CHAPTER 14

CONSTRAINED SEARCH

We have seen that many problems encountered in automated composition may be resolved into a sequence of elementary decisions, each of which admits a fairly small number of options. However, we know from chapter 12 that the number of potential solutions to such problems grows exponentially with the number of decisions. Even though it is theoretically possible to find an optimal solution to any problem using the methods of comparative search (chapter 12), in practice the requisite computations may go on for months or even years.

As an alternative to the intensive procedures of comparative search, this chapter investigates the strategy of constrained search. Of all the decision-making strategies discussed in this book, constrained search undoubtedly comes closest to simulating how human composers actually work. The approach involves specifying a minimum standard above which any solution is acceptable. Evaluative criteria are provided not as formulae for computing relative keys, but rather as constraints. For each decision, the search steps through potential options, testing for
violations. Whenever it encounters an option meeting all of the constraints, the search advances to the next decision; should the search exhaust all available options, it backtracks, revises one or more earlier decisions, and tries again.

Because this approach accepts the first complete solution encountered while rejecting any flawed solution immediately upon discovering a fault, constrained searches avoid the extended digressions which are characteristic of comparative searches. Consequently, constrained searches provide a practical mechanism for solving highly complex problems embracing large numbers of decisions. The disadvantage of constrained search versus comparative search is that the solutions produced by constrained searching are merely "acceptable", not optimal. However, it remains possible to impose heuristics affecting the schedule of decisions and the schedule by which the search considers options for each decision. Such heuristics enable the composer/programmer to bias a solution toward qualities which, though desirable, do not risk the absolute status of constraints.

14.1 APPLICATION: PART-WRITING BY CONSTRAINED SEARCH

In order to illustrate the mechanics of a constrained
search, we shall apply this strategy to a problem of traditional harmony: finding a six-part C major triad which resolves a six-part dominant seventh chord on G. Figure 14-1 depicts a schedule of parts in this progression along with schedules of potential resolutions for each part. The problem divides into six decisions, one for each part; each decision in turn admits up to three options, expressed in Figure 14-1 as melodic motions. Notice that decisions 3 and 6 admit only one acceptable option; the E⁴ in part 3 is the only pitch in a C major chord which resolves F⁴ downward by a step, while the C⁷ in part 6 is necessary to keep the chord in first inversion. In addition to these constraints implicit in the schedules for parts 3 and 6, the search imposes four explicit constraints:

1. no two parts may cross,

2. no two parts may move in consecutive fifths or octaves,

3. the C major chord may contain no more than two G's, and

4. the C major chord must contain exactly one E.

Figure 14-2 chronicles the search for an acceptable C major chord.
Fig. M-1
Figure 14-1: Part-leading schedules - The sequence of decisions proceeds from top to bottom, while the sequence of options for each decision proceeds from left to right.

Figure 14-2: Chronicle of search for an acceptable resolution - The numbers at the top of each column refer to the schedules depicted in Figure 14-1. Bold arrows indicate where one decision holds for multiple solutions. The parenthetic number after a comment indicates the source of conflict with an unacceptable decision.

We now consider the effect of heuristics affecting the schedules of decisions and options. Figure 14-3 depicts an alternate set of schedules for the same problem detailed above. It ranks heuristics for scheduling decisions as follows:

1. Number of options - The least flexible decisions (those with the fewest available options) receive greatest priority.

2. Urgency: The traditional "urge" for a dissonance to resolve downward by step is already implicit in the
restriction that F\textsuperscript{4} may only resolve to E\textsuperscript{4}. However, Figure 14-3 also incorporates the less emphatic "urge" of the leading tone to resolve upward.

3. **Prominence**: Other factors held equal, Figure 14-3 allows greater priority to the more readily audible outer parts, at the expense of inner parts.

4. In the event that the preceding three heuristics apply equally, scheduling of decisions is random.

The heuristics used to schedule options for each decision were:

1. **Tendency**: If a part has a tendency (that is, if it involves a dissonance or a leading tone), the pitch which resolves this tendency receives greatest priority.

2. **Smoothness of progression**: By contrast to Figure 14-1, which simply lists pitches of the C major triad from lowest to highest, Figure 14-3 favors the smallest motions.

3. In the event that the preceding two heuristics apply equally, scheduling of options is random.
Figure 14-4 chronicles a search proceeding according to these revised schedules. It is clear that, while the complete solution selected by both searches are identical, the effort invested in scheduling decisions pays off in decreased searching time. Notice, however, that the decision to lead the B4 upward in Figure 14-4 occurs directly. By contrast, it is simply an accident of circumstance that caused the previous search to take this step.

Figure 14-3: Revised part-leading schedules - The sequence of decisions proceeds from top to bottom, while the sequence of options for each decision proceeds from left to right.

Figure 14-4: Chronicle of search for an acceptable resolution - The numbers at the top of each column refer to the schedules depicted in Figure 14-3. Bold arrows indicate where one decision holds for multiple solutions. The parenthetic number after a comment indicates the source of conflict with an unacceptable decision.
Fig. 14-3
Fig 14-4
14.2 IMPLEMENTATION

A fully general constrained search is a paradigm of what Chapter 10 has designated "horizontal" recursion. Interpreted in this way, the recursive "level" indicates the current decision -- either directly or through a schedule -- while the process terminates when it reaches the goal of selecting an acceptable options for every decision. The basic strategy generalizes the approach taken by subroutines PARTS, EVAL, and LEGAL of program DEMO7 (heading 9.3.2), which implement most of the relevant procedures with the exception of backtracking. Remember that PARTS failed irrecoverably when none of the eight pitches scheduled by EVAL for any given decision satisfied all of LEGAL's constraints. Backtracking enables a search to recover from such failures.

Since backtracking requires the capability to take up where a search has left off in an earlier schedule, it is necessary to keep track of the following information for each decision:

1. the schedule of options,
2. an index to the current option under consideration, and
3. any ancillary data associated with the current decision.

Program SEARCH illustrates how one might implement a constrained search with backtracking. The parameter MDEC specifies the number of decisions, whose schedule is provided by the integer array DECIDX. Array element DECIDX(IDXDEC) holds the current decision, which SEARCH transfers to the holding variable IDEC for increased efficiency. Array element LIMDEC(I) holds the number of options for the Ith decision. Individual schedules reside in the two-dimensional integer array OPTIDX, which allows up to NOPT elements per decision: array element IDXOPT(IDDEC) provides an index to the current position in this schedule, while array element OPTIDX(IDXOPT(IDDEC),IDECC) holds the option itself. The integer array OPTDEC stores selected options for each decision; to determine the current partial solution, one must consult array elements OPTDEC(DECIDX(I)) for I=1,...,IDXDEC.

