CHAPTER 12

COMPARATIVE SEARCH

A search is a process which evaluates many alternate solutions to a problem in order to select one solution meeting some set of criteria. The simple example of looking up a name in a phone book illustrates a single-decision problem: each possible solution corresponds to a single option. A much more complex example of searching is determining a strategy in a game of chess. In this second example, each solution encompasses a sequence of decisions, and each of these decisions may provoke a number of alternate responses from the competing player.

There are a wide variety of algorithms for searching, and as is usual, the most appropriate algorithm varies with the specific application (note 1). By far the most successful approach to solving problems with many decisions has been recursive searching. Such searches were first proposed by Shannon (1950) for "automatic" chess playing; later, they were adapted to logical theorem-proving by computer scientists Alan Newell and J.C. Shaw, in collaboration with psychologist Herbert Simon (1958 a,b; note 2).

In general, a recursive search organizes the decisions of a problem into a well-defined sequence and then constructs a corresponding sequence of partial solutions according the following recursive scheme: the first partial solution consists of the first decision in the sequence; each new partial solution is obtained by advancing to the next decision in the sequence and appending the result to the preceding partial solution. A search through the set of all possible solutions to the problem may then be effected systematically by enumerating every option available to the first decision, enumerating every option available to the second decision given each (partial) solution to the first decision, enumerating every option available to the third decision given each solution to the first two decisions, and so on until every combination of options over all of the decisions involved in the problem has been exhausted.

The present chapter investigates _comparative searches_, which attempt to find _optimal_ solutions to problems encompassing many decisions. The strategy of a comparative search consists of systematically enumerating every possible solution and comparing each new solution, or _candidate_ to the best solution so-far encountered, or _incumbent_. Comparative searches have been used by the author to produce two computer-composed works for solo piano: _Protocol_ (1981; described 1982) and _Gradient_ (1982; described 1983). The advantage of comparative search over all other methods of decision making described in this book is that, given enough time, it always finds the best solution. Its disadvantage is that the number of solutions may be too large for even a computer to evaluate in any reasonable amount of time.

12.1 EXAMPLE: A SEARCH FOR 'OPTIMAL CONSONANCE'

We illustrate the mechanism of comparative search by searching through the F major scale for a four-part chord which is 'optimally consonant' in the sense of having as few or fewer dissonant intervals when compared to any other chord. We further stipulate that for the purposes of this search, each degree in the chord should be different, and judgements of relative consonance should be independent of inversion, voicing, and transposition. Specifically, intervals which are obtainable from one another by displacing a pitch by one or more octaves or by transposing both pitches by an equal number of semitones should be judged equally consonant.

The trick to implementing a comparative search is to organize all items of information necessary to describe each option selected for each decision into parallel arrays.

Solutions may then be enumerated recursively, by treating these arrays as stacks (heading 10.2.2) for which the elements up to the Ith position describe a partial solution up to the Ith decision. The incumbent may be stored in a second set of parallel arrays until the program has run its course; alternately, the sequence of incumbents may be printed for perusal by the program's user.

Program CONCHD enumerates all of the four-part chords in an F major scale, substituting a candidate for the incumbent whenever the candidate is less dissonant. Enumerating all of the four-part chords in an F major scale is equivalent to enumerating all the ways of choosing four numbers out of seven, so CONCHD closely resembles program COMBN2 (heading 10.1) in its mechanism. CONCHD employs the following symbols:

1. The integer variable IPRT gives the current part of the candidate chord for which a scale degree is being considered.

2. Array element SCLPRT(I) gives the degree of the F major scale for the Ith part of the candidate. For reasons of efficiency, SCLPRT(IPRT) is stored in the holding variable ISCL.

3. Array element DEGPRT(I) gives the degree of the chromatic scale starting on F for the Ith part of the candidate. For reasons of efficiency, DEGPRT(IPRT) is stored in the holding variable IDEG.

4. Array DEGSCL converts degrees of the F major scale into degrees of the chromatic scale. The contents of DEGSCL are established by the DATA statement in line 6.

5. Array LETTER holds a two-character mnemonic for each degree of the F major scale.

6. Array element QALPRT(I) holds a packed key (heading 9.1.3) talleying the various types of interval present in first I parts of the candidate. QALPRT assigns greater significances to the more dissonant intervals.

7. Array QALTVL gives relative significances (expressed in powers of 4) for intervals spanning from one to eleven semitones. The contents of QALTVL are fixed at the values specified in line 7.

The variable LEAST holds the minimum value of QALPRT(MPRT) so-far encountered. This value is the only information which CONCHD

12-5a

```
      program CONCHD
C
C     Program for finding an 'optimally consonant' four-part
C     chord by comparative search
C
      parameter (MSCL=7,MPRT=4)
      integer SCLPRT(MPRT),DEGPRT(MPRT),QALPRT(MPRT),
     :        DEGSCL(MSCL),QALTVL(11)
      character*2 LETTER(MSCL)
      data DEGSCL/1,3,5,6,8,10,12/
      data QALTVL/256,16,4,1,0,64,0,1,4,16,256/
      data LEAST/1000000/,LIM/4/,IPRT/1/
      data LETTER/' F',' G',' A','Bb',' C',' D',' E'/
C
      SCLPRT(IPRT) = 0
      do
C       Increment index
        ISCL = SCLPRT(IPRT) + 1
        if (ISCL.lt.LIM+IPRT) then
          SCLPRT(IPRT) = ISCL
          IDEG = DEGSCL(ISCL)
          DEGPRT(IPRT) = IDEG
C         Evaluate contribution of this degree to current chord
          if (IPRT.eq.1) then
            K = 0
          else
            K = QALPRT(IPRT-1)
            do (I=1,IPRT-1)
              K = K + QALTVL(IDEG-DEGPRT(I))
            repeat
          end if
C         Compare current chord to least dissonant chord
C         so-far encountered
          if (K.lt.LEAST) then
C           Current chord is still less dissonant
            if (IPRT.eq.MPRT) then
C             Current chord is complete:  update least dissonant chord
              LEAST = K
              print *, (LETTER(DEGPRT(I)),I=1,MPRT),LEAST
            else
C             Current chord is incomplete:  advance to next part
              QALPRT(IPRT) = K
              IPRT = IPRT + 1
              SCLPRT(IPRT) = ISCL
            end if
          end if
        else
C         Backtrack to previous part
          IPRT = IPRT - 1
          if (IPRT.lt.1) exit
        end if
      repeat
      stop
      end
```

Ex 12-1

needs to retains internally about an incumbent; whenever LEAST exceeds QALPRT(MPRT), the program records the current candidate as incumbent on the output listing, then reduces LEAST to the new minimum value.

-- Programming example 12-1: program CONCHD --

The first task in implementing judgements of relative consonance is to select a base for the packed key. If we consider all the intervallic relationships between adjacent and non-adjacent parts, any four-part chord contains six intervals. However, since any non-adjacent interval must be a combination of adjacent intervals, it can be shown that no single intervallic type may appear more than three times. Program CONCHD therefore uses a base of

