YKA filename:  '',$)'
      accept '(a80)',filnam
c
      call time(ctim)
      type '(1x,a8)',ctim
c
      open(1,file=filnam,status='old',form='unformatted',
     +     access='direct',recl=512,readonly)
c
      call getind(1)
      idec = 1
      iunit = 1
      tlen = 500
      do icomp=1,18
c             read in 18 substations
              call getwvf(idata(1,icomp),ns(icomp),icomp,start(icomp),
     +          tlen,idec,iunit,ierr)
c             now idata(ns) is time series (integer)
       end do
      nsamp = ns(1)   !assume all substations have same number
c                      of samples
      type '('' number of samples:  '',i10)',nsamp
c
c
      call time(ctim)
      type '('' about to start beamforming'')'
      type '(1x,a8)',ctim
c
c     useless extra loop to make it take longer
```

```
c
      call form(ioff,ixbeam,iybeam,ibeam,idata)
c
c     end useless extra loop
      end do
c
      call time(ctim)
      type '(1x,a8)',ctim
c
      stop
      end
c
c*********************************************************************
c
      subroutine form(ioff,ixbeam,iybeam,ibeam,idata)
      integer ioff(20,-5:5)
      integer ixbeam(6100,-5:5),iybeam(6100,-5:5),ibeam(6100,-5:5,-5:5)
      integer idata(6100,20)
      common/bem/nsamp
c
c     form N-S beam
      nbsamp = nsamp - 70     !number of data points in beams
      do ixb = -5,5
              do0 isamp = 1,nbsamp
c                     form beam leaving out 35 samples at each end
c                     for mere programming convenience
                      ixbeam(isamp,ixb) = 0
                      do ich = 1,10   !10 Blue stations
                              ixbeam(isamp,ixb) =
     +                          ixbeam(isamp,ixb) +
     +                          idata(isamp + 35 + ioff(ich,ixb),ich)
                       end do
                end do
       end do
c
c     form E-W beam
      do iyb = -5,5
              do isamp = 1,nbsamp
c                     form beam leaving out 35 samples at each end
c                     for mere programming convenience
                      iybeam(isamp,iyb) = 0
                      do ich = 11,18  !8 Red stations
                              iybeam(isamp,iyb) =
     +                          iybeam(isamp,iyb) +
     +                          idata(isamp + 35 + ioff(ich,iyb),ich)
2                      end do
                end do
       end do
c
c     form 121 beams from N-S and E-W beams
      do ixb = -5,5
              do iyb = -5,5
                      do isamp = 1,nbsamp
                          ibeam(isamp,ixb,iyb) = ixbeam(isamp,ixb) +
     +                          iybeam(isamp,iyb)
                       end do
                end do
       end do
c     beams have been formed
c
      return
      end
```