A call to a hypothetical subroutine ORDER (line 7) establishes the schedule of decisions. Subroutine EVAL (lines 13 and 29) determines individual schedules of options 'on the fly' each time the search advances to a new decision. The nature of ORDER and EVAL will vary with the application, though subroutine EVAL of program DEMO7 is representative; these subroutines may be dispensed with when schedules are provided manually. The logical function LEGAL (called from line 20) determines whether
program SEARCH
parameter (MDEC,MOPT)
integer DECI(MDEC),OPTI(MOPT),OPTDEC(MDEC)
:
   OPTPRT(MPRT),PCHOPT(MOPT,MPRT)
C Schedule decisions
call ORDER(DECIDX,MDEC)
C Search for acceptable solution
IDXDEC = 1
IDEC = DECI(IDXDEC)
C Schedule options for first decision
call EVAL(OPTD(MOPT),LIMDEC(IDEC))
IDXOPT(IDEC) = 0
   do
      I = IDXOPT(IDEC) + 1
      if (I.le.LIMDEC(IDEC)) then
         IDXOPT(IDEC) = I
         OPTDEC(IDEC) = OPTD(I)
         if (LEGAL(DECIDX,OPTDEC,IDXDEC)) then
            if (IDXDEC.eq.MDEC) then
               print *, (OPTDEC(I),I=1,MDEC)
               stop
            else
               Advance to next decision
               IDXDEC = IDXDEC + 1
               IDEC = PRIDEC(IDEC)
            end if
         end if
      else
         Options exhausted:  Backtrack to preceding decision
         IDXDEDEC = IDXDEC - 1
         if (IDXDEC lt 1) stop 'Unsuccessful search.'
         IDEC = DECI(IDXDEC)
      end if
   repeat
end
or not a newly-selected option is acceptable; like ORDER and EVAL, LEGAL will vary with the application, function LEGAL of program DEM07 is representative.

-- Programme example 14-1: program SEARCH --

14.2.1 Dependency-Directed Backtracking

A deficiency in SEARCH arises from the fact that the program simply backtracks to immediately preceding decision. As a result, SEARCH must grope its way backward along the schedule of decisions until it locates the cause of an impasse. For example, suppose SEARCH had attempted the search illustrated in Figure 14-2. Upon encountering the first conflict depicted in that Figure ("Too many E's", in the uppermost row), SEARCH would determine that all (one) of the options available to decision 3 had been exhausted, and in consequence would backtrack to decision 2. The program would then substitute a C5 for the G4 in decision 2 and return to decision 3. Since this action would not effect the number of E's in the chord, the impasse at decision 3 would still remain. Back to decision 2 again. SEARCH would now attempt the third option in decision 2's schedule, E5, only to run up against
the same constraint, "Too many E's". Only then would SEARCH backtrack to revise decision 1, which caused the problem in the first place by selecting an E.

14.2.1.1 An Expedient Method - The search actually depicted in Figure 14-2 incorporates a feature called "dependency-directed backtracking" by Stallman and Sussman, who first describe the problem (1977). A simple though non-rigorous implementation of dependency-directed backtracking involves simply determining the most recent source of conflict for each decision. Program SEARCH can perform such a determination with the following modifications:

1. Declare an integer array called BAKIDX of dimension MDEC in order to keep track of conflicts,

2. Since SEARCH has yet to encounter any sources of conflict at the onset of new decisions (that is, after lines 14 and 30), have it set BAKIDX(IDXDEC) to zero at these points.

3. In place of the logical function LEGAL (line 20),
substitute a new function ISOURC which tests the current option against each constraint and returns either 0 (no conflict) or a positive integer locating the earliest decision which is incompatible with the current option. Store the result of ISOURC in the holding variable IBAK and select whichever of the following branches is applicable:

a. If IBAK is zero, then SEARCH proceeds as if LEGAL had returned .true. (by executing lines 21-31);

b. otherwise, SEARCH sets BAKIDX(IDXDEC) = max0(BAKIDX(IDXDEC),IBAK). This second branch insure that if an impasse arises for the current decision, then SEARCH will backtrack only the minimal number of decisions required to break this impasse.

4. The actual process of backtracking reduces to setting IDXDEC=BAKDEC(IDXDEC). However, if the decision-making process involves cumulative feedback or maintains some other data which is not held 'frozen' for each decision, then it will be necessary to work back decision-by-decision, cancelling out intermediate
computations.

The programs used to generate Demonstration 11 (described later in this chapter) illustrate variations upon this expedient method of backtracking.

14.2.1.2 A Rigorous Method - The mechanism just described is simple to implement and highly effective for most applications. However, if it has to backtrack several times in a row (that is, if upon reaching an impasse, the search backtracks to the most recent source of conflict only to encounter another impasse, and so on), then the mechanism tends to loose track of the original impasse. Consider the sources of conflict indicated below:

<table>
<thead>
<tr>
<th>Decision</th>
<th>Sources of Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>none</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3, 1</td>
</tr>
<tr>
<td>7</td>
<td>6, 5</td>
</tr>
</tbody>
</table>

Suppose the search reaches an impasse at decision 7. It will then backtrack to decision 6, since this decision is the most
recent source of conflict. Suppose, however, that all of the options available to decision 6 have themselves been exhausted. The expedient mechanism described above will then cause the search to backtrack to the most recent conflict with decision 6, which is decision 3. In the process, the mechanism has lost touch with the original impasse, which might also have been broken by revising decision 5, at much less waste of effort.

In order to insure a mechanism rigorous enough to keep track of the original impasse, it is necessary for the program to assemble a complete list of conflicts for each decision. From these lists, the program in turn derives a backtracking schedule as follows: Initially, the schedule is empty. Whenever the search reaches an impasse, the computer merges the current list of conflicts into the backtracking schedule. (Duplicate conflicts are discarded.) The search then backtracks to the most recent conflict on the schedule.

A variety of information structures may be used to implement comprehensive backtracking. If sufficient memory is available, it may be most expedient to store the lists of conflicts in a two-dimensional array indexed by decision and option. Alternately, a one-dimensional array may store pointers to linked lists. This alternative is recommended only when the number of decisions is large and the average number of conflicts per decision remains well under half the number of options, since
each node in the linked list requires two elements, a value and a link. A linked-list structure is recommended for the backtracking schedule itself, since this schedule must accommodate frequent insertions and deletions of items.