$$3 + 1 = 4.$$

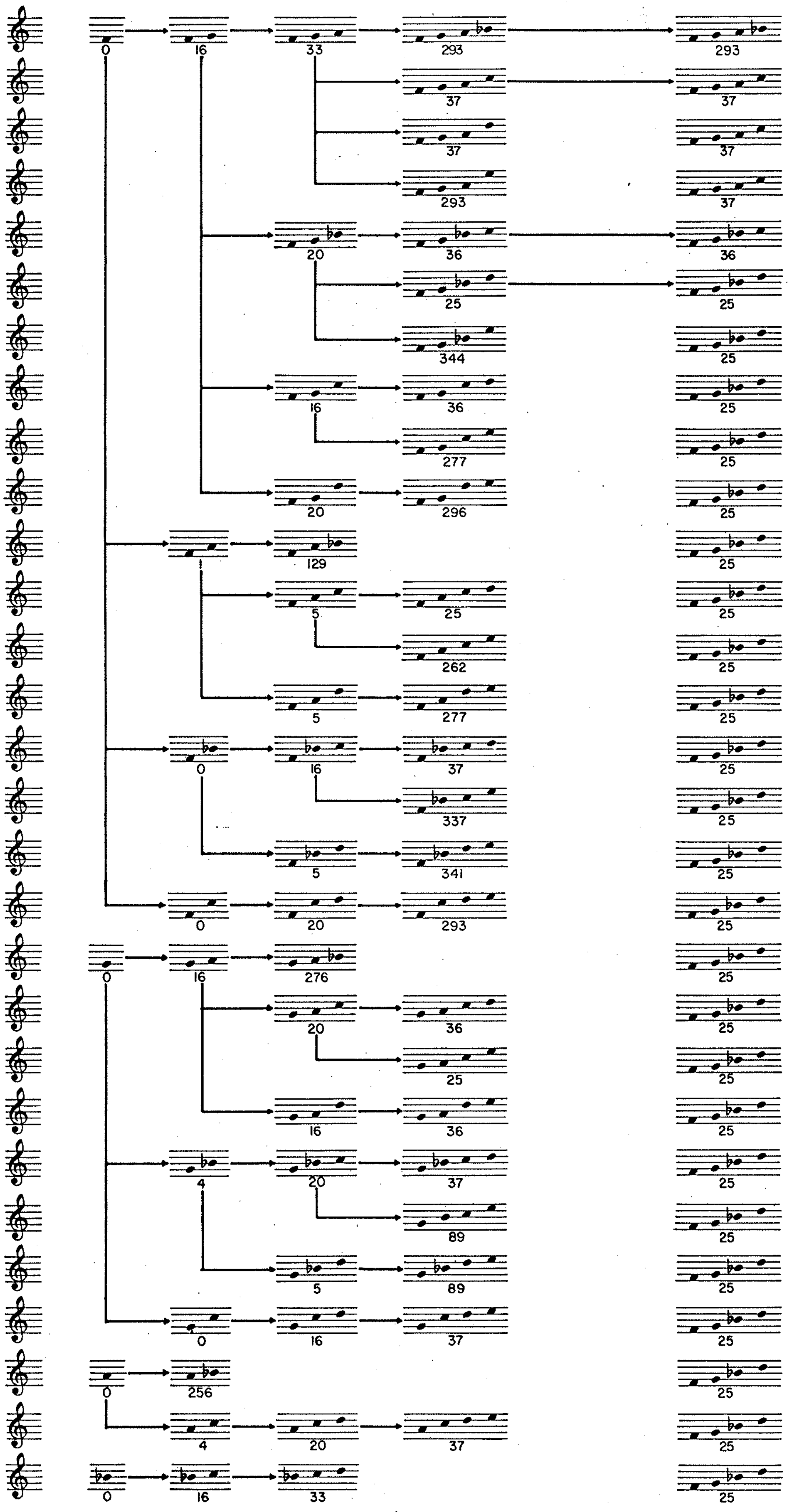
Since none of the chords under consideration contain any doubled chromatic degrees, CONCHD is able to regard the perfect fifth and its inversion, the perfect fourth, as optimally consonant intervals. CONCHD assigns relative significances to each remaining intervallic type as follows (note 3):

Intervallic Type	Significance
major 3rd, minor 6th	$4^0 = 1$
minor 3rd, major 6th	$4^1 = 4$
major 2nd, minor 7th	$4^2 = 16$
tritone	$4^3 = 64$
minor 2nd, major 7th	$4^4 = 256$

Figure 12-1: Chronicle of a comparative search - The leftmost column of single pitches shows choices of the first degree of the 'candidate' chord, while the next three successive columns are each derived by selecting one additional degree and appending it to the partial results obtained in the preceding column. Branching arrows indicate where one choice serves for multiple candidates. The rightmost column shows the 'incumbent' chord.

Figure 12-1 traces the behavior of program CONCHD as it evaluates each combination of four F major degrees. We see that an optimal chord has a packed key of 25, which we may "unpack" to determine the composition of intervals as follows:

12-7a



0
16
33
293
293
37
37
37
37
293
37
20
36
36
25
25
344
25
16
36
25
277
25
20
296
25
1
129
25
5
25
25
262
25
5
277
25
0
16
37
25
337
25
5
341
25
0
20
293
25
0
16
276
25
20
36
25
25
25
16
36
25
4
20
37
25
89
25
5
89
25
0
16
37
25
0
256
25
4
20
37
25
0
16
33
25

Fig 12-1.

Intervallic Types	Contribution
One major second or minor seventh	1 x 16 = 16
Two major thirds or minor sixths	2 x 4 = 8
One minor second or major sixth	1 x 1 = 1
Two perfect fifths or perfect fourths	2 x 0 = 1
	25

Notice that the search illustrated in Figure 12-1 encounters three distinct chords characterized by this same packed key; CONCHD selects F G Bb D solely because the enumerating process encounters this chord first. Notice also that the search backtracks immediately whenever it judges a partial candidate to be less consonant than the full incumbent. This shortcut increases computational efficiency by eliminating unecessary evaluations from the search (note 4). Though it provides relatively little advantage here over a fully exhaustive search, it can substantially reduce searching time in many applications. The shortcut gains effectiveness as the number of decisions increases.

12.2 DETERMINING THE NUMBER OF SOLUTIONS

Before implementing a comparative search, it is good practice to determine its feasibilty by estimating the number of solutions. In general if a problem requires N decisions with k(i) possible options for the ith decision (i ranging from 1 to N), then the number of solutions, K, is given by Equation 12-1:

$$K = k(1) \times k(2) \times k(3) \times \ldots \times k(N) \qquad \text{(Equation 12-1)}$$

For example, in a problem requiring 5 decisions with 4 options for the first decision, 3 options for the second, 7 options for the third, 10 options for the fourth, and 2 options for the last, the number of possible solutions is:

$$4 \times 3 \times 7 \times 10 \times 2 = 1680$$

Table 12-1 details values of Equation 12-1 for a number of important special cases, which we shall examine specifically under the next few headings.

A. $K = M^N$

						N				
M	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	64	128	256	512	1024
3	3	9	27	81	243	729	2187	6561	19683	59049
4	4	16	64	256	1024	4096	16384	65536	262144	1048576
5	5	25	125	625	3125	15625	78125	390625	1953125	9765625
6	6	36	216	1296	7776	46656	279936	1679616	10077696	60466176
7	7	49	343	2401	16807	117649	823543	5764801	40353607	282475249
8	8	64	512	4096	32768	262144	2097152	16777216	134217728	1073741824
9	9	81	729	6561	59049	531441	4782969	43046721	387420489	3486784401
10	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000

B. $K = \frac{M!}{(M-N)!}$

						N				
M	1	2	3	4	5	6	7	8	9	10
1	1									
2	2	2								
3	3	6	6							
4	4	12	24	24						
5	5	20	60	120	120					
6	6	30	120	360	720	720				
7	7	42	210	840	2520	5040	5040			
8	8	56	336	1680	6720	20160	40320	40320		
9	9	72	504	3024	15120	60480	181440	362880	362880	
10	10	90	720	5040	30240	151200	604800	1814400	3628800	3628800

C. $K = \frac{M!}{N!(M-N)!}$

					N					
M	1	2	3	4	5	6	7	8	9	10
1	1									
2	2	1								
3	3	3	1							
4	4	6	4	1						
5	5	10	10	5	1					
6	6	15	20	15	6	1				
7	7	21	35	35	21	7	1			
8	8	28	56	70	56	28	2	1		
9	9	36	84	126	126	84	36	9	1	
10	10	45	120	210	252	210	120	45	10	1

D. $K = \frac{(M+N-1)!}{N!(M-1)!}$

					N					
M	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	3	4	5	6	7	8	9	10	11
3	3	6	10	15	21	28	36	45	55	66
4	4	10	20	35	56	84	120	165	220	286
5	5	15	35	70	126	210	330	495	715	1001
6	6	21	56	126	252	462	792	1287	2002	3003
7	7	28	84	210	462	924	1716	3003	5005	8008
8	8	36	120	330	792	1716	3432	6435	11440	19448
9	9	45	165	495	1287	3003	6435	12870	24310	43758
10	10	55	220	715	2002	5005	11440	24310	48620	92378

Table 12-1

Table 12-1: Special cases of Equation 12-1 - K denotes the number of solutions to N decisions drawing from a pool of M options. A) ordered with replacement; B) ordered without replacement; C) unordered without replacement; D) unordered with replacement.

