CHAPTER 5

RANDOM SELECTION II: STATISTICAL FRAMES

The notion of a frame derives from cinematography, where the word denotes a single exposure on a strip of film. In broader usage, a "frame" connotes a 'frozen' moment within an ongoing process. For examples: in speech synthesis, a "frame" might denote one of a sequence of momentary, static formants used to simulate a dipthong or glide; in artificial intelligence, a "frame" might denote a momentary state of knowledge within an ongoing process of learning.

In the present chapter, we presume that a composition is conceived as a sequence of statistical frames, each embracing a number of musical items. Our goal is to select an attribute for each item in such a manner that the collection of options exploited by the frame as a whole conforms as closely as possible to a prescribed distribution. Typically "items" might be notes or phrases. "Options" for notes might include ~~be~~ degrees of a scale, articulations, or specific registers; the prescribed distribution may work either to maintain these options in uniform balance or to emphasize certain options at the expense of others.

"Options" for phrases might include ~~be~~ transpositions of a scale from which degrees are selected for the constituent notes, ~~or they~~ also ~~might be~~ average values influencing local articulations or registers.

The only requirement for a statistical frame is that a fixed statistical distribution holds good for the entire frame. Sometimes the statistical frames will correspond to the musical architecture; other times framing and architecture may proceed independently. An example of dependence between framing and architecture would be when each phrase in a musical passage had its own distributions of durations, degrees, and registers. An example of independence between framing and architecture is serial procedure, which imposes a discrete uniform distribution of options. Here, the prohibition against repeating any item in a series before the whole series has been stated means that the number of distinct options (e.g. the twelve chromatic degrees) ~~items (usually twelve)~~ must exactly match the number of items ~~elements~~ in a frame. A phrase may embrace just a few notes in a series, or it may overlap several series.

Indeed, frames governing different musical attributes need not even proceed in synchronous. Continuing with the example of serial procedure, a composer might juxtapose a series of twelve chromatic degrees over a series of five articulations (not including rests), a series of seven registers, and a series of eleven durations. The resulting frames would only come into alignment every 4620 notes!

The least biased strategy for organizing a statistical frame is analogous to the following mechanism:

1. Assemble a collection of identical balls, one ball for each item in the frame.

2. For each option, calculate from the prescribed distribution how many times the option should appear in the frame, then mark a corresponding number of balls with a code for this option.

3. Place all the balls in an urn and mix them around.

4. For each item in the frame, draw one ball blindly from the urn. The code marked on the ball will indicate which option has been selected for the current item. Do <u>not</u> replace the ball in the urn.

It should be clear that options selected using this mechanism will conform exactly to a prescribed distribution. This result stands in strong contrast to the methods of direct random selection described in the Chapter 4 (note 1). In later chapters, we shall consider ways of rendering the process of organization sensitive to stylistic criteria so that, for example, it would assign the 'best' options to items possessing one or more desirable qualities.

5.1 GENERATING DETERMINATE STATISTICAL POOLS

The most direct way of implementing the above unbiased mechanism starts by generating a pool of options which conforms to the prescribed distribution. This pool ~~may then be~~ is next shuffled randomly (note 2) and then sampled using the procedures for sampling a row described in Chapter 3 (heading 3.4.1).

5.1.1 Transformations of Uniform Pools

We now investigate methods of tailoring a population so that it adheres both to a prescribed statistical distribution and to a prescribed number of items. A simple method for generating a statistical population is available if there exists some formula which will transform uniform samples into samples conforming to the desired distribution. In such cases, we need simply feed equally spaced samples from the uniform range through this transformation.

The library subroutine FILLX illustrates how this method might be implemented to generate statistical pools conforming to for John Myhill's generalization of the exponential distribution (heading 4.2.3.1). FILLX requires four arguments:

1. POOL - Pool of samples. POOL must be a real array whose dimension in the calling program is LENGTH.

2. AVG - Average magnitude of samples. AVG must be real.

3. PROPOR - Ratio of maximum to minimum sample. PROPOR must be real.

4. LENGTH - Number of samples in pool. LENGTH must be an integer.

Table 5-1 shows pools of samples generated by subroutine FILLX with PROPOR set consistently to 16. The parameter AVG is calculated so as to illustrate filling an interval of time 100 units long (for example, 100 sixteenth notes) with a varible number of rhythmic periods. Notice that the cumulative sum falls far short of 100.0 when LENGTH is small; the effectiveness of

Ex 5-1

```
subroutine FILLX(POOL,AVG,PROPOR,LENGTH)
real POOL(1)
if (AVG.le.0. .or. PROPOR.lt.1.0) then
  stop 'Bad argument to FILLX.'
else if (PROPOR.lt.1.01) then
  do (L=1,LENGTH)
    POOL(L) = AVG
  repeat
  return
else if (PROPOR.lt.1000.0) then
  R2 = PROPOR ** (-1.0/(PROPOR-1.0))
  R1 = R2 ** PROPOR
else
  R2 = 1.0
  R1 = 0.0
end if
Y = (R2-R1)/float(LENGTH)
X = R1 + Y/2.0
do (L=1,LENGTH)
  POOL(L) = -AVG * alog(X)
  X = X + Y
repeat
return
end
```

LENGTH	AVG	Pool	Sum	Max/min
1	100.00	81.73	81.73	1.00
2	50.00	69.96 22.59	92.55	3.09
4	25.00	47.53 26.66 15.45 7.73	97.38	6.14
8	12.50	28.70 20.23 15.23 11.67 8.90 6.64 4.72 3.06	99.20	9.36
16	6.25	16.08 12.99 10.93 9.39 8.15 7.12 6.23 5.46 4.77 4.15 3.58 3.06 2.58 2.14 1.73 1.34	99.78	11.99

Table 5-1

FILLX in this regard increases with LENGTH degenerative with larger values of PROPOR. Large values of PROPOR ~~itself depends on~~ themselves require a large-sized population; even with as many as 16 samples, the actual ratio falls far short of the prescribed one.

-- Programming example 5-1: subroutine FILLX --

Table 5-1: Statistical pools generated by subroutine FILLX for PROPOR=16.0 - The parameter AVG is calculated in each instance by dividing 100 by LENGTH.

5.1.2 Generating Pools from Stored Discrete Distributions

Suppose we desire to generate a population of N samples according to a discrete distribution whose density for the ith option is f(i). Then Equation 5-1 gives n(i), the number of times option i will occur in this population.

$$n(i) = N * f(i) \qquad \text{(Equation 5-1)}$$

Unfortunately, Equation 5-1 tends to produce results like:

13.55	0's
16.26	1's
9.76	2's
5.25	3's

and so on. A program must account for the fractional parts of these numbers if it is to tailor the size of the population to the size of the frame.

The library subroutine FILL generates pools of samples from stored discrete distributions. FILL requires six arguments:

1. POOL - FILL returns LENGTH samples beginning in this location. POOL must be an array of dimension LENGTH; it may be either integer or real.

2. VALUE - Repertory of options. VALUE must be an array of dimension NUM whose type is identical to that of POOL.

3. WEIGHT - Array of weights associated with each option in VALUE. WEIGHT must be a real array of dimension NUM.

4. SUM - Sum of weights stored in array WEIGHT. SUM must be real.

5. LENGTH - Number of samples to be assembled in array POOL. LENGTH must be an integer.

Ex 5-2

```
subroutine FILL(POOL,VALUE,WEIGHT,SUM,LENGTH,NUM)
dimension POOL(1),VALUE(1),WEIGHT(1)
Y = SUM/float(LENGTH)
X = Y/2.0
N = 1
do (L=1,LENGTH)
  do
    W = WEIGHT(N)
    if (X.le.W) exit
    X = X - W
    N = N + 1
    if (N.gt.NUM) stop 'Bad weights for FILL.'
  repeat
  POOL(L) = VALUE(N)
  X = X + Y
repeat
return
end
```