14.2.2 Prescience

This chapter has used the word "impasse" to designate situations in which a search attempts to make a decision, but discovers that all of the available options are in some sense unacceptable. In such situations, an option must fall into one of two categories:

1. Options which are immediately found unacceptable either by the constraints implemented in function LEGAL or function ISOURC, or

2. Options which currently seem to be acceptable, but which propagate unresolved conflicts at later points in the search.

An example of the latter category of unacceptable option is the
E5 considered for the first decision (the uppermost part) in Figure 14-2. It seems very clear intuitively that if one (implicit) constraint requires the third-from-uppermost part to resolve F4 to E4 while another (explicit) constraint states "the C major chord must contain exactly one E", then the search is going to run into problems if it tries to select E5 for the uppermost part. Unfortunately, this information has not been communicated to the search of Figure 14-2, which blithely attempts to use E5 anyway.

For all the help which backtracking provides in recovering from fruitless digressions, the quickest way out is often to foresee such digressions and avoid them in the first place. For example, rather than saying "the C major chord must contain exactly one E", one could instead say "only the third-from-uppermost part may contain an E".

Such prescience must often be incorporated at the expense of generality, though not always. As another example, consider the simple problem of composing an arpeggio. Assume that there are ten attacks for the program to place within eight consecutive beats given two constraints: 1) each beat should contain at least one attack, and 2) no beat should have more than two (simultaneous) attacks. An unprescient way of coding the first constraint would be wait until all of the attacks had been used up, then check through the arpeggio to see if any holes remain.
A better way would be to keep tallies both of the number of empty beats and the number of unplaced attacks and to ask the following question each time the search attempted to double up attacks: "Will there still be sufficient unplaced attacks left to fill up the remaining empty beats?"

14.3 CONSTRAINED SEARCHES IN AUTOMATED COMPOSITION

The strategy of solving compositional problems by searching was used as early as the *Illiac Suite* (1957). In attempting to process streams of randomly generated notes through a "sieve" of stylistic rules, Hiller and Isaacson very quickly noted that "...with the addition of more rules, the probability of obtaining a successful piece of music would soon become very small". This problem led to their incorporation of a "try-again method" which allowed the Illiac to regenerate a note whenever it was confronted with a violation. This method was limited to retries on specific decisions only and lacked in efficiency, having no provision to restrain the Illiac from retrying notes which the computer had already rejected.

An article by Stanley Gill (1968) describes an approach used by Gill to compose a short piece entitled *Variations on a Theme*
by Berg. Though Gill does describe explicit procedures, it is
evident from his tree-graph of the decision-making process that
Gill's program was capable of backtracking to earlier decisions
whenever it ran out of options.

Of Larry Polansky's *Four Voice Canons*, numbers 2 (1975)
and 3 (1976) were both written using computer programs which
incorporate backtracking. The *Four Voice Canons* are based on
series of values used to determine musical attributes such as
pitch, duration, envelope, and various other aspects of timbre.
The number of values in each series varies from canon to canon.
Polansky's programs generated lists of permutations of this
series conforming to two constraints:

1. any permutation is derived from its predecessor in the
   list through the exchange of two elements; for example,
   the first and last elements of the five-note series
   "ABCDE" may be exchanged to obtain "EBCDA", and

2. every possible permutation occurs exactly three times in
   the list.

Polansky derived lists for each musical attribute to produce a
sequence of notes, and then overlaid the resultant sequence with
itself for times to produce his canons.
Kemal Ebcioglu has implemented constrained searches as means for testing the dictums of traditional contrapuntal theory. His 1980 paper describes a program for generating a single counterpoint against a cantus firmus, subject to rules provided by the user. In more recent work, Ebcioglu has developed programs which accept a chorale melody and attempt to compose four-part homophony in the style of J.S. Bach. His results have been impressive, duplicating Bach's own harmonization exactly in more than one instance.

Constrained search has become the primary technique used by Charles Ames. To compose Gradient for solo piano (1982), Ames used constrained searches to compose a progression of six-part chords and subsequently to arpeggiate each chord. With Undulant for seven instruments (1983) Ames implemented constrained searches capable of scheduling options "on the fly" for each decision, based on cumulative feedback. He also introduced a linked information structure capable of representing contrapuntal textures of arbitrary complexity (note 1).

14.4 DEMONSTRATION 11: CONSTRAINED SEARCH

Demonstration 11 illustrates the use of constrained searches
in a full-fledged composing program. The composition produced by
this program is a study in what Robert Erickson (1975) terms
"perceptual channeling", that is, the mechanism by which
listeners perceive disjoint musical events as components of an
ongoing process, or "channel". In Demonstration 11, the major
factor contributing to channeling is register, although the fact
that each pitch constantly associates with a fixed group of
partners also plays an important role.

14.4.1 Compositional Directives

The compositional process divides into four stages of
production:

1. **Stage I: Material** - composing the eight 'cells'
depicted in Figure 14-6;

2. **Stage II: Form** - selecting material for each segment
   in order to determine the compositional profile depicted
   in Figure 14-7;

3. **Stage III: Rhythm** - composing rhythm and selecting
cells for each note; and

4. **Stage IV: Pitch** - selecting inflections for each note.

As happened in Demonstration 10, the form of Demonstration 11 is induced from the 'bottom up' in Stage IV on the basis of qualities inherent in material composed previously by Stage I. Information from Stage II enables Stage III to describe all of the notes in the piece to the extent of rhythms and cell numbers; Stage IV completes the process by filling in inflections. The final product appears in Figure 14-9.

14.4.1.1 Stage I: Material - Figure 14-6 depicts the material of the piece, which consists of eight melodic cells. Each cell consists of three 'inflections' of a register: low, middle and high; these 'inflections' are realized by chromatic pitches spaced no farther apart than a whole tone. The material has the following properties:

1. each cell consists of two melodic steps, where a step may be either a semitone or a whole tone,
Cell 1  Cell 2  Cell 3  Cell 4  Cell 5  Cell 6  Cell 7  Cell 8

Fg's 14-6
2. of the four intervallic structures possible given the preceding constraint, each structure appears exactly twice,

3. no two cells overlap,

4. no degree of the chromatic scale appears more than twice in all the material, and

5. no two cells share more than one common chromatic degree.

Figure 14-6: Material for Demonstration 11 - Curved brackets indicate semitones; square brackets indicate whole tones.

14.4.1.2 Stage II: Form - The work consists of 18 segments. Eight segments draw material from one cell only; five segments draw material from two cells simultaneously; three segments draw material from three cells; and the remaining two draw material from four cells at once. Since the effect is of "implied counterpoint", it will be appropriate to use the word 'part' to
distinguish between cells simultaneously exploited in a single segment and also to refer to the various segments as 'solos', 'duets', 'trios', and 'quartets', depending on the number of parts involved.