12.2.1 Decisions with a Constant Number of Options

Often the number of options remains constant for each decision. Given N decisions, each with M options, Equation 12-2 yields the number of solutions, K. Table 12-1A gives values of K for each N and M up to 10.

$$K = M^N \qquad \text{(Equation 12-2)}$$

Though the number of options must remain fixed at N for Equation 12-2 to apply, the specific options in themselves may vary. Suppose, for example, that one wishes to compose a seven-part chord, with seven possible pitches for each part. Table 12-1A indicates that there would be 823,542 candidates for each chord.

12.2.2 Selecting Options From a Common Pool

Many applications require all options to be drawn from a common pool. In such cases the number of solutions is strongly affected by two important criteria:

1. When selecting an option eliminates it from the pool, we say that each selection is accomplished "without replacement"; otherwise, we say it is accomplished "with replacement". For example, if one employs a comparative search to select degrees of a chord from a scale, then whether selection occurs with or without replacement depends on whether or not it is permissible to double degrees of the chord. Selection without replacement admits many fewer solutions than selection with replacement.

2. Whenever any two solutions are considered equivalent if they embrace the same set of options but in different arrangements, we call the process of selection "unordered"; when two such solutions are considered

different, we call the process "ordered". Returning to the example of selecting degrees of a chord from a scale, then whether the selection is "ordered" or "unordered" depends on whether or not selecting a degree assigns it to a specific part (e.g., melody, inner part, bass). Unordered selection admits many fewer solutions than ordered selection.

Program CONCHD (heading 12.1) provides a basic model for all of the varieties of comparative search discussed under the present heading. In CONCHD, the Ith "decision" consists of selecting a degree for the Ith part in a chord. Array DEGSCL stores a common pool of options, in this case, degrees of the F major scale, while array SCLPRT holds the index to the option under consideration by each decision. We now consider how CONCHD may be adapted to the various conditions of replacement and order described above:

1. As it stands, CONCHD implements <u>unordered selection without replacement</u>. Notice that upon advancing to the Ith part (line 40), the program initializes SCLPRT(I) to SCLPRT(I-1). Since the program immediately increments SCLPRT(I) with the next iteration of the loop (line 14), this practice insures that the sequence of scale degrees

always ascends without repetitions.

2. Adapting the program to unordered selection with replacement would involve initializing SCLPRT(I) to SCLPRT(I-1)-1 upon advancment to the Ith part. With this modification the sequence of scale degrees would still ascend, but immediate repetitions of degrees would be admitted.

3. Ordered selection with replacement would require initializing SCLPRT(I) to zero upon advancment to the Ith part. This modification would admit. any sequence of scale degrees.

4. Finally, converting CONCHD to ordered selection without replacement would require initializing SCLPRT(I) to zero upon advancement to the Ith part, plus a constraint against any degree already under consideration by an earlier part. This modification would admit any non-repeating sequence of scale degrees.

Given a process which selects N options in order from a pool containing M elements, Equation 12-2 applies if selection occurs with replacement. If selection occurs without replacement, then N may not exceed M, and Equation 12-3 yields the number of solutions, K. The case of a ordered selection without replacement with N=M corresponds to a comparative search used to organize the resources allotted to a statistical frame. Table 12-1B gives values of K for each M up to 10 and each N not exceeding M.

$$K = \frac{M!}{(M-N)!} \qquad \text{(Equation 12-3)}$$

For example, suppose we wish to determine how many four-part chords may be drawn from a seven-note scale if we take into account the degree selected for each part. This is a process of ordered selection with M=7 and N=4. If we allow doublings of degrees, then selections occur with replacement, so Table 12-1A indicates that the number of potential chords is 2401. If we do not allow doublings, then selections occur without replacement, so Table 12-1B indicates 840 chords.

Given a process which selects N unordered options without replacement from a pool containing M elements, then N may not exceed M, and Equation 12-4 yields the number of solutions, K. Table 12-1C gives values of K for each M up to 10 and each N not exceeding M.

$$K = \frac{M!}{N!(M-N)!} \qquad \text{(Equation 12-4)}$$

For example, suppose we wish to determine how many distinct four-part chords with no doublings may be drawn (without regard to inversion or voicing) from a seven-note scale. Then we have M=7 and N=4, so consulting Table 12-1C tells us that there are 35 such chords.

Given a process which selects drawing N unordered options from a pool containing M elements with replacement, then Equation 12-5 yields the number of solutions, K. Table 12-1D gives values of K for each M and N up to 10.

$$K = \frac{(M+N-1)!}{N!(M-1)!} \qquad \text{(Equation 12-5)}$$

For example, suppose we again wish to determine how many distinct four-part chords may be drawn without regard to inversion or voicing from a seven-note scale, but this time wish to allow doublings. Then we again have M=7 and N=4, but consulting Table 12-1D tells us that the number of potential chords increases to 84.

12.3 DEMONSTRATION 10: COMPARATIVE SEARCH

Demonstration 10 illustrates how comparative searches can be used to compose a piece of music. Demonstration 10 reverses the approach taken with Demonstrations 8 and 9, for which musical details were deduced from the 'top down', as consequences of the musical design: the form of Demonstration 10 is induced from the 'bottom up' up, on the basis of qualities inherent in previously composed material. The piece reflects the author's composition Protocol to the extent that Demonstration 10's material consists of recurrent modules, each comprised of several chords whose order of progression is contextually determined. As with Demonstration 7, Demonstration 10 employs techniques of 'implied polyphony' to adapt this chordal material to the monophonic nature of the clarinet.

12.3.1 Compositional Directives

The ~~act~~ process of composing Demonstration 10 divides into three stages of production:

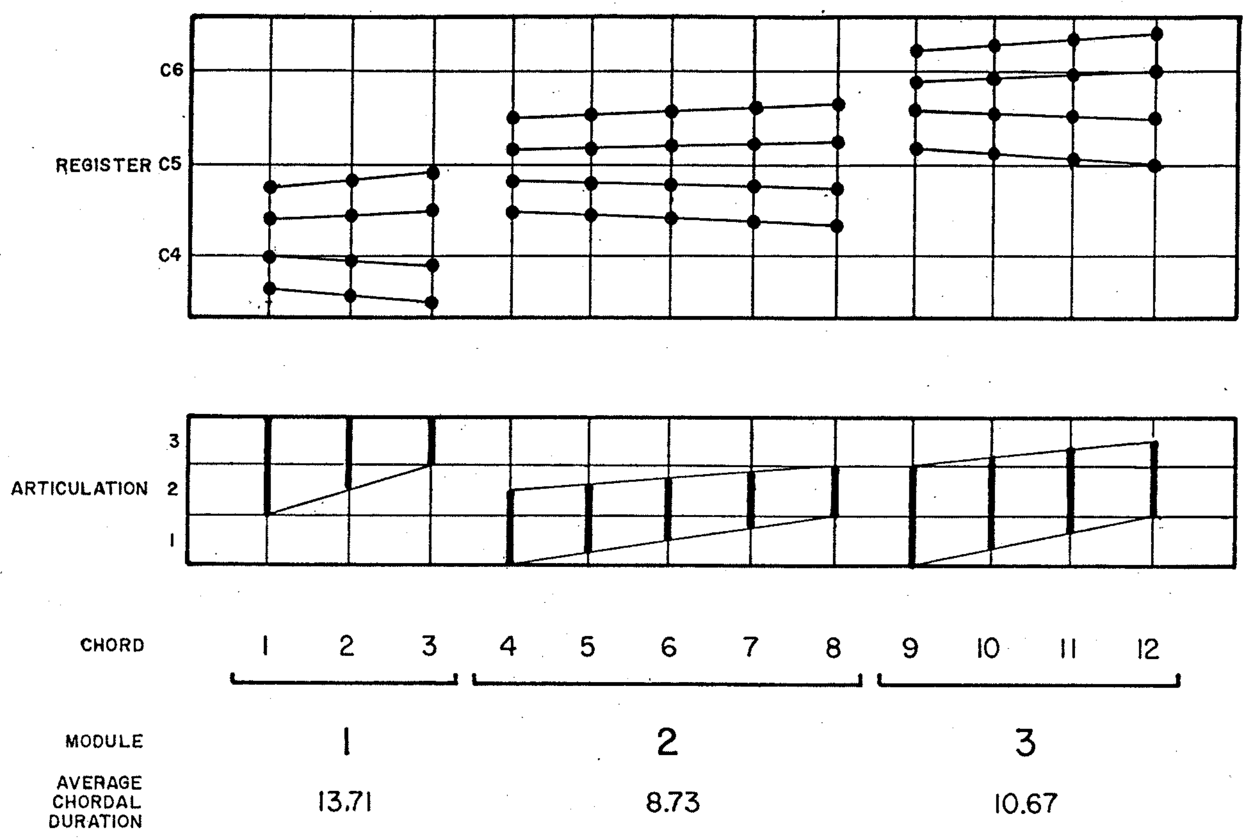
1. Stage I: ~~Chords~~ Material - This stage takes the registral information shown in Figure 12-2 and composes the 12 chords depicted in Figure 12-3.