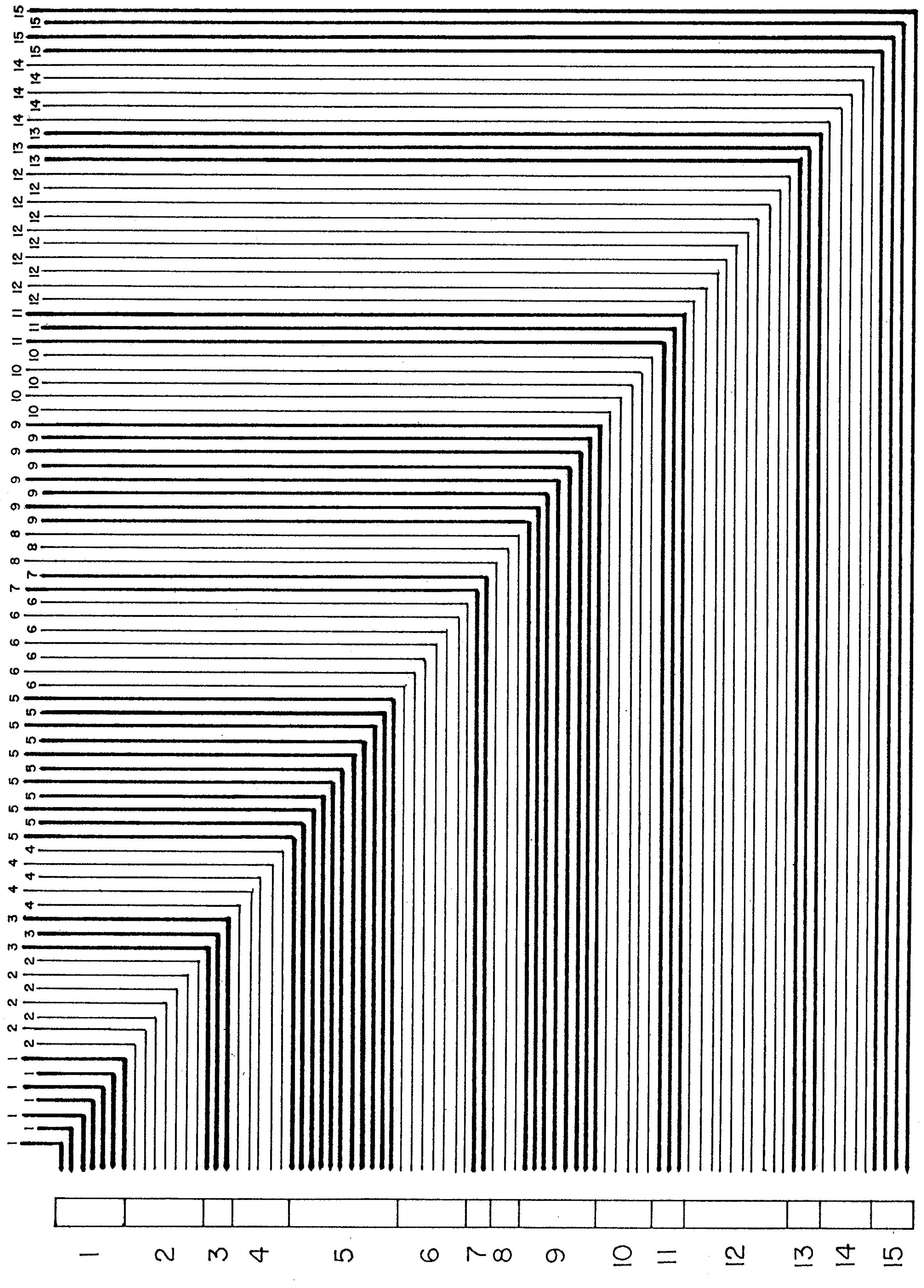


Fig 5-1

6. NUM - Number of distinct options.

Figure 5-1 illustrates how FILL accomodates fractional parts. The principle behind FILL ~~are~~ is similar to the principle behind the library subroutine SELECT (heading 4.2.2.5): the relative weight associated with each option is treated as a "region" ~~of a total~~ of a uniform range stretching from 0 to SUM. In this case, LENGTH values equally spaced between 0 and SUM are compared against these relative weights in order to determine how many 'copies' of each option should be placed in the pool.

-- Programming example 5-2: subroutine FILL --

Figure 5-1: Mechanics of subroutine FILL - The left-hand strip depicts relative weights for a repertory of 15 options, while the numbers along the top represent a pool of 83 samples.

5.2 RANDOM SHUFFLING

The library subroutine SHUFLE randomly shuffles an array. ~~(note 3).~~ SHUFLE requires two arguments:

Ex 5-3

```
C     subroutine SHUFLE(VALUE,NUM)
      dimension VALUE(1)
      J = NUM
      X = float(J)
      do
C       Select random location K between 1 and J (inclusive)
        K = ifix(X*RANF()) + 1
C       Exchange VALUE(J) with VALUE(K)
        V = VALUE(J)
        VALUE(J) = VALUE(K)
        VALUE(K) = V
C       Decrement index
        J = J - 1
        if (J.le.1) exit
        X = X - 1.
      repeat
      return
      end
```

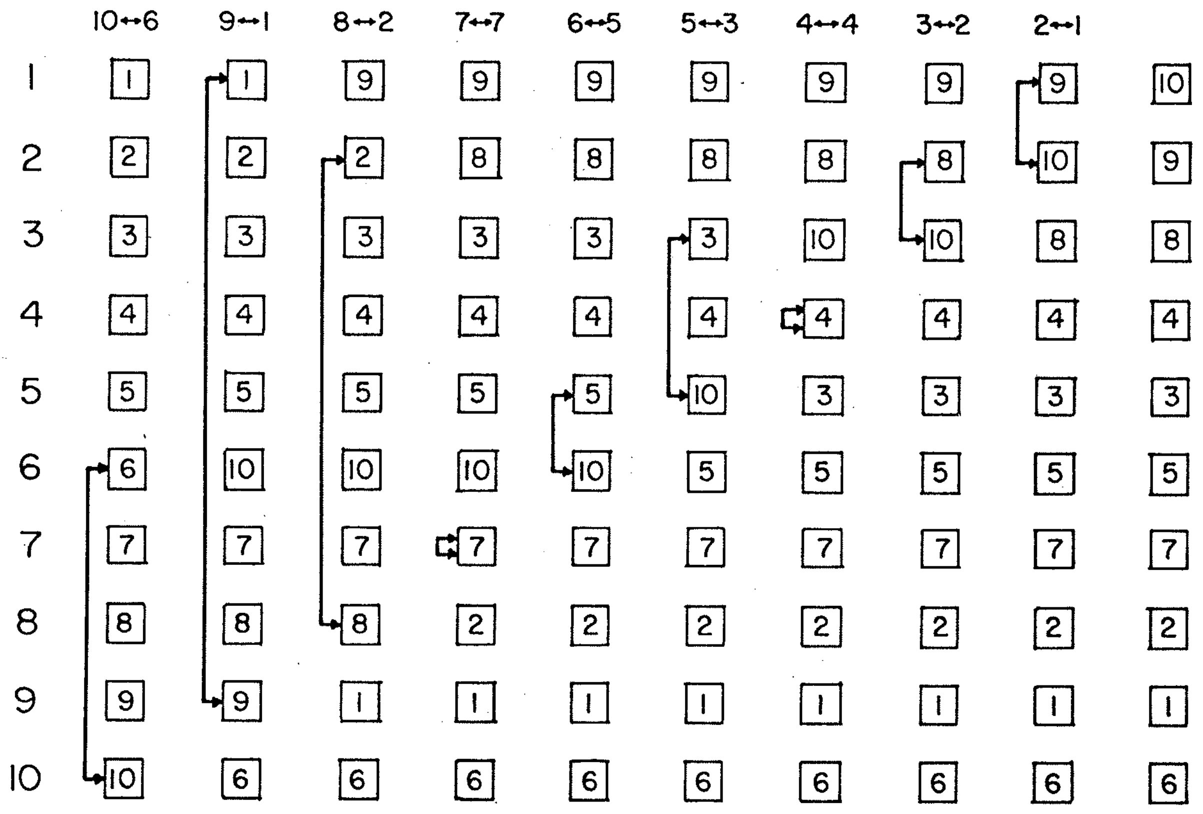

10↔6 9↔1 8↔2 7↔7 6↔5 5↔3 4↔4 3↔2 2↔1
1 1 1 9 9 9 9 9 9 9 10
2 2 2 2 8 8 8 8 8 10 9
3 3 3 3 3 3 3 10 10 8 8
4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 10 3 3 3 3
6 6 10 10 10 10 5 5 5 5 5
7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 2 2 2 2 2 2 2
9 9 9 1 1 1 1 1 1 1 1
10 10 6 6 6 6 6 6 6 6 6

Fig 5-2

1. VALUE - Array of samples to be shuffled. Value may be either a real or an integer array.

2. NUM - Number of samples stored in array VALUE.

The algorithm for SHUFLE was developed by Moses and Oakford (1963) and independently by Durstenfield (1964); it works by stepping backwards through the array, exchanging elements at random either with themselves or with earlier elements (note 3). Figure 5-2 illustrates how SHUFLE might randomly permute an array containing the integers from 1 to 10.

-- Programming example 5-3: subroutine SHUFLE --

Figure 5-2: Mechanics of Random Shuffling - The first 9 columns show the state of VALUE during consecutive iterations, while the two numbers above each column indicate which two locations SHUFLE has decided to exchange. The last column shows the completely shuffled array.

Gottfried Michael Koenig's PROJECT1 program (Koenig, 1970a) and PROJECT2 program (Koenig, 1970b) both include a feature called SERIES which samples a statistical pool, shuffling the pool at the end of each frame. The similarly named subroutine given here performs a similar task. This subroutine requires four arguments:

1. RESULT - Each call to SERIES transfers one option from array VALUE into this location (line 9). RESULT may be either a real number or an integer.

2. VALUE - Pool of samples. VALUE must be an array of dimension NUM whose type is identical to that of RESULT.

3. IDX - Index to most recently consulted position in array VALUE. This index insures that no entry in VALUE repeats until the entire pool has been exhausted. SERIES updates IDX with each call and shuffles VALUE whenever IDX reaches NUM (lines 3-8). IDX must be an integer.

4. NUM - Number of samples in array VALUE. NUM must be an integer.

-- Programming example 5-4: subroutine SERIES --

Ex 5-4

```
subroutine SERIES(RESULT,VALUE,IDX,NUM)
dimension VALUE(1)
if (IDX.ge.NUM) then
  call SHUFLE(VALUE,NUM)
  IDX = 1
else
  IDX = IDX + 1
end if
RESULT = VALUE(IDX)
return
end
```

5.3 DETERMINING THE SIZE OF FRAMES

For the approach described in this chapter to be effective, the number of items in a frame should be large enough to express the characteristics of the prescribed distribution. For example, a Bernoulli distribution (heading 4.2.2.1) with one chance in five of success makes no sense at all for frames containing less than five items. Equation 5-2 provides a rule for determining the minimum effective number of items in a frame, where f represents the density function for a discrete distribution of options. In order for the ith option to be represented in the frame, the number of items must be at least as large as the number N(i):

$$N(i) = \frac{1}{f(i)} \qquad \text{(Equation 5-2)}$$

For continuous statistical distributions, in order for the region bounded by x1 and x2 to be represented in a frame by at least one value, the number of elements in the frame must be at least as large as the number N(x1,x2) given in Equation 5-3, where the function f is the continuous density function.

$$N(x1,x2) = \frac{1}{f[(x1+x2)/2] * (x2-x1)} \qquad \text{(Equation 5-3)}$$

5.4 APPLICATIONS

5.4.1 Herbert Brun: Sonoriferous Loops

Figure 5-3: Herbert Brun, Sonoriferous Loops, measures 0-9 - Copyright 1964 Herbert Brun.