Figure 14-7: Profile of Demonstration 11 - Segment durations, numbers of simultaneous cells, and average periods were specified manually by the author; the cellular content of each segment was composed by computer.

The constraints governing selection of cells for each segment are:

1. No two solos, duets, trios, or quartets may share an identical configuration of cells; neither may two quartets share more than two cells. This constraint insures a diversity of segments.

2. Two cells in adjacent registers may not occur in the same segment if their closest inflections lie within a minor third. This constraint inhibits 'cross channeling' between registrally adjacent cells.
3. A solo may not exploit any cell appearing in the immediately preceding segment; duets, trios, and quartets must share at least one cell with their immediate predecessor if the number of parts remains the same or increases. These constraints serve to provide a 'dovetailing' effect between consecutive segments.

14.4.1.3 Stage III: Rhythm - Stage III of the composing process selects periods between consecutive attacks by direct random selection using an exponential distribution limited using John Myhill's procedures (heading 4.4.2.1) so that the ratio of maximum to minimum durations is 8.0. Figure 14-7 details the average period between attacks for each segment. Notice that this average decreases (equivalently, the density of notes increases) as amount of available material rises.

The program selects cells using random selection with cumulative feedback (heading 7.2). This procedure allows unpredictable short-term choices while balancing cell-usage balances out over the long-term.

Articulation is sensitive to whether or not a note's successor shares the same cell:
1. If two consecutive notes share the same cell, then the program acknowledges this relationship by indicating either that the pair should be slurred or that the successor should be tongued with no intervening rest. This decision is conducted by Bernoulli trial (heading 4.4.1.1); the more parts occurring in a segment, the greater the likelihood that the pair will be slurred.

2. If two consecutive notes exploit different cells, then program acknowledges this difference by insisting that the successor always be tongued. A Bernoulli trial with 50% probability of success decides whether or not the program inserts an intervening sixteenth rest.

14.4.1.4 Stage IV: Pitch - The final stage of the composing process selects for each note in the composition which of the three regisral inflections available to the cell specified in Stage III will provide the pitch. The program attempts to keep these inflections in balance by employing cumulative feedback in order to favor the least-used inflection of any cell. It also forbids any cell from repeating an inflection without in the
meantime stating at least one of two alternate inflections.

Since the type of harmonic connections suggested by inter-cell consonances would contradict the central idea of the piece, the pitches in Demonstration 11 adhere to a dissonant style. As a minimum precaution against 'cross-channeling', the pitch-selecting program avoids virtual octaves; that is, when one cell plays a chromatic degree shared by a second cell, at least one of the two cells must play a different degree before the second cell may use the shared degree. In addition, consecutive notes must obey the stylistic matrix illustrated in Figure 14-8. Unlike stylistic constraints employed for Demonstrations 6, 7, and 8, the program skips over chromatic identities in order to apply this matrix to the first two distinct degrees immediately preceding the current note.

Figure 14-8: Stylistic matrix for Demonstration 11. Columns indicate 'current' chromatic intervals, given the 'most recent' interval indicated by the row. Non-blank entries show 'acceptable' intervallic sequences.

Figure 14-9: Transcription of Demonstration 11.
program DEMO11
C Demonstration of constrained search
C parameter (MCEL=8,MPAT=35,MSEG=18)
integer NUMSEG(0:MSEG),LIMSEG(0:MSEG),DURSEG(MSEG),CELPRAT(MPAT)
real CUMCEL(MCEL),INCSEG(MSEG)
C common CUMCEL,NUMSEG,LIMSEG,DURSEG,INCSEG,CELPRAT
C
IPAT = 0
LIMSEG(0) = IPAT
do (ISEG=1,MSEG)
   IPAT = IPAT + NUMSEG(ISEG)
   LIMSEG(ISEG) = IPAT
   INCSEG(ISEG) = float(DURSEG(ISEG))/float(NUMSEG(ISEG))
   repeat
   call FORM
   call RHYTHM
   stop

C subroutine FORM
C parameter (MCEL=8,MPAT=35,MSEG=18)
integer NUMSEG(0:MSEG),LIMSEG(0:MSEG),DURSEG(MSEG),CELPRAT(MPAT)
integer BAKSEG(MSEG),IDXCEL(MPAT),CELIDX(MCEL,MSEG),
     ILGCEL(MCEL,MCEL),LSEGCEL(MCEL,MCEL,OKAY)
real CUMCEL(MCEL),INCSEG(MSEG)
real FUZCEL(MCEL)
C equivalence (ILGCEL,LSEGCEL)
C common CUMCEL,NUMSEG,LIMSEG,DURSEG,INCSEG,CELPRAT
C data ILGCEL / 0, 0, 0, -1, -1, -1, -1, -1,
     : 0, 0, 0, -1, -1, -1, -1, -1,
     : -1, 0, 0, -1, -1, -1, -1, -1,
     : -1, -1, -1, 0, 0, -1, -1, -1,
     : -1, -1, -1, 0, 0, -1, -1, -1,
     : -1, -1, -1, -1, 0, 0, 0,
     : -1, -1, -1, -1, -1, 0, 0 /
C Initialization
C do (ICEL=1,MCEL)
C CUMCEL(ICEL) = 0.0
C do (ISEG=1,MSEG)
     CELIDX(ICEL,ISEG) = ICEL
     repeat
     repeat
     C Search for acceptable arrangement of cells
     ISEG = 1
     NUM = NUMSEG(ICEL)
     LIMO = LIMSEG(ICEL-1)
     LIM1 = LIMSEG(ICEL)
     IPAT = 1
     LCEL = MCEL - NUM + 1
     BAKSEG(ISEG) = 0
     IDXCEL(IPAT) = 0
C Schedule cells for first segment
     call FUZZY(CELDIX(1,ISEG),CUMCEL,FUZCEL,1.0,MCEL)
     do
     I = IDXCEL(IPAT) + 1
     if (I.LE.LCEL) then
         IDXCEL(IPAT) = I
         ICEL = CELIDX(I,ISEG)
         CELPRAT(IPAT) = ICEL
     C Constraints:
     OKAY = .true.
     IBAK = ISEG
C No duplicate segments; four-part segments may not share
C more than two cells
     if (IPAT.EQ.LIM1) then
         do (IS=1,ISEG-1)
         if (NUMSEG(IS).EQ.NUM) then
             K = 0
             IP = LIMO
             do
                 IP = IP + 1
                 IC = CELPRAT(IP)
                 LP = LIMSEG(IC-1)
                 do (NUM times)
                     LP = LP + 1
                 if (IC.EQ.CELPRAT(LP)) then
                     K = K + 1
                 exit
                 end if
             repeat
             if (IP.EQ.LIM1) exit
             repeat
         end if
     endif
     endif
end if
if (K.eq.NUM .or. K.ge.3) then
   OKAY = .false.
   IBAK = IS
end if
end if
repeat
end if