2. Stage II: Form - This stage steps through each of the segments described in Table 12-2 and arranges its chords into the progression detailed in Figure 12-4.

3. Stage III: Content - The third stage takes the results of PROGRS along with the ranges of articulation shown in Figure 12-2 to arpeggiate the progression of chords in a manner playable on a clarinet.

The final result appears in Figure 12-5.

12.3.1.1 ~~Chords~~ Material - As shown in Figure 12-2, the twelve chords of Demonstration 10 combine into three 'modules'. Each of these modules is unified by a characteristic register, by a range of articulations, and by an average chordal duration which holds for of the module as a whole.



C6
REGISTER C5
C4
ARTICULATION
3
2
1
CHORD
1 2 3 4 5 6 7 8 9 10 11 12
MODULE
1
2
3
AVERAGE CHORDAL DURATION
13.71
8.73
10.67

Fig 12-2

Figure 12-2: Compositional data for Demonstration 10.— Dots on the registral graph indicate center pitches of a five-semitone gamut available to each part in each chord. The bold vertical lines on the graph of articulations define regions of relative likelihood for three styles of playing: 1) slurred, 2) normal, and 3) detached. The average chordal duration supplied for each module indicates sixteenth notes.

Stage I of the composing process seeks to reconcile directives intended to promote chromatic diversity, dissonance, and divergent part-leading. The most emphatic of these directives assume the status of constraints:

1. As a minimum condition of chromatic diversity, no two chords may share more than two degrees of the chromatic scale.

2. As a minimum condition of dissonance, no two pitches in a chord may occupy the same degree of the chromatic scale.

3. Parallelisms are restricted in the following way:

a. For any pair of chords residing in the same module, the interval relating any two parts in one chord may not be duplicated in the corresponding two parts of the other chord.

b. For any pair of chords residing in different modules, the intervals relating any three parts one chord may not be duplicated in the corresponding three parts of the other chord.

Each chord is composed by a comparative search; backtracking is limited to the extent that once the program advances to a new chord, no revision of any previous chord takes place. Subject to the constraints listed above, each search employs two criteria for evaluating candidates:

1. <u>Chromatic redundancy</u>: For each degree in a chord, the program tallies how many times the degree appears in previously composed chords. The maximum of these tallies yields a worst-case measure of the chord's 'chromatic redundancy'.

2. <u>Sonorous quality</u>: The program derives this number by first tallying the number of times each type of interval

occurs in a chord and then packing these tallies into a single number which assigns the greater significances to the more consonant intervals.

Less 'redundant' candidates displace incumbents; more 'redundant' candidates are discarded. When candidates and incumbents are equally 'redundant', then the candidate displaces the incumbent only when the candidate's value of 'sonorous quality' is lower. Figure 12-3 shows the results of the twelve chordal searches. Notice that the formulation of 'sonorous quality' favors chords of the diminished seventh (chords 2 and 4) over chords of superimposed fourths, fifths, and tritones (especially, chord 8, but see also chords 7 and 11).

Figure 12-3: The 12 chords - The two "keys" supplied beneath each chord indicate 1) 'chromatic redundancy' relative to the preceeding chords and 2) 'sonorous quality', expressed as packed tallies of intervallic types, with the greater significances associated with the more consonant intervals.

With 5 pitches available to each part and 4 parts, Equation 12-2 applies to indicate a theoretic total of 625 chords to choose from. In practice, the constraints against doublings,

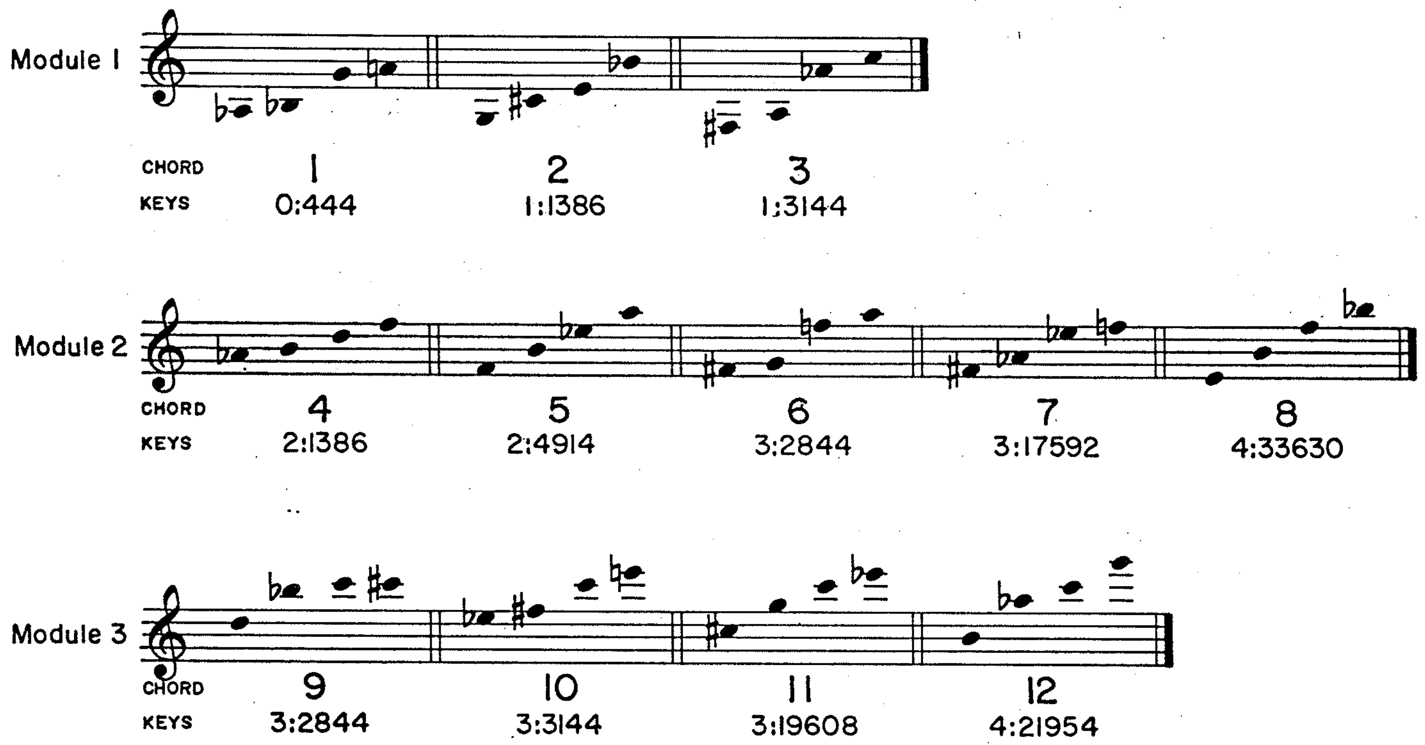

Module 1
CHORD
KEYS
1
0:444
2
1:1386
3
1:3144
Module 2
CHORD
KEYS
4
2:1386
5
2:4914
6
3:2844
7
3:17592
8
4:33630
Module 3
CHORD
KEYS
9
3:2844
10
3:3144
11
3:19608
12
4:21954

Fig 12-3

parallelism, and chromatic overlap substantially reduce the number of acceptable chords. Only when a partial chord meets the absolute requirements imposed by these constraints does ~~CHORDS~~ a search proceed to evaluate its 'chromatic redundancy' and 'sonorous quality'.