One of the earliest compositions employing random shuffles to organize statistical frames is Herbert Brun's Sonoriferous Loops (1964; described in Hiller, 1970). This work consists of nine sections alternating an ensemble of five instruments with digitally synthesized interludes of three contrapuntal parts. Each section is distinguished by a distinct selection of "play/rest probabilities" governing the relative activity in each part, instrumental or synthetic. Pitches in Sonoriferous Loops conform to twelve-tone rows (and, by implication, are organized into frames of twelve notes) which Brun's program randomly shuffles at the beginning of each cycle. This process of selection provides degrees for all parts simultaneously, so that the composite texture obeys a uniform distribution of degrees; however, Brun further constrains the process so that each part

SONORIFEROUS LOOPS

Herbert Brün
Opus 32 - 1964

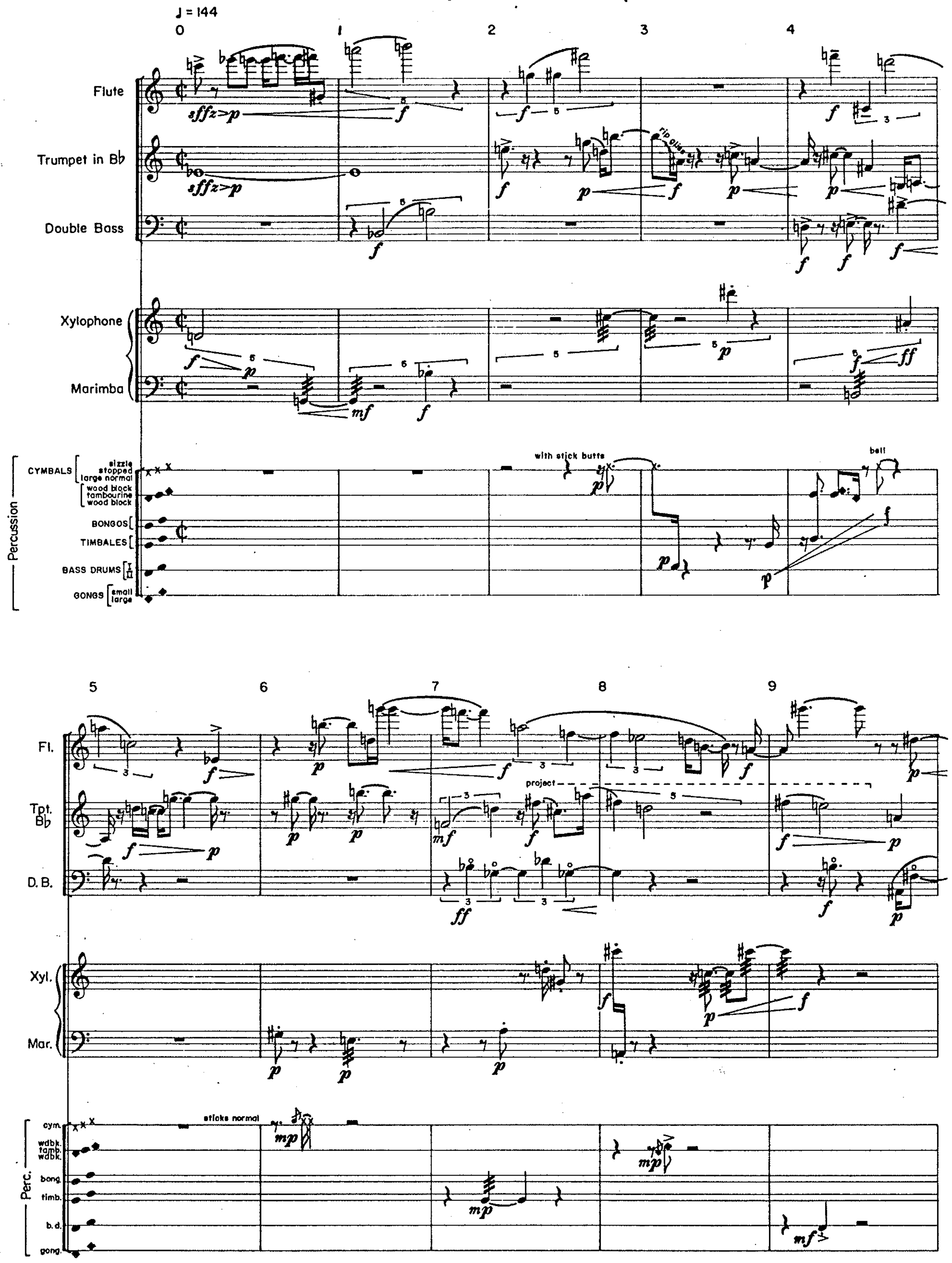


Fig 5-3

internally obeys a similar distribution: according to Brun, if a selected degree does not conform to the internal distribution of a part, then the program replaces this unacceptable degree in the row -- rendering it available for use by another part -- and selects another option.

5.4.2 Gottfried Michael Koenig: Uebung fur Klavier

The notion which we designate in this book as "statistical frames" provides much of the underlying conceptual basis for Gottfried Michael Koenig's PROJECT1 and PROJECT2 programs. Of Koenig's computer-composed scores, his most elaborate is the Uebung fur Klavier composed using PROJECT2. The score to this work consists of twelve "structures". Each structure appears in three "variants" generated from more-or-less the same directives to PROJECT2. Koenig acknowledges ~~accommodates~~ the random aspects of his composing process by allowing the performer to choose between variants; the fact that all three variants otherwise adhere rigorously to the Koenig's directives is reflected in Koenig's instructions that at least one variant of each structure must be played, and that all twelve structures must occur in a prescribed order.

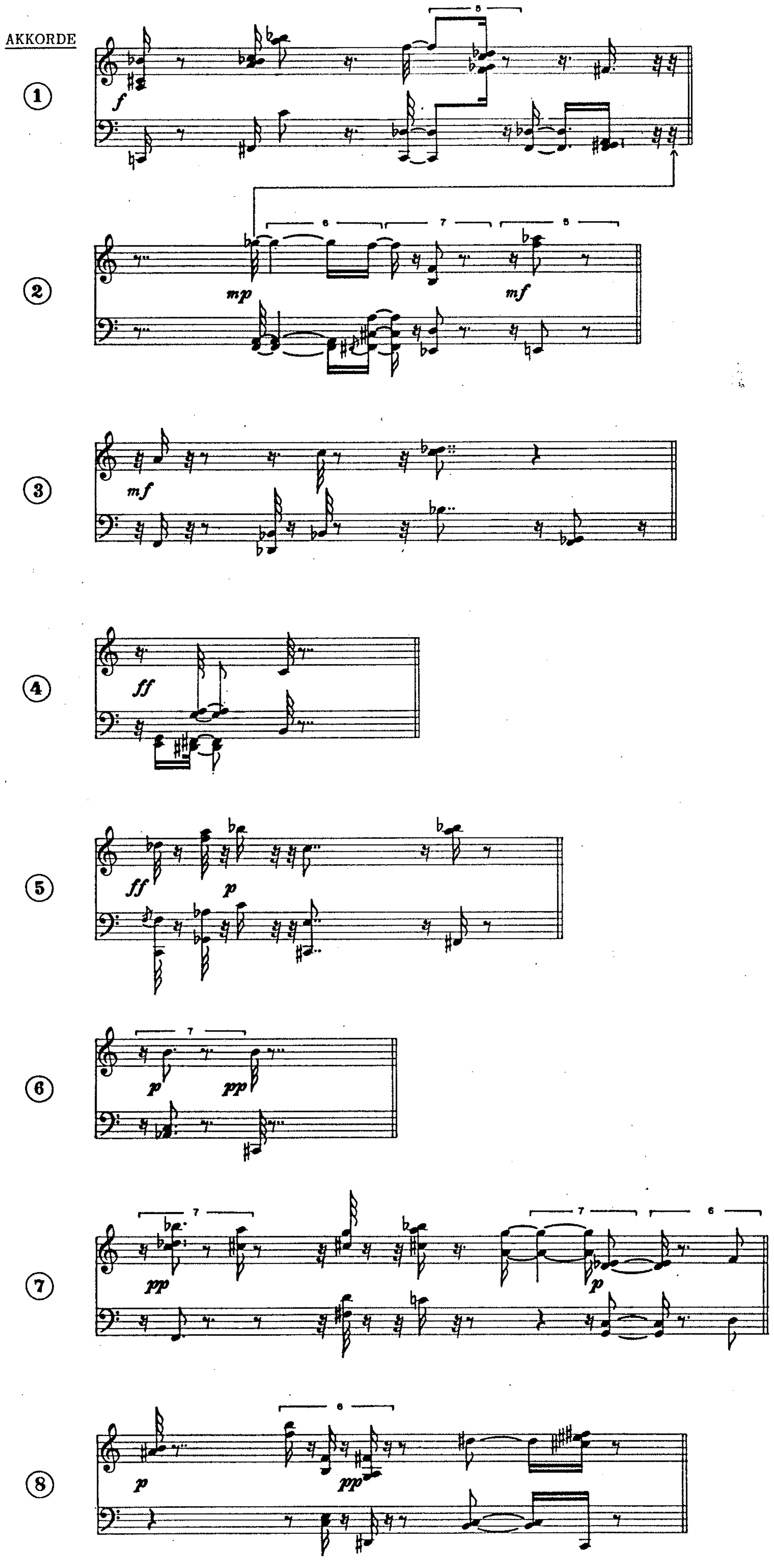


Fig 5-4 (page 1)