C Test for unacceptable pair of cells in same segment
IP = LIMO
do
   IP = IP + 1
   if (IP.eq.IPRT) exit
   if (.not. LEGCEL(CELPRAT(IP), ICEL)) then
      OKAY = .false.
      exit
   end if
repeat

C Count number of cells shared with preceding segment
if (IPRT.eq.LIM1) then
   N = NUMSEG(ISEG-1)
   K = 0
   IP = LIMO
do
   IP = IP + 1
   IC = CELPRAT(IP)
   LP = LIMSEG(ISEG-2)
do (N times)
   LP = LP + 1
   if (IC.eq.CELPRAT(LP)) then
      K = K + 1
   end do
   exit
repeat
end if
if (IP.eq.IPRT) exit
repeat

C Solo may not share cell
if (NUM.eq.1) then
   if (K.gt.0) then
      OKAY = .false.
      IBAK = min(IBAK, ISEG-1)
   end if
else if (NUM.ge.NUMSEG(ISEG-1) .and. K.ne.1) then
   C Other segments must share one cell if number of cells stays the
   same or increases
   OKAY = .false.
   IBAK = min(IBAK, ISEG-1)
end if
end if

C Accept or reject this cell
if (OKAY) then
   C Cell is acceptable for this part
   CUMCEL(ICEL) = CUMCEL(ICEL) + INCSEG(ISEG)
   C Advance to next part
   IPRT = IPRT + 1
   if (IPRT.gt.LIM1) then
      C Advance to next segment
      ISEG = ISEG + 1
      if (ISEG.gt.WSEG) return
      NUM = NUMSEG(ISEG)
      LIMO = LIMSEG(ISEG-1)
      LIM1 = LIMSEG(ISEG)
      MCEL = MCEL - NUM + 1
      IDXCEL(IPRT) = 0
      BAKSEG(ISEG) = 0
      C Schedule cells for next segment
      call FUZZY(CELIDX(1, ISEG), CUMCEL, FUZCEL, 1, 0, MCEL)
   else
      LCEL = LCEL + 1
      IDXCEL(IPRT) = IDXCEL(IPRT-1)
   end if
else
   C Cell is not acceptable for this part
   BAKSEG(ISEG) = maxo(BAKSEG(ISEG), IBAK)
end if
else
    Cells exhausted: Backtrack to preceding part
    if (IPRT=1,i.e.,LIMO) then
        if (IBAK.i.t.1) stop 'Unsuccessful search.'
    do
        IPRT = IPRT - 1
        if (IPRT,i.e.,LIMO) then
            ISEG = ISEG - 1
            NUM = NUMSEG(ISEG)
            LIMO = LIMSEG(ISEG-1)
            end if
            ICEL = CELPAT(IPRT)
            CUMCEL(ICEL) = CUMCEL(ICEL) - INCSEG(ISEG)
        if (ISEG.eq.IBAK) exit
        repeat
    LCEL = MCEL - NUM + 1
    else
      IPRT = IPRT - 1
      ICEL = CELPAT(IPRT)
      CUMCEL(ICEL) = CUMCEL(ICEL) - INCSEG(ISEG)
      LCEL = LCEL - 1
      end if
    end if
  repeat
subroutine RHYTHM
  parameter (MCEL=6,MPRT=35,MSEG=16,MNUM=4)
  integer NUMSEG(0:MSEG),LIMSEG(0:MSEG),DURSEG(MSEG),CELPAT(MPRT)
  real CUMCEL(MCEL),INCSEG(MSEG),AVGNUM(MNUM),ARTIC(MNUM)
  logical SUCCESS
  common CUMCEL,NUMSEG,LIMSEG,DURSEG,INCSEG,CELPAT
  data AVGNUM/3.0,0.5,2.2,1.7/,ARTIC/5.,66.,81./
  data HUGE/100000000.0/
  C
  Initialize
  open (2, file='DEMO11.RHY', status='NEW')
  C
  Increments for cumulative feedback in selecting cells for notes
  determined by cell-usage accumulated in FORM; likelihood of
  selecting least-used cell is 3 times smallest increment
  SUM = 0.
  OFFSET = HUGE
  do (ICEL=1,MCEL)
      C = CUMCEL(ICEL)
      SUM = SUM + C
      INCCEL(ICEL) = C
      OFFSET = min1(OFFSET,C)
      CUMCEL(ICEL) = 0.
  repeat
  OFFSET = OFFSET - 3.0
  SUM = SUM / float(MCEL)
  C
  Compose rhythm and select cell-numbers for each note
  ITIME = 0
  KTIME = 0
  REMAIN = 0.
  LIMO = 0
  do (ISEG=1,MSEG)
      NUM = NUMSEG(ISEG)
      LIMO = LIMO
      LIMO = LIMSEG(ISEG)
      KTIME = KTIME + DURSEG(ISEG)
      AVGPER = AVGNUM(NUM)
      do
          Select current period
          PER = RANV(AVGPER,.0) + REMAIN
          IPER = max0(1.,PER+.0)
          REMAIN = PER - float(IPER)
          C
          Determine largest cumulative statistic for cells in this segment
          T = 0.
          CMAX = 0.
          IPRT = LIMO
          do
              IPRT = IPRT + 1
              C = CUMCEL(CELPAT(IPRT))
              T = T + C
              CMAX = max1(CMAX,C)
          if (IPRT.eq.LIMO) exit
          repeat
          C
          END