12.3.1.2 Form - The piece divides into twelve segments, each of which draws material from a single module. Table 12-2 details descriptive information for each segment. The number of chords appearing in a segment depends upon both the length of the segment and the module's average chordal duration:

Table 12-2: Formal layout of Demonstration 10.

Given the chordal content of a segment (note 5), Stage II of the composing process is responsible for determining in what order the chords will occur. This problem is fundamentally one of organizing a statistical frame; where programs for earlier demonstrations would have either shuffled the chords randomly (chapter 5) or sorted the chords so that preferred chords received preferred the more desirable positions (chapter 9), the technique of comparative search enables the current program to be

Segment	Measure & Beat	Duration	Module	Number of Chords
1	1:0	69	1	5
2	9:5	52	2	6
3	16:1	52	1	4
4	22:5	25	3	3
5	26:5	53	2	6
6	33:2	42	3	4
7	38:4	45	2	5
8	44:1	48	1	3
9	50:1	45	3	4
10	55:6	35	1	3
11	60:1	42	2	5
12	65:3	61	3	6

Table 12-2

sensitive to context when it considers alternate chordal progressions.

By contrast to the chord-composing searches, the searches for 'optimal' chordal progressions impose no constraints. Two criteria serve to evaluate candidates:

1. '<u>Blandness</u>': For each progression of chords, the program tallies how many pairs of consecutive chords share 0, 1, 2, or 4 chromatic degrees in common (note 6). It then packs these tallies into a single key, assigning greater significances to consecutive chords sharing more degrees. The resulting quantity gives a measure of 'blandness' (note 7) in the progression.

2. '<u>Balance</u>': In order to keep the chordal resources in balance, the program maintains cumulative statistics reflecting the total duration over which each chord has been employed. The 'balance' of a progression is a vector (note 8) which is determined by considering the largest statistic as a primary key, the next largest statistic as a secondary key, and so on for each statistic.

The primary goal of the search is to arrange the chords in such a manner that 'blandness' will be minimal. Given progressions which meet this condition, a subsidiary goal is to arrange the chords so as to assign longer durations to those chords which occur least often in a segment. Random shuffling insures an unbiased selection between progressions which the program otherwise judges equally suitable.

Figure 12-4: Profile of Demonstration 10 - Module and chord numbers refer to Figures 12-2 and 12-3. The 'blandness' given for each segment reflects pairs of consecutive chords sharing one or more common chromatic degrees.

The number of chords in any one segment ranges from 3 to 6. Since the process of selecting chords for positions is accomplished with regard to order but without replacement and since the size of the pool is in each case equal to the number of positions, Equation 12-4 tells us that the number of possible arrangements varies from 6 to 720.

Figure 12-5: Transcription of Demonstration 10.