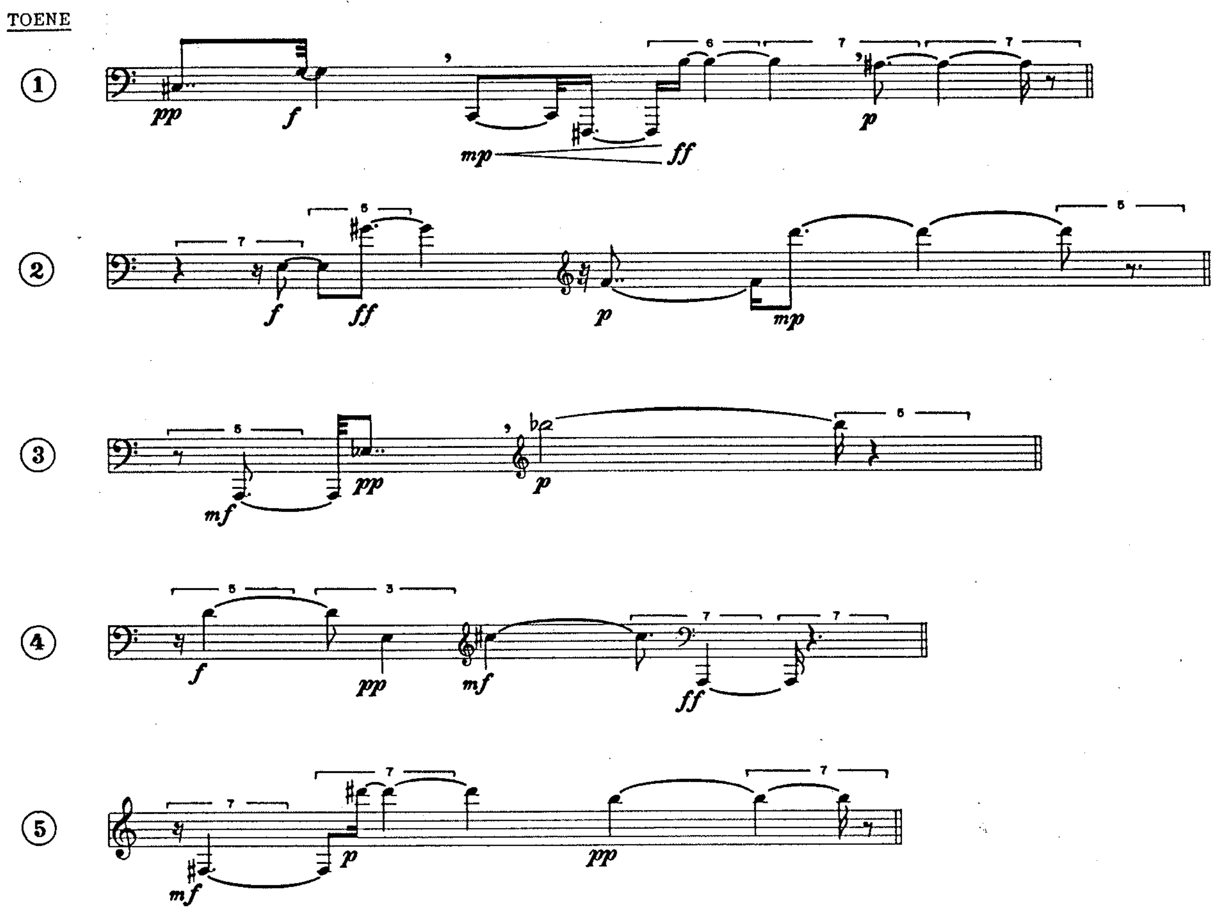


Fig 5-4 (page 2)

Figure 5-4: Gottfried Michael Koenig, _Uebung fur Klavier_ (1970), structure 8, variant 1 - Copyright 1970 Gottfried Michael Koenig.

To illustrate the workings of PROJECT2, we will examine structure 8 of the _Uebung fur Klavier_. Figure 5-4 reproduces variant 1 of this structure. It incorporates two sets of material, the eight "groups of chords" and the five "groups of tones". These two sets of material result from two independent sets of directives to PROJECT2. Koenig provides the following synopsis of structure 8 in his instructions to the performer, which allow elements of performer discretion much like those used to control the performance as a whole:

> A transition: chords and tones alternate. There are also several groups of both chords and tones in every variant. Not all the groups of chords or tones have to be played, but those selected must be in the given order. Groups of tones must always be separated by chords, groups of chords can join on to one another (without jumping over groups of chords). A second group of chords then joins on rhythmically where indicated by the arrow. Such arrows, when present, have no significance if groups of chords alternate with

groups of tones. What must be played are: at least three groups of tones in the first variant .... The number of groups of chords is left to the player, but he should begin with a group of chords.

Table 5-2: Chordal table for structure 8 of Koenig's Uebung fur Klavier.

Table 5-3: Intervallic table for structure 8 of Koeing's Uebung fur Klavier.

PROJECT2 incorporates three "principles" for selecting chromatic degrees which Koenig names ROW, CHORD, and INTERVAL. All three "principles" affect only degrees -- register is treated as an independent parameter. ROW is monophonic; it implements the basic serial procedures described in Chapter 3. Koenig does not employ ROW in Structure 8. CHORD is homophonic. Given a table of chords, each chord containing from two to twelve degrees, CHORD employs a selection procedure such as ALEA (heading 4.2.1.2) or SERIES to select one chord from this table and to apply a (register-free) transposition. For the "groups of chords" in structure 8, Koenig supplies CHORD with the list of chords detailed in Table 5-2 and directs ALEA to choose chords

Index	Degrees					
1	C#	F				
2			F#	A		
3					Bb	C
4		F	F#			
5				A	Bb	
6	C#	F	F#			
7	C#	F				C
8			F#	A	Bb	
9		F	F#	A		
10	C#				Bb	C
11				A	Bb	C
12	C#	F	F#	A		
13	C#	F			Bb	C
14			F#	A	Bb	C
15		F	F#	A	Bb	
16	C#	F	F#			C
17	C#			A	Bb	C

Table 5-2

Most Recent Interval	Current Interval										
	m2	M2	m3	M3	P4	TT	P5	m6	M6	m7	M7
m2	yes	no	no	no	no	no	no	no	no	yes	yes
M2	no	yes	yes	yes	no	no	yes	yes	no	yes	yes
m3	no	yes	no	no	no	no	yes	no	yes	no	no
M3	no	no	no	no	no	yes	no	yes	no	yes	no
P4	no	no	no	no	no	no	yes	no	yes	yes	no
TT	no	yes	no	no	no	yes	no	yes	no	no	no
P5	no	yes	yes	no	yes	no	no	no	no	no	no
m6	no	yes	no	yes	no	no	no	no	no	yes	no
M6	no	no	yes	no	yes	no	no	no	no	yes	no
m7	yes	yes	no	yes	yes	yes	no	no	yes	yes	no
M7	yes	yes	no	no	no	no	no	no	no	no	yes

Table 5-3

and to select transpositions. INTERVAL is monophonic, like ROW. INTERVAL implements a strategy which we will come to know in Chapter 6 as a "Markov chain". Each call to INTERVAL returns one of the twelve chromatic intervals, this interval is then applied to the 'most recent degree' in order to determine the 'current degree'. This 'current degree' then becomes the 'most recent degree' for the next call to INTERVAL, perpetuating the chain. INTERVAL exploits this process to incorporate sensitivity to the local intervallic context: in order to select the current interval, INTERVAL consults a logical table detailing which of the twelve chromatic intervals are acceptable as the 'current interval' given the 'most recent interval' (the 'current interval' then becomes the 'most recent interval' next time around) and employs ALEA to select from these acceptable intervals. For the "groups of tones" in structure 8, Koenig supplies INTERVAL with the logical table detailed in Table 5-3.

Among the remaining parameters selected by PROJECT2 are register, "entry delay" (Koenig's term for periods between consecutive attacks), duration, and dynamics. For each of these parameters, the user of PROJECT2 must either specify a constant or instead provide a "supply" of options and specify the selection feature as SEQUENCE (heading 3.4.1), ALEA, SERIES, or one of several other features yet to be described in this book. Registers are expressed as lower and upper limits within which

PROJECT2 places a degree at random; in Structure 8 Koenig holds these limits constant. To select "entry delays", Koenig uses ALEA. Durations for the groups of chords are determined similarly, subject to Koenig's proviso that no chord should overlap its successor (if a duration provided by ALEA does not meet this proviso, PROJECT2 discards it and initiates a new choice); durations of tones are identical with entry delays. To select chordal dynamics, Koenig employs a feature called GROUP which repeats each dynamic a variable number of times before choosing a new one; in this instance, GROUP employs SERIES both to determine the number of repetitions and to choose a new dynamic when all repetitions have been exhausted.

5.5 DEMONSTRATION 3: STATISTICAL FRAMES

Figure 5-5: Compositional data for Demonstration 3.