C Select a cell for current note
R = RAND() * (float(NUM)*2*(CMAX+OFFSET)-1)
IPAT = LIM0
do
   IPAT = IPAT + 1
   ICEL = CELPAT(IPAT)
   W = CMAX - CUMCEL(ICEL) + OFFSET
   if (R.le.W) exit
   R = R - W
   if (IPAT.eq.LIM0) exit
repeat
C CUMCEL(ICEL) = CUMCEL(ICEL) + float(IPER)*INCCEL(ICEL)
C Articulate preceding note
   if (ITIME.gt.0) then
      IDUR1 = IPER1
      IDUR1 = IDUR1 + 1
   else
      if (SUCCECCARTIC(NUM1)) IDUR1 = IDUR1 + 1
      end if
   end if
   write (2,100) ITIME1,IPER1,IDUR1,ICEL1,ISEG1
100 format (5(1X,F9.0))
   end if
   ITIME1 = ITIME + IPER
C ISEG1 = ISEG + 1
C Advance to next note
   if (ITIME.ge.KTIME) exit
   repeat
   C Subtract expected cumulative sum for each cell used in this segment
   IPAT = LIM0
   do (NUM times)
      IPAT = IPAT + 1
      ICEL = CELPAT(IPAT)
      CUMCEL(ICEL) = CUMCEL(ICEL) - SUMSEG(ISEG)
   repeat
   write [2,100] ITIME1,IPER1,IPER1,ICEL1,ISEG1
66 write [2,100] -1,-1,-1,-1,-1
close [2]
end
program PITCH
parameter (MCSEL=8,MNFL=3,MQUE=50)
integer HEAD,TAIL,OLDCEL(MCSEL),
: TIMQUE(MQUE),PERQUE(MQUE),DURQUE(MQUE),CELQUE(MQUE),
: SEQUE(MQUE),OLOQUE(MQUE),NEWQUE(MQUE),CNTQUE(MQUE),
: DEQUE(MQUE),NFLQUE(MQUE)
integer BAKQUE(MQUE),IDXNFLQ(MQUE),NFLIDX(MNFL,MQUE),
: DEGNFL(MNFL,MCEL),CUMNFL(MNFL,MCEL),IGLTTLV(11,11)
logical LOLTTLV(11,11)
equivalence (LOLTTLV,IGLTTLV)
common HEAD,TAIL,IQUE,IUNCT,LIM,OLDCEL,
: TIMQUE,PERQUE,DURQUE,CELQUE,SEQUE,OLOQUE,NEWQUE,CNTQUE,
: DEQUE,NFLQUE
data IGLTTLV/-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,
: -1,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
: -1,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,
Sequence of degrees must conform to stylistic matrix

C IOLD1 = IQUE
84 do
85 if (IOLD1.eq.TAIL) go to 147
86 IOLD1 = IRET(IOLD1,IQUE)
87 IODE1 = DEGQUE(IOLD1)
88 if (IODE1.ne.IODE) exit
89 repeat
90 IOLD2 = IOLD1
91 do
92 if (IOLD2.eq.TAIL) go to 147
93 IOLD2 = IRET(IOLD2,IQUE)
94 IODE2 = DEGQUE(IOLD2)
95 if (IODE2.ne.IODE1 .and. IODE2.ne.IODE) exit
96 repeat
97 ITVL1 = IODE - IODE1
98 if (ITVL1.lt.0) ITVL1 = ITVL1 + 12
99 ITVL2 = IODE1 - IODE2
100 if (ITVL2.lt.0) ITVL2 = ITVL2 + 12
101 if (.not.LOLTLY(ITALYL1,ITVL2)) then
102 IBAK = min0(IBAK,CNTQUE(IOLD1))
103 end if
104 continue
105 C Accept or reject this inflection
106 if (IBAK.eq.ICNT) then
107 CUMNFL(INFL,ICEL) = CUMNFL(INFL,ICEL) + DURQUE(IQUE)
108 C Advance to next note
109 ICNT = ICNT + 1
110 if (IQUE.eq.HEAD) then
111 if (IHEAD(MQUE).eq.TAIL) call WNOTE
112 call ANOTE
113 if (TIMQUE(MHEAD).lt.0) go to 300
114 end if
115 IQUE = IADV(IQUE,IQUE)
116 C Schedule inflections for next note
117 ICEL = CELQUE(IQUE)
118 OLCQUE(INFL,ICEL) = IQUE
119 call SHUFLE(INFLOX(1,IQUE),MNFL)
120 call ISORT(INFLOX(1,IQUE),CUMNFL(INFL,ICEL),MNFL)
121 IDYNFL(IQUE) = 0
122 BAKQUE(IQUE) = 0
123 else
124 BAKQUE(IQUE) = max0(BAKQUE(IQUE),IBAK)
125 end if
126 C Inflections exhausted: Backtrack to most recent conflict
127 IBAK = BAKQUE(IQUE)
128 if (IBAK.eq.0 .and. ICNT.gt.1) then
129 IBAK = ICNT - 1
130 else if (IBAK.lt.LIM) then
131 stop 'Unsuccessful search.'
132 end if
133 do
134 IQUE = IRET(IQUE,IQUE)
135 ICNT = ICNT - 1
136 INFL = NPLQUE(IQUE)
137 ICQ = CELQUE(IQUE)
138 OLCQUE(INFL,ICQ) = IQUE
139 CUMNFL(INFL,ICQ) = CUMNFL(INFL,ICQ) - DURQUE(IQUE)
140 if (ICNT.eq.IBAK) exit
141 repeat
142 end if
143 repeat
144 do
145 if (TIMQUE(TAIL).lt.0) exit
146 call WNOTE
147 close [2]
148 close [3]
149 stop
150 end
subroutine RNOTE
parameter (MCEL=8, MQUE=50)
integer HEAD, TAIL, OLDCALL(MCEL),
  : TIMQUE(MQUE), PERQUE(MQUE), DURQUE(MQUE), CELQUE(MQUE),
  : SEQUE(MQUE), OLDCALL(MQUE), NEWQUE(MQUE), CNTQUE(MQUE),
  : DEQUE(MQUE), NFQUE(MQUE)
common HEAD, TAIL, IQQUE, ICNT, LIM, OLDCALL,
  : TIMQUE, PERQUE, DURQUE, CELQUE, SEQUE, OLDCALL, NEWQUE, CNTQUE,
  : DEQUE, NFQUE
C
  HEAD = IADV(HEAD, MQUE)
read (2, 10) TIMQUE(HEAD), PERQUE(HEAD), DURQUE(HEAD), CELQUE(HEAD),
  : SEQUE(HEAD)
10 format (S15)
if (TIMQUE(HEAD).lt.0) return
ICEL = CELQUE(HEAD)
IOLD = OLDCALL(ICEL)
OLDCALL(HEAD) = IOLD
if (IOLD.gt.0) NEWQUE(IOLD) = HEAD
NEWQUE(HEAD) = 0
OLDCALL(ICEL) = HEAD
C
CNTQUE(HEAD) = ICNT
return
end