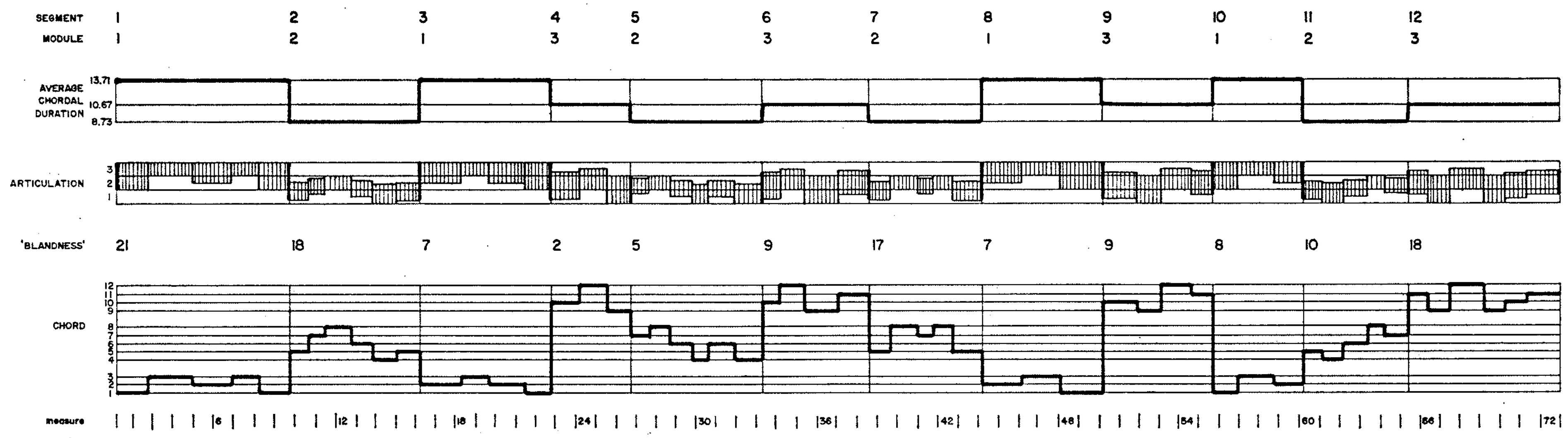


Fig 12-4

12-23a

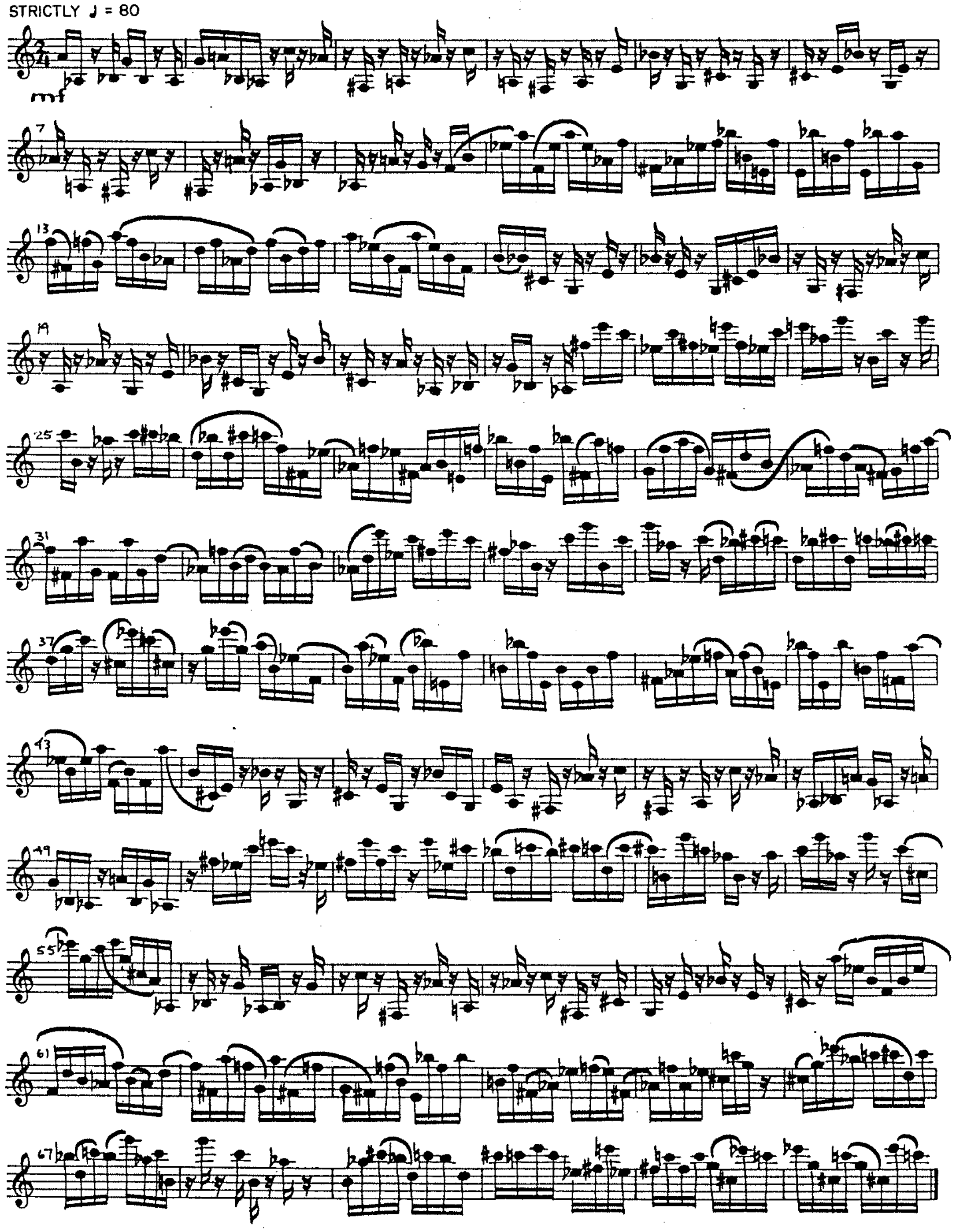
12-23b

Demonstration 10

Clarinet

Charles AMES

STRICTLY ♩ = 80



© Charles Ames 1984

Fig 12-5

12-23c

```
      program DEMO10
C
C     Demonstration of comparative search
C
      parameter (MSEC=12,MPRG=54,MCHD=12,MPRT=4)
      integer PCHPRT(MPRT,MCHD),DEGPRT(MPRT,MCHD),NUMSEC(MSEC),
     :        PRGSEC(MSEC),DURPRG(MPRG),POLPRG(MPRG),CHDPRG(MPRG)
      common PCHPRT,DEGPRT,NUMSEC,PRGSEC,DURPRG,POLPRG,CHDPRG
C
      open (2,file='DEMO10.DAT',status='NEW')
      call CHORDS
      call PROGRS
      call ARPEGG
      close (2)
      stop
      end

      subroutine CHORDS
      parameter (MSEC=12,MPRG=54,MCHD=12,MPRT=4,MPCH=5)
      integer PCHPRT(MPRT,MCHD),DEGPRT(MPRT,MCHD),NUMSEC(MSEC),
     :        PRGSEC(MSEC),DURPRG(MPRG),POLPRG(MPRG),CHDPRG(MPRG)
      integer MODCHD(MCHD),IDXPRT(MPRT),REGPRT(MPRT,MCHD),
     :        PCHTMP(MPRT),DEGTMP(MPRT),USETMP(MPRT),QALTMP(MPRT),
     :        OFFSET(MPCH),QALTVL(11),USEDEG(12)
      logical PARALL
      common PCHPRT,DEGPRT,NUMSEC,PRGSEC,DURPRG,POLPRG,CHDPRG
      data MODCHD/1,1,1,2,2,2,2,2,3,3,3,3/
      data OFFSET/0,1,-1,2,-2/
      data QALTVL/1,49,343,2401,16807,7,16807,2401,343,49,1/
      data USEDEG/0,0,0,0,0,0,0,0,0,0,0,0/
      data REGPRT/44,48,53,57, 43,48,54,58, 42,47,54,59,
     :            54,58,62,66, 54,58,62,67, 53,57,63,67,
     :            52,57,63,67, 52,57,63,68, 62,68,71,75,
     :            61,67,71,76, 61,67,72,76, 60,66,72,77/
C
      do (ICHD=1,MCHD)
        LUSE = 10000000
        LQAL = 10000000
        IPRT = 1
        IDXPRT(IPRT) = 0
        do
C         Increment index
          IDX = IDXPRT(IPRT) + 1
          if (IDX.le.MPCH) then
            IDXPRT(IPRT) = IDX
            IPCH = REGPRT(IPRT,ICHD) + OFFSET(IDX)
            IDEG = MOD(IPCH,12) + 1
            PCHTMP(IPRT) = IPCH
            DEGTMP(IPRT) = IDEG
C           Test for duplicate degrees
            do (I=1,IPRT-1)
              if (DEGTMP(I).eq.IDEG) go to 47
            repeat
C           Test for parallelisms
            if (IPRT.ge.2) then
              do (LCHD=1,ICHD-1)
                if (MODCHD(LCHD).eq.MODCHD(ICHD)) then
                  if (PARALL(PCHPRT(1,LCHD),PCHTMP(1),IPRT,2,FLAG)) go to 47
                else if (IPRT.ge.3) then
                  if (PARALL(PCHPRT(1,LCHD),PCHTMP(1),IPRT,3,FLAG)) go to 47
                end if
C               Test that current chord contains no more than
C               two degrees in common with any previous chord
                if (IPRT.ge.3) then
                  K = 0
                  do (I=1,IPRT)
                    do (J=1,MPRT)
                      if (DEGTMP(I).eq.DEGPRT(J,LCHD)) then
                        K = K + 1
                        exit
                      end if
                    repeat
                  repeat
                  if (K.ge.3) go to 47
                end if
              repeat
            end if
C           Evaluate contribution of this degree to chromatic redundancy
            if (IPRT.eq.1) then
              USETMP(IPRT) = USEDEG(IDEG)
            else
              USETMP(IPRT) = max0(USETMP(IPRT-1),USEDEG(IDEG))
            end if
```

Ex 12-2 (page 1 of 4)

```
C           Compare maximum chromatic redundancy of current chord to
C           that of best chord so-far obtained
            if (USETMP(IPRT).le.LUSE) then
C             Evaluate contribution of this degree to sonorous quality
C             of current chord
              if (IPRT.eq.1) then
                QALTMP(IPRT) = 0
              else
                K = QALTMP(IPRT-1)
                do (I=1,IPRT-1)
                  INTRVL = IDEG-DEGTMP(I)
                  if (INTRVL.le.0) INTRVL = INTRVL + 12
                  K = K + QALTVL(INTRVL)
                repeat
                QALTMP(IPRT) = K
              end if
C             Compare sonorous quality of current chord to that of best
C             chord so-far encountered
              if (QALTMP(IPRT).lt.LQAL) then
C               Current chord still has fewer consonances
                if (IPRT.eq.MPRT) then
C                 Current chord is complete:
C                 Update best chord
                  do (I=1,MPRT)
                    DEGPRT(I,ICHD) = DEGTMP(I)
                    PCHPRT(I,ICHD) = PCHTMP(I)
                  repeat
                  LUSE = USETMP(IPRT)
                  LQAL = QALTMP(IPRT)
                else
C                 Current chord is incomplete:  advance to next part
                  IPRT = IPRT + 1
                  IDXPRT(IPRT) = 0
                end if
              end if
            end if
          else
C           Backtrack to previous part
            IPRT = IPRT - 1
            if (IPRT.lt.1) exit
          end if
  47      continue
        repeat
C       Update usage of chromatic degrees
        do (I=1,MPRT)
          IDEG = DEGPRT(I,ICHD)
          USEDEG(IDEG) = USEDEG(IDEG) + 1
        repeat
      repeat
      return
      end

      function PARALL(CHD1,CHD2,NPRT,NLEV)
      integer CHD1(4),CHD2(4),PRTTMP(4)
      logical PARALL
C
      LIM = NPRT - NLEV
      ITMP = 1
      PRTTMP(ITMP) = 0
      do
        IPRT = PRTTMP(ITMP) + 1
        if (IPRT.le.LIM+ITMP) then
          PRTTMP(ITMP) = IPRT
          if (CHD1(NPRT) - CHD1(IPRT)
     :        .eq. CHD2(NPRT) - CHD2(IPRT)) then
            ITMP = ITMP + 1
            if (ITMP.eq.NLEV) then
              PARALL = .true.
              return
            else
              PRTTMP(ITMP) = IPRT
            end if
          end if
        else
          if (ITMP.eq.1) exit
          ITMP = ITMP - 1
        end if
      repeat
      PARALL = .false.
      return
      end
```