Demonstration 3 illustrates an automated compositional process in which all decisions are effected by sampling randomly shuffled pools without replacement. Like Demonstration 2, compositional control over Demonstration 3 is limited to prescribing distributions options affecting attributes of

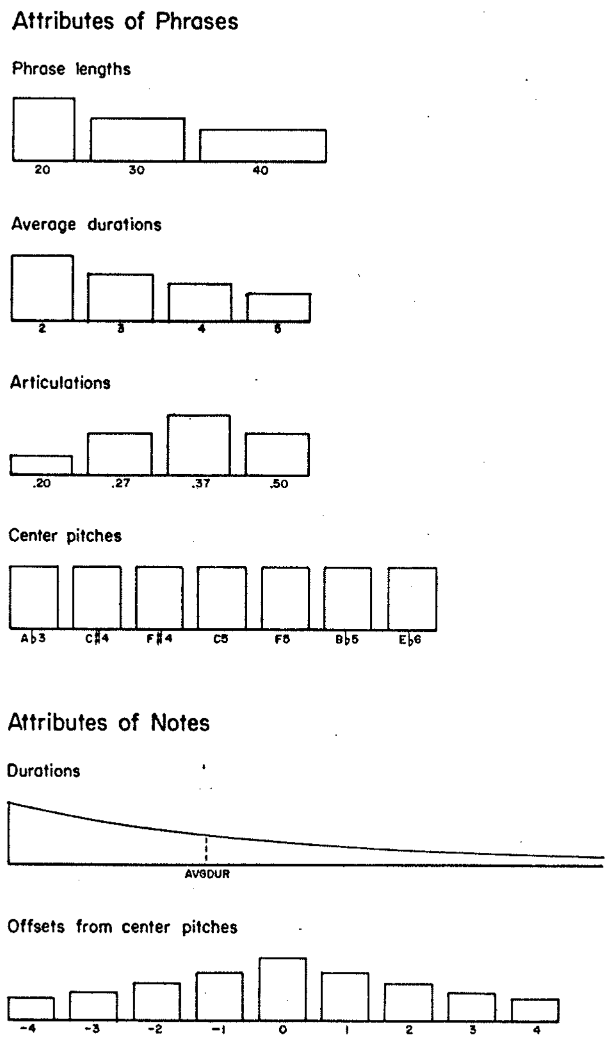

Attributes of Phrases
Phrase lengths
20
30
40
Average durations
2
3
4
5
Articulations
.20
.27
.37
.50
Center pitches
A♭3
C♯4
F♯4
C5
F5
B♭5
E♭6
Attributes of Notes
Durations
AVGDUR
Offsets from center pitches
-4
-3
-2
-1
0
1
2
3
4

Fig 5-5

phrases and notes. However, the techniques described earlier in this chapter insure much tighter adherence to these distributions than had been possible using direct random selection.

5.5.1 Compositional Directives

Figure 5-5 depicts the distributions of musical attributes affecting phrases and notes in Demonstration 3. The most prominent stylistic trait distinguishing Demonstration 3 from its predecessor is that where Demonstration 2 exploited the twelve chromatic degrees with equal probability, Demonstration 3 exploits gamuts of only nine adjacent semitones within a given phrase and weights pitches at the center of a gamut three times as strongly as it weights the outer pitches. Figure 5-6 graphs the musical attributes selected for phrases; a transcription of the musical product appears in Figure 5-7.

Figure 5-6: Profile of Demonstration 3.

Figure 5-7: Transcription of Demonstration 3.