subroutine WNOTE
parameter (MNFL=3, MCEL=8, MQUE=50)
character*3 MNENFL(MNFL, MCEL)
integer HEAD, TAIL, OLDCALL(MCEL),
  : TIMQUE(MQUE), PERQUE(MQUE), DURQUE(MQUE), CELQUE(MQUE),
  : SEQUE(MQUE), OLDCALL(MQUE), NEWQUE(MQUE), CNTQUE(MQUE),
  : DEQUE(MQUE), NFQUE(MQUE)
common HEAD, TAIL, IQQUE, ICNT, LIM, OLDCALL,
  : TIMQUE, PERQUE, DURQUE, CELQUE, SEQUE, OLDCALL, NEWQUE, CNTQUE,
  : DEQUE, NFQUE
C
  data MNENFL// 'E3', 'F3',' G3', 'A3','B3','C4',
  : 'C4','D4','E4','F4','G4','H4','A4',
  : 'B5','C5','D5','E5','F5','G5','A5',
  : 'A5','B5','C5','D5','E5','F6','G6'/
ITIME = TIMQUE(TAIL)
MEAS = ITIME/8
IBEAT = ITIME - MEAS*8
IGAP = PERQUE(TAIL) - DURQUE(TAIL)
ICEL = CELQUE(TAIL)
if (IGAP.lt.0) then
     write (3, 10) MEAS+1, IBEAT, PERQUE(TAIL),
     : MNENFL(NFQUE(TAIL), ICEL)
10 format (I2,';', 11,14,8X,A3)
else if (IGAP.eq.0) then
     type (3, 10) MEAS+1, IBEAT, PERQUE(TAIL),
     : MNENFL(NFQUE(TAIL), ICEL)
     write (3,16)
16 format (' Break')
else
     type (3, 10) MEAS+1, IBEAT, DURQUE(TAIL),
     : MNENFL(NFQUE(TAIL), ICEL)
     write (3,20)
20 format (' Rest')
end if
INEW = NEWQUE(TAIL)
if (INEW.gt.0) then
     OLDCALL(INEW) = 0
else if (TAIL.eq.OLDCALL(ICEL)) then
     OLDCALL(ICEL) = 0
end if
TAIL = IADV(TAIL, MQUE)
LIM = LIM + 1
return
end

function IADV(I,M)
IADV = I + M
if (IADV.gt.M) IADV = IADV - M
return
end

function IMET(I,M)
IMET = I - 1
if (IMET.lt.1) IMET = IMET + M
return
end
14.4.2 Implementation

-- Programming example 14-2: program DEMO11 (7 pages) --

Program DEMO11 proper serves merely as a controlling program for the two major subroutines FORM and RHYTHM. FORM selects material (note 2) for segments (Stage II), while RHYTHM composes all of the notes in the piece to the extent of describing periods, cells, and articulations (Stage III). RHYTHM stores its intermediate results in the file DEMO11.RHY for later processing by the independent program PITCH. This last program selects inflections for each note (Stage IV) and creates a mnemonic listing of the final products.

14.4.2.1 Searching for an Acceptable Form - Subroutine FORM implements a constrained search which selects from one to four cells for each segment. The initial data resides in two arrays: array element NUMSEG(I) holds the number of parts in the Ith segment while array element DURSEG(I) holds the segment's
duration in sixteenths (data for these two arrays are provided by 
lines 6-8 of the BLOCK DATA subroutine). FORM stores and 
accesses cells for each part in each segment by employing arrays 
CELPRT and LIMSEG along with the following scheme of pointers:
cells selected for the Ith segment reside in elements 
LIMSEG(I-1)+1 through LIMSEG(I) of CELPRT. Program DEMO11 proper 
automatically computes the relative positions stored in LIMSEG 
from the cell counts stored in NUMSEG (lines 10-16; excepting 
line 15).

The variable IPRT serves as the recursive index and as a 
pointer to the part and segment currently under consideration. 
Notice that FORM does not reset IPRT to 1 when it advances to a 
new segment; referring to Figure 14-7 for examples, IPRT=2 for 
segment 2, part 1; IPRT=7 for segment 5, part 2; and so on.

Since a cell may appear no more than once within any 
segment, FORM derives one schedule of cells for the whole segment 
and then proceeds to allot cells to parts by sampling this 
schedule without replacement. Array element CELIDX(J,I) holds 
the cell scheduled Jth in line for the Ith segment; array 
element IDXCEL(K) indicates which position in the schedule is 
currently being considered for the segment and part accessed by 
the pointer K; therefore, the actual cell number resides in 
array element CELIDX(IDXCEL(J),I). Sampling without replacement 
is effected by maintaining IDXCEL(LIMSEG(I-1)+1) through
IDXCEL(LIMSEG(I)) in strictly increasing order (due to line 137 of FORM).

Prior to the search, program DEMO11 proper computes for each segment the portion of the segment's duration which the program expects to devote to individual parts (line 15 of the loop spanning lines 10-16), storing this value in the real array INCSEG. Array CUMCEL maintains statistics of cumulative usage for each of the eight cells; each time it selects a cell, FORM increments the appropriate element of array CUMCEL by INCSEG(ISEG). Calls to the library subroutine FUZZY (heading 9.2) effect random scheduling with a strong bias -- the offset of 1.0 falls far short of the smallest value stored in INCSEG -- toward the least-used cells (lines 38 and 134 of FORM).

The bulk of subroutine FORM imposes the constraints upon the search. The tests for segments with identical material (lines 50-75) compare the current segment to every preceding segment, counting up the number of common cells in each case. The test for too-close cells (lines 77-85) steps through each cells already selected for the current segment and consults the logical array LGLCEL (initialized in lines 11-18) in order to determine whether or not the cell currently under consideration is compatible with this earlier commitment. Requirements for dovetailing (lines 88-116) are confirmed by first counting up the number of cells shared with the most recent segment, then
considering special cases.

Each time FORM encounters a configuration which proves
unviable in the light of choices made for an earlier segment, it
updates the backtracking variable BAKSEG(ISEG). FORM
scrupulously considers every possible configuration of cells for
the current segment until either it discovers a workable
arrangement or it runs out of combinations. In the latter case,
BAKSEG provides the most recent segment responsible for any
conflict.

14.4.2.2 Generating Notes - The main body of subroutine RHYTHM
consists of an outer loop (lines 32-94) iterating once for each
of the 18 segments and an inner loop (lines 38-86) iterating once
for each note in a segment.

RHYTHM selects periods between consecutive attacks (lines
40-42) via the library function RANX (heading 4.4.2.1). Average
periods reside in array AVGNUM (initialized in line 8) and depend
on the number of parts in the current segment, NUMSEG(ISEG).