Ex. 12-2 (page 2 of 4)

```
  1              subroutine PROGRS
  2              parameter (MSEC=12,MPRG=54,MCHD=12,MPRT=4,MTMP=6)
  3              integer PCHPRT(MPRT,MCHD),DEGPRT(MPRT,MCHD),NUMSEC(MSEC),
  4             :        PRGSEC(MSEC),DURPRG(MPRG),POLPRG(MPRG),CHDPRG(MPRG)
  5              integer PRGTMP(MTMP),CHDTMP(MTMP),BLNTMP(MTMP),BLNCHD(MCHD,MCHD),
  6             :        SCDTMP(MCHD),SCDCHD(MCHD),USETMP(MCHD),USECHD(MCHD)
  7              common PCHPRT,DEGPRT,NUMSEC,PRGSEC,DURPRG,POLPRG,CHDPRG
  8              data SCDTMP/1,2,3,4,5,6,7,8,9,10,11,12/
  9              data SCDCHD/1,2,3,4,5,6,7,8,9,10,11,12/
 10              data USETMP/0,0,0,0,0,0,0,0,0,0,0,0/
 11              data USECHD/0,0,0,0,0,0,0,0,0,0,0,0/
 12        C
 13        C     Analyze chords for common chromatic degrees
 14              do (ICHD=1,MCHD)
 15                do (LCHD=1,MCHD)
 16                  K = 0
 17                  do (IPRT=1,MPRT)
 18                    IDEG = DEGPRT(IPRT,ICHD)
 19                    do (LPRT=1,MPRT)
 20                      if (IDEG.eq.DEGPRT(LPRT,LCHD)) then
 21                        K = K + 1
 22                        exit
 23                      end if
 24                    repeat
 25                  repeat
 26                  BLNCHD(LCHD,ICHD) = K
 27                repeat
 28              repeat
 29        C
 30        C     Arrange chords of each module into best progression
 31              ICHDO = 0
 32              do (ISEC=1,MSEC)
 33                LPRG1 = PRGSEC(ISEC)
 34                NUM = NUMSEC(ISEC)
 35                call SHUFLE(POLPRG(LPRG1),NUM)
 36                LPRG2 = LPRG1 + NUM
 37                ITMP = 1
 38                LPRG = LPRG1
 39                PRGTMP(ITMP) = LPRG1 - 1
 40                LBLN = 10000000
 41                USETMP(SCDTMP(1)) = 10000000
 42                do
 43                  IPRG = PRGTMP(ITMP) + 1
 44                  if (IPRG.lt.LPRG2) then
 45                    PRGTMP(ITMP) = IPRG
 46        C           Determine chord
 47                    ICHD = POLPRG(IPRG)
 48                    CHDTMP(ITMP) = ICHD
 49                    USECHD(ICHD) = USECHD(ICHD) + DURPRG(LPRG)
 50        C           Insure selection without replacement
 51                    do (I=1,ITMP-1)
 52                      if (PRGTMP(I).eq.IPRG) go to 47
 53                    repeat
 54        C           Evaluate common-tone linkages in current progression
 55                    if (ITMP.eq.1) then
 56                      if (ICHDO.gt.O) BLNTMP(ITMP) = 7**(BLNCHD(ICHD,ICHDO))
 57                    else
 58                      BLNTMP(ITMP) = BLNTMP(ITMP-1)
 59             :                       + 7**(BLNCHD(ICHD,CHDTMP(ITMP-1))-1)
 60                    end if
 61        C           Compare current progression to best progression
 62        C           so-far encountered
 63                    if (BLNTMP(ITMP).le.LBLN) then
 64        C             Progression still has fewer common-degree linkages
 65                      if (ITMP.ge.NUM) then
 66        C               Progression is complete
 67                        call LSORT(SCDCHD,USECHD,MCHD)
 68                        if (BLNTMP(ITMP).eq.LBLN) then
 69        C                 Same common-tone linkages:  compare progressions
 70        C                 for relative usage
 71                          do (LCHD=1,MCHD)
 72                            LUO = USETMP(SCDTMP(LCHD))
 73                            LU1 = USECHD(SCDCHD(LCHD))
 74                            if (LUO.gt.LU1) exit
 75                            if (LUO.lt.LU1) go to 47
 76                          repeat
 77                          if (LCHD.gt.MCHD) go to 47
 78                        end if
 79                        L = LPRG1
 80                        do (I=1,NUM)
 81                          CHDPRG(L) = CHDTMP(I)
 82                          L = L + 1
 83                        repeat
```

Ex 12-2 (page 3 of 4)

```
                do (LCHD=1,MCHD)
                  USETMP(LCHD) = USECHD(LCHD)
                  SCDTMP(LCHD) = SCDCHD(LCHD)
                repeat
                LBLN = BLNTMP(ITMP)
              else
C               Progression is still incomplete:  advance to next chord
                ITMP = ITMP + 1
                LPRG = LPRG + 1
                PRGTMP(ITMP) = LPRG1 - 1
                go to 49
              end if
            end if
          else
            if (ITMP.eq.1) exit
            ITMP = ITMP - 1
            LPRG = LPRG - 1
            ICHD = CHDTMP(ITMP)
          end if
   47     USECHD(ICHD) = USECHD(ICHD) - DURPRG(LPRG)
   49     continue
        repeat
        do (LCHD=1,MCHD)
          USECHD(LCHD) = USETMP(LCHD)
          SCDCHD(LCHD) = SCDTMP(LCHD)
        repeat
        ICHDO = CHDPRG(LPRG2-1)
      repeat
      return
      end

      block data
      parameter (MSEC=12,MPRG=54,MCHD=12,MPRT=4)
      integer PCHPRT(MPRT,MCHD),DEGPRT(MPRT,MCHD),NUMSEC(MSEC),
     :        PRGSEC(MSEC),DURPRG(MPRG),POLPRG(MPRG),CHDPRG(MPRG)
      common PCHPRT,DEGPRT,NUMSEC,PRGSEC,DURPRG,POLPRG,CHDPRG
C
      data NUMSEC/5,6,4,3,6,4,5,3,4,3,5,6/
      data PRGSEC/1,6,12,16,19,25,29,34,37,41,44,49/
      data DURPRG/13,18,16,10,12, 8,6,11,8,10,9, 17,11,14,10,
     :          12,11,9, 8,8,9,6,11,11, 7,10,14,11, 9,11,6,8,11,
     :          16,15,17, 14,10,12,9, 10,14,11, 8,8,10,7,9,
     :          8,9,14,8,9,13/
      data POLPRG/1,1,2,3,3, 4,5,5,6,7,8, 1,2,2,3, 9,10,12,
     :          4,4,6,6,7,8, 9,10,11,12, 5,5,7,8,8, 1,2,3,
     :          9,10,11,12, 1,2,3, 4,5,6,7,8, 9,9,10,11,11,12/
      end
```

12.3.2 Implementation

-- Programming example 12-2: program DEMO10 (4 pages) --

The comparative searches in program DEMO10, reproduced here as example 12-2 (note 9), generalize the procedures employed by program CONCHD (heading 12-1): since several items of information are necessary to describe each option selected by a decision, these items are organized into parallel stacks accessible by a single index.

The symbols of program DEMO10 and its subroutines adhere to five mnemonic "roots" pertaining to the various roles played by information in the musical design:

1. The mnemonic root CHD signifies information pertaining to entire chords. The parameter MCHD gives the number of chords, while the variable ICHD indicates specific chords. In subroutine CHORDS, array element MODCHD(I) indicates which module contains the Ith chord.