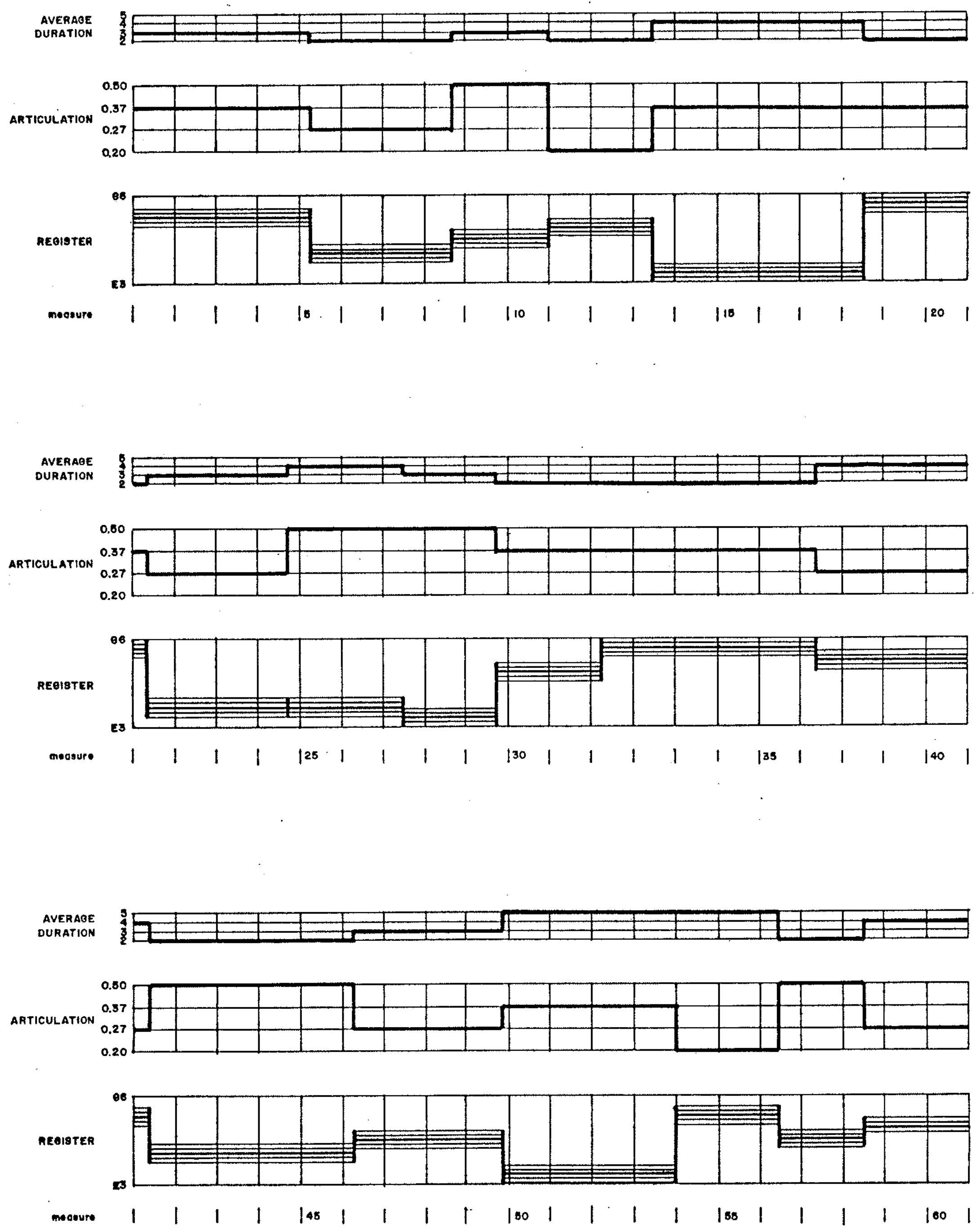

AVERAGE DURATION
5
4
3
2
ARTICULATION
0.50
0.37
0.27
0.20
REGISTER
66
E3
measure
5
10
15
20
25
30
35
40
45
50
55
60

Fig 5-6

Demonstration 3

Clarinet

Charles AMES

STRICTLY ♩ = 80

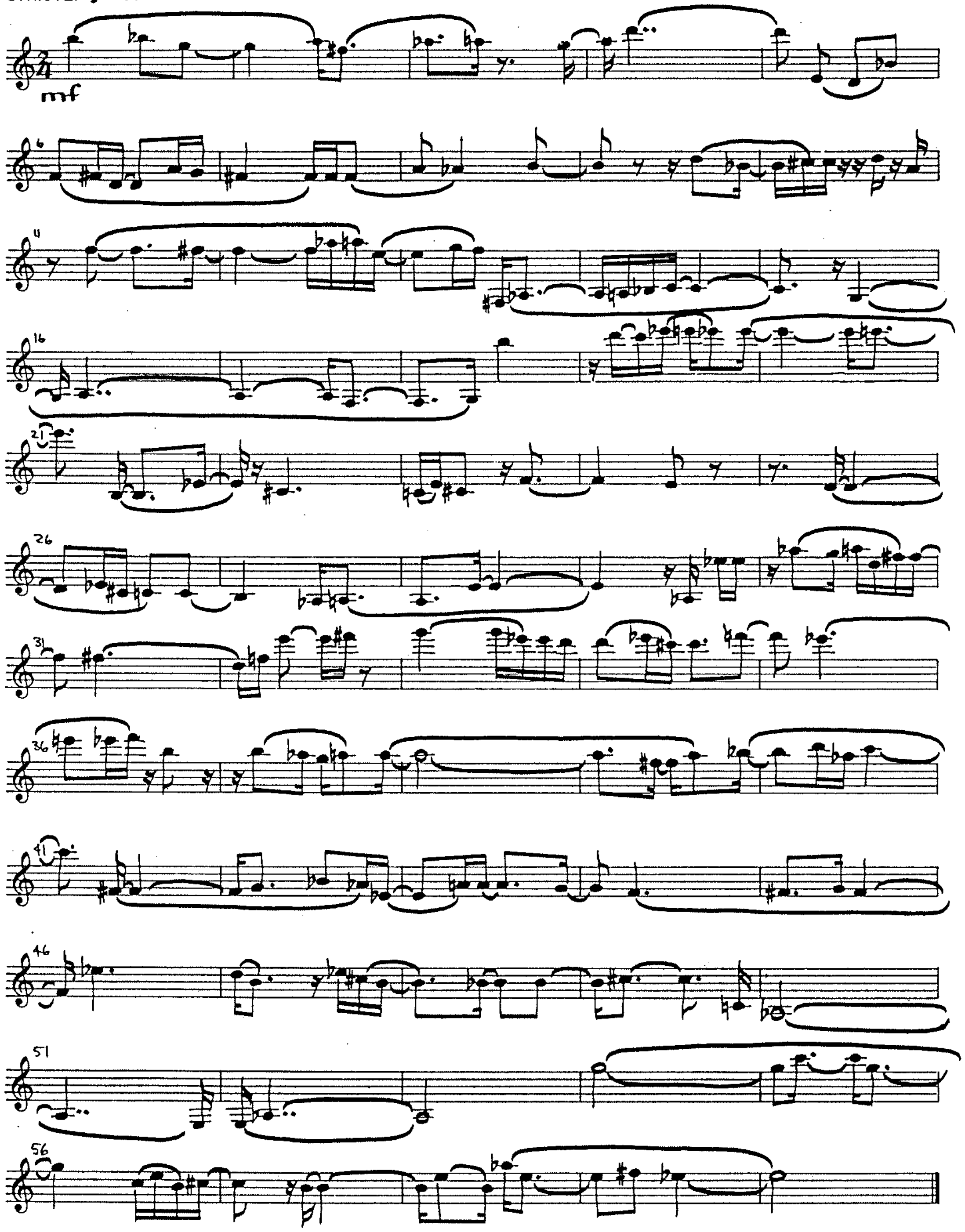


Fig 5-7

5.5.2 Implementation

-- Programming example 5-5: program DEMO3 (2 pages) --

Like program DEMO2, program DEMO3 reflects the ~~two-tiered~~ musical structure of phrases and notes as a design of nested loops: an "outer" compositional loop (lines 43 to 58 of DEMO3 proper) for phrases and an "inner" compositional loop (lines 10-39 of PHRASE) for notes and rests. An extended initialization section (lines 27-39 of DEMO3) ~~serves to~~ generates pools of samples for each musical attribute through calls to the library subroutines FILL and FILLX. The actual selection of attributes is far simpler than in DEMO2 because the distributions illustrated in Figure 5-5 are inherent in the pools; a call to the library subroutine SERIES is sufficient in each case.

The symbols of DEMO3 adhere to seven mnemonic 'roots' corresponding to attributes of phrases or units:

1. PHR - length of phrase.

2. AVG - average duration of notes; the average duration of rests is half as large.

3. ART - articulation, expressed as the probability that an

```
      subroutine PHRASE(KTIME,ITIME,AVGDUR,ARTIC,IREG,
     :                  POLUNF,POLDUR,POLPCH)
      parameter (MUNF=20,MDUR=10,MPCH=40)
      integer POLPCH(MPCH)
      real    POLUNF(MUNF),POLDUR(MDUR)
      data IDXDUR/0/,IDXUNF/0/,IDXPCH/0/
C
C     Inner composing loop
C
      do
C       Select note or rest (no two consecutive rests)
        if (IPCH.eq.0) then
          R = 1.0
        else
          call SERIES(R,POLUNF,IDXUNF,MUNF)
        end if
        if (R.gt.ARTIC) then
C         Select duration of note
          call SERIES(DUR,POLDUR,IDXDUR,MDUR)
          DUR = AVGDUR*DUR + REMAIN
          IDUR = max0(1,ifix(DUR+0.5))
          REMAIN = DUR - float(IDUR)
C         Select pitch
          call SERIES(IPCH,POLPCH,IDXPCH,MPCH)
          IPCH = IREG + IPCH
        else
C         Select duration of rest
          call SERIES(DUR,POLDUR,IDXDUR,MDUR)
          DUR = AVGDUR/2.0 * DUR + REMAIN
          IDUR = ifix(DUR)
          REMAIN = DUR - float(IDUR)
C         Null pitch indicates rest
          IPCH = 0
        end if
C       Write note or rest
        call WNOTE(ITIME,IDUR,IPCH)
C       Test for end of phrase
        if (ITIME.ge.KTIME) exit
      repeat
      return
      end
      program DEMO3
C
C     Demonstration of statistical frames
C
      parameter (MPHR=13,MAVG=19,MART=8,MREG=7,
     :           MUNF=20,MDUR=10,MPCH=40)
      integer POLPCH(MPCH),POLREG(MREG),POLPHR(MPHR)
      real    POLAVG(MAVG),POLART(MART),POLUNF(MUNF),
     :        POLDUR(MDUR)
      integer VALPHR(3),VALPCH(9)
      real    VALAVG(4),VALART(4)
      real    WGTPHR(3),WGTPCH(9),WGTAVG(4),WGTART(4)
      data VALPHR/20,30,40/,WGTPHR/6.,4.,3./,SUMPHR/13.0/
      data VALAVG/2.,3.,4.,5./,WGTAVG/7.,5.,4.,3./,SUMAVG/19./
      data VALPCH/-4,-3,-2,-1, 0, 1, 2, 3, 4/,
     :     WGTPCH/1.00,1.32,1.73,2.28,3.00,2.28,1.73,1.32,1.00/,
     :     SUMPCH/15.66/
      data VALART/.2,.27,.37,.5/,WGTART/1.,2.,3.,2./,SUMART/8./
      data POLREG/44,49,54,60,65,70,75/
      data POLUNF/.025,.075,.125,.175,.225,.275,.325,
     :            .375,.425,.475,.525,.575,.625,.675,
     :            .725,.775,.825,.875,.925,.975/
      data IDXPHR/0/,IDXAVG/0/,IDXART/0/,IDXREG/0/
C
C     Initialization
C
      open (2,file='DEMO3.DAT',status='NEW')
      ITIME = 0
      MTIME = 8 * 60
C     Generate pool of phrase lengths
      call FILL(POLPHR,VALPHR,WGTPHR,SUMPHR,MPHR,3)
C     Generate pool of average durations for notes
      call FILL(POLAVG,VALAVG,WGTAVG,SUMAVG,MAVG,4)
C     Generate pool of rest probabilities
      call FILL(POLART,VALART,WGTART,SUMART,MART,4)
C     Generate pool of durations
      call FILLX(POLDUR,1.0,1000.0,MDUR)
C     Generate pool of offsets for pitch
      call FILL(POLPCH,VALPCH,WGTPCH,SUMPCH,MPCH,9)
C
C     Outer composing loop
C
      do
C       Select duration of phrase
        call SERIES(IPHR,POLPHR,IDXPHR,MPHR)
        KTIME = KTIME + IPHR
C       Test for end of composition
        if (KTIME.gt.MTIME) exit
C       Select average duration for notes in phrase
        call SERIES(AVGDUR,POLAVG,IDXAVG,MAVG)
C       Select probability of rest
        call SERIES(ARTIC,POLART,IDXART,MART)
C       Select register
        call SERIES(IREG,POLREG,IDXREG,MREG)
C       Compose phrase
        call PHRASE(KTIME,ITIME,AVGDUR,ARTIC,IREG,
     :              POLUNF,POLDUR,POLPCH)
      repeat
      close (2)
      stop
      end
```

Ex 5-5

iteration of the inner composing loop will produce a rest rather than a note.

4. REG - register, expressed as the central pitch in a nine-semitone gamut.

5. UNF - uniform values between zero and one, used in play/rest trials.

6. DUR - duration of note or rest.

7. PCH - deviations from central pitch of gamut.

Arrays beginning with POL serve as pools of values for each attribute; the size of each pool is given by a parameter starting with the letter M. Each pool has an associated index, required by SERIES; this index begins with IDX. In those cases where pools are generated from smaller sets of values using subroutine FILL, the original values reside in arrays beginning with VAL, their associated weights reside in arrays beginning with WGT, and the sum of these weights is held by a variable beginning with SUM.

~~Most of the opening portion of DEMO3 (lines 27-39) is devoted to generating pools of samples. DEMO3 is otherwise~~

~~similar in design to DEMO2; an outer loop selects attributes for phrases (lines 43-58 of DEMO3 proper) while an inner loop composes specific notes (lines 10-39 of PHRASE). The actual selection of attributes is simpler than in DEMO2 because distributions are inherent in the pools; a call to SERIES suffices.~~ ¶The notion of statistical frames is inherent in Demonstration 3 in that these distributions become manifest each time SERIES cycles through a pool.

nc¶ Notice in particular how DEMO3 generates the pool of exponentially distributed durations (line 37) by specifying an value of 1. for the AVG argument to subroutine FILLX. It is left to PHRASE to stretch durations as necessary when it composes individual notes or rests ([illegible] lines 20 and 29).

5.6 AN ALTERNATE APPROACH: KOENIG'S "RATIO" FEATURE

Gottfried Michael Koenig's PROJECT2 program (Koenig 1970b) contains a feature called RATIO which produces results which are equivalent to what one would obtain using the library subroutines FILL and SERIES. However, the mechanism of RATIO is quite different. The user of PROJECT2 provides a "supply" of distinct options and designates a number of occurances for each option;

the total of all these numbers of occurances must match the number of items in the frame. Each call to RATIO returns an option selected at random, where the relative likelihood of each option depends upon its associated number of occurances. Once it has selected an option, RATIO decrements this number and correspondingly reduces this option's likelihood of being chosen during the next call; when the number falls to zero, the option ceases to be selected.

An especially attractive feature of Koenig's strategy is its potential sensitivity to duration. For example, suppose that we desire to maintain an equilibrium among the degrees of a scale. An obvious method of enforcing this desire would be to impose Schoenberg's rule against repeating any degree until the whole scale has been used (note 4). However, this restriction fails when long and short durations are interspersed: without drastic compensations such as extreme registral displacement or extreme dynamic emphasis, we would be unlikely to judge one sixteenth-note B equally significant to one whole-note F. An alternate approach is to require equal <u>durational</u> emphasis of each degree, so that we could balance a whole note F by, say, three (not necessarily consecutive) B's, a sixteenth note, a half note, and a double-dotted quarter note. Using Koenig's method, such a requirement can by imposed by having RATIO process cumulative durations in place of numbers of occurances.

5.6.1 Implementation

The library subroutine RATIO implements Koenig's RATIO feature with sensitivity to duration. RATIO requres 6 arguments:

1. RESULT - RATIO selects an index I to one of NUM options stored in array VALUE and transfers VALUE(I) to RESULT (line 10). RESULT may be either an integer or a real number.

2. VALUE - Repertory of options. VALUE must be an array of dimension NUM whose type is identical to that of RESULT.

3. WEIGHT - Weights associated with each option in array VALUE. Initially, these weights correspond to the portions of the total duration of all the items in the frame. After it selects an index I, RATIO reduces WEIGHT(I) to either WEIGHT(I)-DUR or 0, whichever is larger (lines 12-17). WEIGHT must be a real array of dimension NUM.

4. SUM - Sum of the NUM weights stored in array WEIGHT. Initially, SUM corresponds to the total duration of all the items in the frame. Each call to RATIO reduces this

quantity by DUR. SUM must be real.

5. DUR - Duration of item for which RESULT is being selected.

6. NUM - Number of options.

Lines 13-16 of subroutine RATIO prevent any option from assuming negative weights. For example, if the ith option has weight 3.0 and is selected for an item with duration 4.0, then line 12 would reduce the weight to -1.0. Without lines 13-16, this negative weight would not only remove the ith option from further consideration, it would also reduce the effective weight associated with the i+1st option by 1.0. To see why, consider what would happen during the next call to RATIO. Suppose that line 3 produces a random value R large enough to survive through the first i-1 weights. Then during the ith iteration of lines 4-8, the program will set W to -1.0 in line 5, determine that R is larger than W (since R must be positive) in line 6, and proceed to subtract W from R in line 7. However, since W is already negative, line 7 will act to <u>increase</u> R by 1.0. Should the i+1st weight be a fraction smaller than 1.0, such a negative weight would effectively remove both the ith option <u>and</u> the i+1st option from consideration.

Ex 5-6

```
subroutine RATIO(RESULT,VALUE,WEIGHT,SUM,DUR,NUM)
dimension VALUE(1),WEIGHT(1)
R = SUM * RANF()
do (I=1,NUM)
  W = WEIGHT(I)
  if (R.le.W) exit
  R = R - W
repeat
if (I.gt.NUM) stop 'Bad weights for RATIO.'
RESULT = VALUE(I)
SUM = SUM - DUR
W = W - DUR
if (W.lt.0) then
  SUM = SUM - W
  W = 0.
end if
WEIGHT(I) = W
return
end
```

-- Programming example 5-6: subroutine RATIO --

If the selection is to be peformed relative to numbers of occurances (that is, without sensitivity to duration), then array element WEIGHT(I) should be set to the desired number of occurances for the Ith option, while SUM ~~will~~ should reflect the total number of items in the frame. In this case DUR must be set to unity for each call.

To incorporate sensitivity to duration, it is necessary to take into account how long each item lasts. Suppose, for example, that we have a frame containing MNOT notes, that the real array element DURNOT(I) contains the duration associated with the Ith note in the frame and that we wish to select an option to be stored in the integer array OPTNOT(I) for I=1,...,MNOT. Suppose further that we have MOPT options, that an integer array element VALOPT(J) and a real array element DENSTY(J) contain the value and density for the Jth option for J=1,...,MOPT. (The MOPT values in array DENSTY must sum to unity.) Then the following excerpt of code would calculate the required initial values for WEIGHT and SUM and perform the required calls to RATIO:

-- Programming example 5-7 --

Ex 5-7

```
C      Determine cumulative duration over all notes in frame
       SUMDUR = 0.
       do (INOT=1,MNOT)
         SUMDUR = SUMDUR + DURNOT(INOT)
       repeat
C      Determine absolute weights for each option
       do (IOPT=1,MOPT)
         WGTOPT(IOPT) = SUMDUR * DENSTY(IOPT)
       repeat
C      Select options for each note in frame
       do (INOT=1,MNOT)
         call RATIO(OPTNOT(INOT),VALOPT,WGTOPT,SUMDUR,DURNOT(INOT),MOPT)
       repeat
```

It makes little sense to apply subroutine RATIO if a statistical frame contains insufficient notes to truly reflect the prescribed distribution of options. If Dmax and Dmin represent the longest and shortest durations, respectively, while NUM represents the number of options, then the number of notes in a frame should equal or exceed the value Nmin defined in Equation 5-4.

$$\text{Nmin} = \frac{\text{Dmax}}{\text{Dmin}} * \text{NUM} \qquad \text{(Equation 5-4)}$$

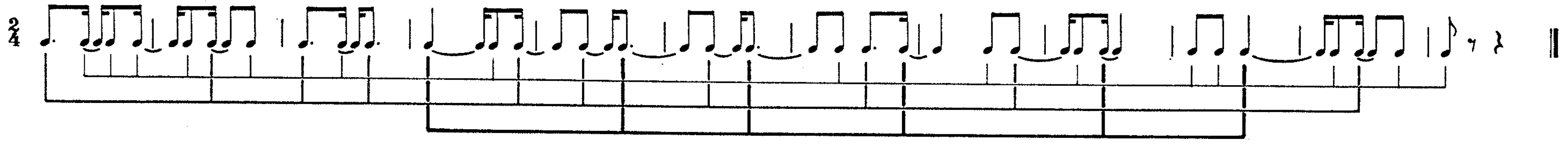
It is not necessary to assign equal weights to each option. One can alternately prescribe some array of densities DENSTY, then initialize WEIGHT(I) to DENSTY(I)*SUM. In such cases Equation 5-4 generalizes to become Equation 5-5, where Pmin represents the smallest of all NUM densities.

$$\text{Nmin} = \frac{\text{Dmax}}{\text{Dmin}} * \frac{\text{NUM}}{\text{Pmin}} \qquad \text{(Equation 5-5)}$$

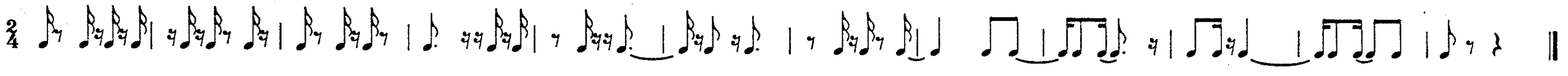
5.6.2 Example: Composing a 'Balanced' Melody

We illustrate the use of RATIO by using it to compose a melody consisting of 31 notes, each described by three attributes

STAGE I: Periods



STAGE II: Durations



STAGE III: Pitches

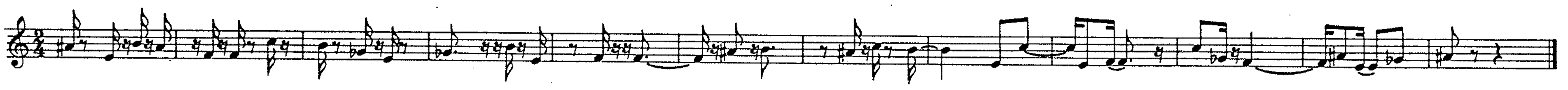


Fig 5-8

5-26a

selected during ~~distinct~~ successive stages of production: period, duration, and pitch. With regard to periods and pitches, selection is 'balanced' in the sense that each option receives equal emphasis over the melody as a whole. Figure 5-8 depicts the results of each stage.

Figure 5-8: Composing a 'balanced' melody - In Stage I, thin brackets show periods of length two, medium brackets show periods of length three, and thick brackets show periods of length five.

5.6.2.1 Stage I: Periods - Consecutive attacks may be separated by two, three, or five sixteenths. The process of selecting periods is numerically oriented. The initial numbers of occurances for these options are set so that the cumulative amount of time allotted to each option remains fixed: periods of length two occur 15 times, periods of length three occur 10 times, while periods of length five occur only 6 times. Notice that in each case the length of a period multiplied by the number of occurances yields a cumulative result of 30. Table 5-4 chronicles the attrition of the numbers of occurances allotted to each period as RATIO composes the sequence:

Measure & Beat	Weights 2	3	5	Sum	RANF()	Sum*RANF()	Selected Period
1:0	15.	10.	6.	31.	.655	20.33	3
1:3	15.	9.	6.	30.	.189	5.68	2
1:5	14.	9.	6.	29.	.329	9.56	2
1:7	13.	9.	6.	28.	.245	6.86	2
2:1	12.	9.	6.	27.	.386	10.43	2
2:3	11.	9.	6.	26.	.696	18.10	3
2:6	11.	8.	6.	25.	.394	9.86	2
3:0	10.	8.	6.	24.	.721	17.32	3
3:3	10.	7.	6.	23.	.003	0.09	2
3:5	9.	7.	6.	22.	.426	9.39	3
4:0	9.	6.	6.	21.	.884	18.56	5
4:5	9.	6.	5.	20.	.280	5.61	2
4:3	8.	6.	5.	19.	.661	12.58	3
5:2	8.	5.	5.	18.	.551	9.92	3
5:5	8.	4.	5.	17.	.858	14.60	5
6:2	8.	4.	4.	16.	.747	11.96	3
6:5	8.	3.	4.	15.	.827	12.42	5
7:2	8.	3.	3.	14.	.052	0.73	2
7:4	7.	3.	3.	13.	.691	8.99	3
7:7	7.	2.	3.	12.	.762	9.14	5
8:4	7.	2.	2.	11.	.280	3.08	2
8:6	6.	2.	2.	10.	.724	7.24	3
9:1	6.	1.	2.	9.	.068	0.62	2
9:3	5.	1.	2.	8.	.774	6.20	5
10:0	5.	1.	1.	7.	.111	0.78	2
10:2	4.	1.	1.	6.	.175	1.05	2
10:4	3.	1.	1.	5.	.986	4.93	5
11:1	3.	1.	0.	4.	.049	0.20	2
11:3	2.	1.	0.	3.	.726	2.18	3
11:6	2.	0.	0.	2.	.824	1.65	2
12:0	1.	0.	0.	1.	.757	0.76	2

Table 5-4

Table 5-4: Composing a sequence of rhythmic periods using subroutine RATIO.