2. The mnemonic root PRT signifies information pertaining

to individual parts within a chord. The parameter MPRT gives the number of parts, while the recursive index IPRT indicates specific parts. Array element PCHPRT(I,J) gives the pitch selected by CHORDS for the Ith part of the Jth chord, while element DEGPRT(I,J) gives the corresponding chromatic degree.

3. The mnemonic root SEG signifies information pertaining to individual segments. In particular, the parameter MSEG gives the number of segments, while the variable ISEG indicates the segment currently being composed. Array element NUMSEG(I) gives the number of chordal positions in the Ith segment; array element PRGSEG(I) indicates the first position in the progression occuring in the Ith segment.

4. The mnemonic root PRG signifies information pertaining to the progression of chords. The parameter MPRG gives the total number of chords in all segments combined, while the recursive index IPRG indicates specific positions in the progression. Array element DURPRG(I) gives the duration allotted to the Ith position; array element CHDPRG(I) indicates which of the 12 chords occupies this position. Array POLCHD contains pools of

chords to be used in each segment, stored in the following way: for the Ith segment of the piece, set IPRG=PRGSEG(I) and NUM=NUMSEG(I); then array elements POLCHD(IPRG) through POLCHD(IPRG+NUM-1) hold the appropriate pool of chords.

12.3.2.1 Chords - The outermost loop of subroutine CHORDS (lines 18-115) iterates on the variable ICHD, so as to initiate comparative searches for each of the 12 chords.

The next-to-outermost loop (lines 23-114) provides the recursive mechanism for each search. The level of recursion corresponds to the current part, IPRT; array element IDXPRT(I) gives the index to the option under consideration for the Ith part. In this case, options consist of the registral locus stored in array element REGPRT(I,ICHD) along with displacements of one or two semitones above and below this locus. Displacements corresponding to each index are stored in array OFFSET (initialized in line 10); from the locus and displacement, CHORDS computes a pitch (line 28) and degree (line 29). Arrays PCHTMP and DEGTMP store these values temporarily until such time as CHORDS determines whether the current candidate compares favorably or not to the incumbent. In the former case, CHORDS proceeds to transfer values from PCHTMP and DEGTMP to more permanent residence in PCHPRT and DEGPRT, at least until a better chord is encountered.

The first thing CHORDS does when considering an option is to check for doublings (lines 33-35), parallelism (lines 39-43 along with the logical function PARALL), and chromatic overlap (lines 46-57). Only when these constraints are satisfied does CHORDS attempt any heuristic evaluation.

Evaluation of 'chromatic redundancy' employs arrays USEDEG and USETMP. Array element USEDEG(I) tallies the number of already-composed chords employ the Ith degree of the chromatic scale, while element USETMP(J) holds the largest of these tallies for the first J degrees in the candidate. CHORDS evaluates the chromatic redundancy of each partial candidate (lines 61-66) by selecting the maximum between the current degree's tally and largest value for all previous parts; the variable LUSE gives the 'chromatic redundancy' of the incumbent.

Evaluation of 'sonorous quality' proceeds only when the candidate is no more redundant than the incumbent. Array QALTVL (initialized in line 11) indicates relative significances for each type of interval from the semitone to the major seventh. CHORDS considers only intervals between degrees, without regard to inversion or octave displacement (lines 77-79). Since four-part chords are characterized by 6 intervals, significances in QALTVL are expressed as powers of 7. Array element QALTMP(I) gives the contributions to the candidate of all intervals resulting from the first I degrees, while the variable LQAL gives the 'sonorous quality' of the incumbent.

12.3.2.2 Chordal Progression - PROGRS's main loop (lines 31-116) iterates on the variable ISEG, so as to initiate comparative searches for each of the 12 segments.

The loop immediately within this main loop (lines 41-115) provides the recursive mechanism for each search. The level of recursion corresponds to the position in the progression, expressed two ways: the variable ITMP indicates the position locally, within the segment, while the variable LPRG indicates the position globally, relative to the entire piece. ITEMP ranges from 1 to NUMSEG(ISEG). LPRG ranges from PRGSEG(ISEG) up to -- but not including -- PRGSEG(ISEG)+NUMSEG(ISEG); for efficiency, PROGRS stores the latter two positions in the holding variables LPRG1 and LPRG2, respectively. Array element PRGTMP(I) gives the index to the option under consideration for the Ith position. In this case, options are drawn with regard to order but without replacement from a pool of chords indicated by array elements POLPRG(LPRG1) through POLPRG(LPRG2). Array CHDTMP stores these options temporarily until such time as PROGRS determines whether the candidate progression compares favorably or not to the incumbent. In the former case, PROGRS proceeds to transfer values from CHDTMP to more permanent residence in CHDPRG, at least until a better progression is encountered.

Prior to its first search, PROGRS analyzes each pair of chords for common chromatic degrees (lines 14-28). Upon completion of this analysis, array element BLNCHD(I,J) gives the number of degrees shared in common by the Ith and Jth chords. We shall take this number as reflecting the relative 'blandness' of progressing immediately from chord I to chord J. In deriving the 'blandnesses' for progressions of several chords, PROGRS assigns greater significance to pairs sharing more degrees in common. Since the maximum number of chords which PROGRS must arrange within any segment is 6, significances of individual 'blandnesses' are expressed as powers of 7 (lines 54-60). Array element BLNTMP(K) gives the 'blandness' of the first K chords currently under consideration for the segment, while the variable LBLN gives the lowest 'blandness' of the incumbent progression.

The method of evaluating statistical balance between the chords is intriguing because ~~it~~ the program consults ~~a vector derived from~~ several keys whose relative significances vary according to context. The keys themselves are cumulative durations stored in array USETMP. (This array bears no relation to the USETMP appearing in subroutine CHORDS). Each time PROGRS selects a chord for a position, PROGRS augments the appropriate element of USETMP by the position's duration (line 48); conversely, PROGRS diminishes USETMP whenever it discards a chord (line 104). Array SCDTMP holds pointers to each element of USETMP; a call to LSORT (line 67) schedules these elements in descending order so that the largest cumulative durations have the greatest significance in the vector. Arrays USECHD and SCDCHD hold cumulative durations and an associated schedule for the best incumbent. Comparisons between candidates and incumbents are undertaken only when the program judges both progressions to be equally 'bland'. PROGRS implements such comparisons by stepping through pairs of durations in order of significance until it finds a pair which differs. It then favors the chord with the smaller duration (lines 68-78).

12.4 NOTES

1. We have encountered elementary algorithms for searching as early as Chapter 4: subroutine SELECT uses a <u>sequential</u> search, while subroutine QUICK uses a <u>binary</u> search.

2. This work by Newell, Shaw, and Simon was but one component of a general investigation of creative problem solving upon which the modern discipline of "artificial intelligence" is founded.

3. The rankings of major thirds as more consonant than minor thirds and especially of tritones as more consonant than minor seconds is somewhat arbitrary from a psychoacoustic standpoint. However, such rankings remain fully valid as expressions of stylistic preferences relative to specific ~~musical contexts.~~ compositional objectives.

4. The shortcut is known in the jargon of artificial intelligence as "alpha-beta pruning" (Knuth and Moore, 1975).

5. The chordal content of each segment was derived by an auxiliary program which sampled randomly shuffled pools (chapter 5) in order both to insure a diverse selection of chords within each segment and to balance the chordal resources numerically over the course of the piece.

6. No pair of distinct chords may share more than two degrees, due to the constraints upon the 12 chords.

7. The designation 'blandness' should be taken as an expression of stylistic preference for the purposes of this piece only.

8. A vector is a numeric description of a quality which employs two or more numbers. For example, any position on a flat surface may be described relative to a ~~fixed~~ stationary observer by a number indicating the direction and another number indicating the distance.

9. Since subroutine ARPEGG very closely resembles its counterpart in program DEMO7 (heading 9.3.2), this subroutine is not included in Example 12-2.

12.5 RECOMMENDED READING

Ames, Charles. "Protocol: Motivation, Design, and Production of a Composition for Solo Piano", Interface, volume 11, number 3 (1982), page 213.