5.6.2.2 Stage II: Durations - Figure 5-9 illustrates how RATIO selects durations for each of the rhythmic periods illustrated in Figure 5-8. This application divides the 31 notes into into six frames of five to six notes. Each call to RATIO selects one note out of a frame in order to append a sixteenth onto the ~~chosen~~ selected note's duration. Once again, selection is made on a numeric basis, with the "number of occurances" for each note given by the number of free sixteenths available to the note. Initially, every note in the frame has a duration of one sixteenth, so notes with period 2 have 1 free sixteenth; notes with period 3 have 2 free sixteenths, and notes with period 5 have 4 free sixteenths. The likelihood of choosing a note depends upon the number of free sixteenths left in its period; when the duration of a note reaches the period, the number of free sixteenths drops to zero, so the note is no longer considered.

In order to achieve a graded progression from detached to sustained articulations, the percentage of the total number of free sixteenths which RATIO is asked to fill rises incrementally from 0% in the first frame to 100% in the sixth. This percentage

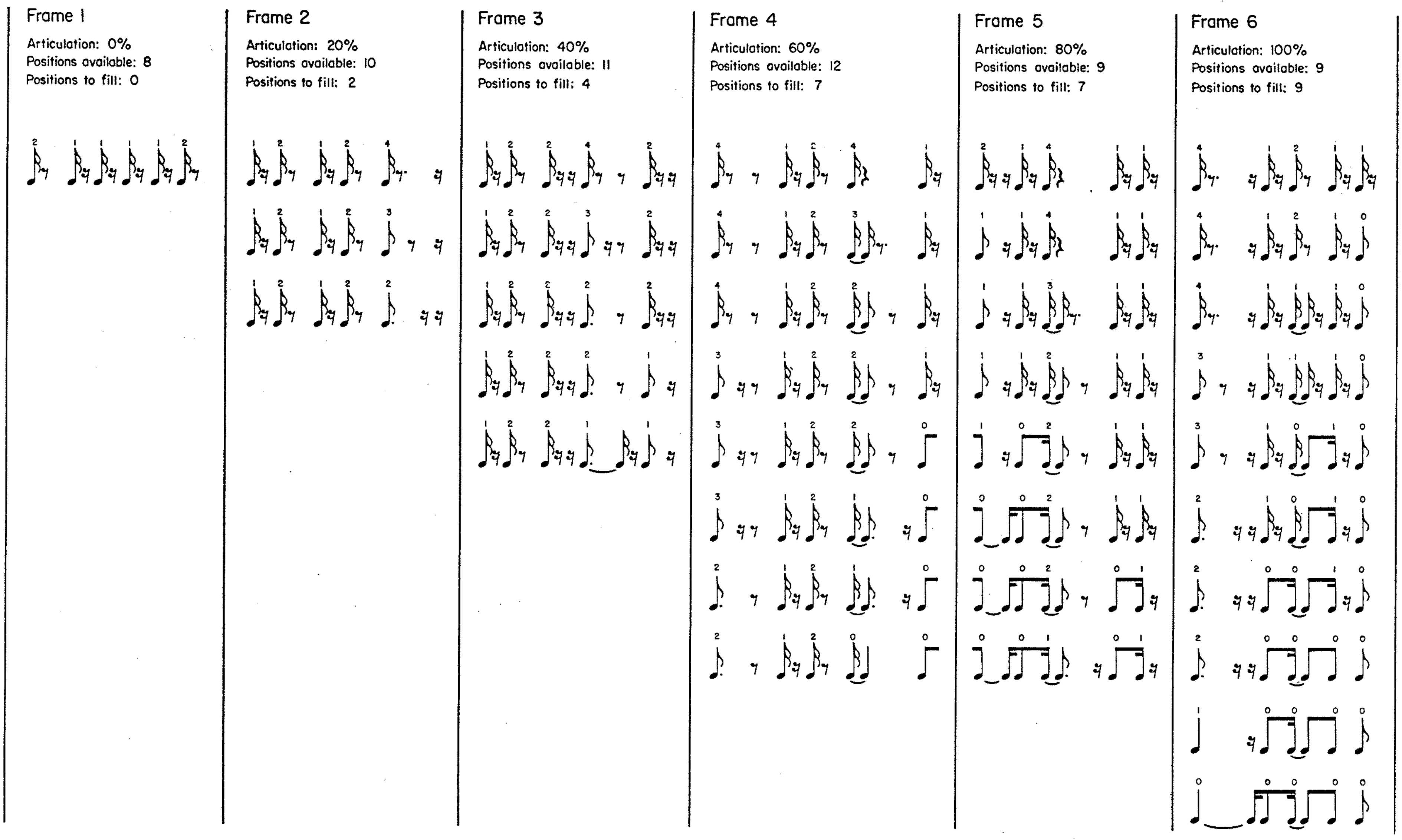

Frame 1
Articulation: 0%
Positions available: 8
Positions to fill: 0
Frame 2
Articulation: 20%
Positions available: 10
Positions to fill: 2
Frame 3
Articulation: 40%
Positions available: 11
Positions to fill: 4
Frame 4
Articulation: 60%
Positions available: 12
Positions to fill: 7
Frame 5
Articulation: 80%
Positions available: 9
Positions to fill: 7
Frame 6
Articulation: 100%
Positions available: 9
Positions to fill: 9

Fig 5-9

determines the number of times RATIO is invoked within each frame.

Figure 5-9: Selection of durations for the melody depicted in Figure 5-8.

5.6.2.3 Stage III: Pitches - Pitches in Figure 5-8 are selected from six options: E4, F4, Gb4, A#4, B4, and C5. The sequence of decisions consists of having RATIO select one pitch for each note, moving from left to right in the melody. The object is to provide equal durational emphasis to each pitch, so since the total of all the durations selected during stage II is 60.0, each of the six pitches receives an initial weight of 10. Table 5-5 chronicles the attrition of the weights allotted to each pitch as RATIO composes the sequence:

Table 5-5: Selecting pitches with sensitivity to duration.

Measure & Beat	Duration	Weights E4	F4	Gb4	A#4	B4	C5	Selected Pitch
1:0	1.	10.	10.	10.	10.	10.	10.	A#4
1:3	1.	10.	10.	10.	9.	10.	10.	E4
1:5	1.	9.	10.	10.	9.	10.	10.	B4
1:7	1.	9.	10.	10.	9.	9.	10.	A#4
2:1	1.	9.	10.	10.	8.	9.	10.	F4
2:3	1.	9.	9.	10.	8.	9.	10.	F4
2:6	1.	9.	8.	10.	8.	9.	10.	C5
3:0	1.	9.	8.	10.	8.	9.	9.	B4
3:3	1.	9.	8.	10.	8.	8.	9.	Gb4
3:5	1.	9.	8.	9.	8.	8.	9.	E4
4:0	3.	8.	8.	9.	8.	8.	9.	Gb4
4:5	1.	8.	8.	6.	8.	8.	9.	B4
4:7	1.	8.	8.	6.	8.	7.	9.	E4
5:2	1.	7.	8.	6.	8.	7.	9.	F4
5:5	4.	7.	7.	6.	8.	7.	9.	F4
6:2	2.	7.	3.	6.	8.	7.	9.	A#4
6:5	3.	7.	3.	6.	6.	7.	9.	B4
7:2	1.	7.	3.	6.	6.	4.	9.	A#4
7:4	1.	7.	3.	6.	5.	4.	9.	C5
7:7	5.	7.	3.	6.	5.	4.	8.	B4
8:4	2.	7.	3.	6.	5.	0.	8.	E4
8:6	3.	5.	3.	6.	5.	0.	8.	C5
9:1	2.	5.	3.	6.	5.	0.	5.	E4
9:3	4.	3.	3.	6.	5.	0.	5.	F4
10:0	2.	3.	0.	6.	5.	0.	5.	C5
10:2	1.	3.	0.	6.	5.	0.	3.	Gb4
10:4	5.	3.	0.	5.	5.	0.	3.	C5
11:1	2.	3.	0.	5.	5.	0.	0.	A#4
11:3	3.	3.	0.	5.	3.	0.	0.	E4
11:6	2.	0.	0.	5.	3.	0.	0.	Gb4
12:0	2.	0.	0.	3.	3.	0.	0.	A#4

Table 5-5

5.7 NOTES

1. While direct random selection also corresponds to drawing balls from an urn in the manner described here, it differs in the crucial respect that after the mark on each ball has been noted, the ball is replaced in the urn and the balls are remixed prior to the next draw.

2. Even when the process of organizing a frame is stylistically conditioned, it is often useful to shuffle a pool before imposing other criteria of organization. This precaution removes any bias inherent in the way samples in the pool are derived.

3. This algorithm comes to the author by way of Donald Knuth (1969, page 125).

4. Schoenberg is best known for applying this restriction to the chromatic scale in his twelve-tone system. However, even in a diatonic context he advises: "Monotony is often produced through too frequent repetition of one tone or of a succession of several tones, and through remaining too long within the range of a fourth or fifth" (Preliminary Exercises in Counterpoint, 1963). As reasonable as this comment of Schoenberg's may seem, it should be taken as an expression of taste rather than an inflexible

dictum. ~~[illegible]~~ Much different tastes guided Stravinsky, for example, when he composed the choral melodies for the finale of his Symphony of Psalms.

5.8 RECOMMENDED READING

Koenig, Gottfried Michael, 1970b. "PROJECT 2: A programme for musical composition", Electronic Music Reports, number 3.

Laske, Otto, 1981. "Composition theory in Koenig's Project One and Project Two", Computer Music Journal, volume 5, number 4, page 54.