The subprogram selects cells (lines 44-65) using the methods
of the library subroutine DECIDE (heading 7.2). At the end of
each segment, RHYTHM steps through the active cells, subtracting
each cell's 'expected' cumulative statistic SUM*INCCEL(ICEL) from
the actual value CUMCEL(ICEL) (lines 86-91). This procedure
insures that if a cell has received its 'fair share' of notes
during a segment, it starts out fresh in the next one; however,
when cells have been either slighted or overindulged, RHYTHM
retains awareness of such imbalances and acts to compensate for
them in later segments.

Articulations (lines 67-72) must be selected one step behind
in the process since how a note connects to its successor depends
on whether the successor exploits the same cell or a different
one. RHYTHM selects articulations by asking the library function
SUCCES (heading 4.4.1.1) to conduct Bernoulli trials; array
ARTIC (initialized in line 8) yields likelihoods that two
consecutive notes sharing identical cell numbers will be slurred;
as with average this likelihood depends on the number of parts in
the current segment. of parts

14.4.2.3 Searching for Acceptable Pitches - The independent
program PITCH with its attendant subroutines RNOTE and WNOTE
implement a constrained search which selects inflections for each
note in the piece. PITCH organizes data pertaining to individual
notes in several parallel queues (heading 10.2.1). Each queue is
distinguished by the mnemonic 'root' QUE; the following mnemonic
prefixes signify information read in by PITCH from the intermediate file DEMO11.RHY:

1. TIM - Starting time in sixteenths,
2. PER - Period to next attack in sixteenths,
3. DUR - Duration of note in sixteenths,
4. CEL - Melodic cell (1-8), and
5. SEG - Segment (1-18).

Subroutine RNOTE creates three additional items of data per note. Two items, a backward link OLDQUE and forward link NEWQUE, enable PITCH to access notes quickly when it needs information pertinent to specific cells. Figure 14-10 illustrates the linked structure derived by RNOTE for an actual sequence of notes read in from DEMO11.RHY. Pointers to the head of the backward list for each cell reside in the auxiliary array OLDCEL. The third item supplied by RNOTE, a decision count CNTQUE, indicates a note's position in the absolute sequence of decisions; this information assists the backtracking mechanism (note 3).

Figure 14-10: Data structure for program PITCH - Each row of numbers signifies a note; the left network of arrows shows backward links while the right network shows forward links. The information depicted here
<table>
<thead>
<tr>
<th>Backward Link</th>
<th>Index</th>
<th>Decision Count</th>
<th>Measure &amp; Best</th>
<th>Period</th>
<th>Duration</th>
<th>Cell</th>
<th>Segment</th>
<th>Forward Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>51</td>
<td>18:3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>52</td>
<td>18:5</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>53</td>
<td>18:7</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>54</td>
<td>19:0</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>55</td>
<td>19:4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>56</td>
<td>19:5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>57</td>
<td>20:1</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>58</td>
<td>20:5</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>59</td>
<td>20:7</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>60</td>
<td>21:1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>61</td>
<td>21:2</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>62</td>
<td>22:0</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>63</td>
<td>22:2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>64</td>
<td>22:7</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>65</td>
<td>23:1</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>16</td>
<td>66</td>
<td>23:3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>67</td>
<td>23:4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>68</td>
<td>23:5</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>19</td>
<td>69</td>
<td>24:0</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>70</td>
<td>24:2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>71</td>
<td>24:3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>22</td>
<td>72</td>
<td>24:7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>73</td>
<td>25:1</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>19</td>
<td>24</td>
<td>74</td>
<td>25:3</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>75</td>
<td>25:5</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>27</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>76</td>
<td>26:0</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
<td>77</td>
<td>26:3</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>26</td>
<td>28</td>
<td>78</td>
<td>26:7</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>79</td>
<td>27:0</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>80</td>
<td>27:1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>81</td>
<td>27:3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>82</td>
<td>27:6</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>34</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
<td>83</td>
<td>27:8</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>35</td>
</tr>
<tr>
<td>23</td>
<td>34</td>
<td>84</td>
<td>28:1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td>33</td>
<td>35</td>
<td>85</td>
<td>28:3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>26</td>
<td>36</td>
<td>86</td>
<td>29:3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>0</td>
<td>37</td>
<td>87</td>
<td>21:4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>39</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>88</td>
<td>14:6</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>36</td>
<td>39</td>
<td>89</td>
<td>15:0</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>35</td>
<td>40</td>
<td>90</td>
<td>15:5</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>91</td>
<td>16:0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>43</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>92</td>
<td>16:2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>44</td>
</tr>
<tr>
<td>42</td>
<td>43</td>
<td>93</td>
<td>16:5</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>43</td>
<td>44</td>
<td>94</td>
<td>16:6</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>46</td>
</tr>
<tr>
<td>46</td>
<td>45</td>
<td>95</td>
<td>16:7</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>47</td>
</tr>
<tr>
<td>44</td>
<td>46</td>
<td>96</td>
<td>17:1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>48</td>
</tr>
<tr>
<td>45</td>
<td>47</td>
<td>97</td>
<td>17:3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>49</td>
</tr>
<tr>
<td>47</td>
<td>48</td>
<td>98</td>
<td>17:5</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>50</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>99</td>
<td>18:1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>51</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>100</td>
<td>18:2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>52</td>
</tr>
</tbody>
</table>
describes the portion of Figure 14-9 beginning with the second quarter of measure 14 and ending after the fifth sixteenth of measure 28.

The head and tail of the queue are indicated by the integer variables HEAD and TAIL, respectively. The variable IQUE both serves as the recursive index and locates the note currently under consideration. The integer functions IADV and IRET handle the "wrap-around" arithmetic necessary to keep this and related indices between 1 and MQUE. Array element NFLQUE(IDXNFL(IQUE),IQUE) holds the inflection under scrutiny; PITCH also transfers this value to the holding variable INFL for more efficient access. Array DEGNFL (initialized in lines 25-26) supplies chromatic degrees for each inflection in each cell; PITCH transfers the current note's degree from array element DEGNFL(INFL,CELQUE(IQUE)) to the holding variable IDEG. Both INFL and IDEG are also stored in queues of their own for easy future reference: NFLQUE and DEGQUE.

Each time PITCH selects an inflection for a note, the subprogram increments the appropriate element of array CUMNFL by the duration the note. Scheduling is first rendered unbiased by random shuffling (heading 5.2), after which a call to the library subroutine ISORT (heading 9.1) strictly favors the least-used inflections (lines 49-50 and 120-121).
The constraints controlling how degrees of the chromatic scale may recur are greatly facilitated by the linked structure illustrated in Figure 14-10. PITCH initiates its test for inflections repeated immediately within the same cell (lines 65-69) by locating the most recent note exploiting the same cell through array OLDCEL. The test then reduces to simply comparing inflection numbers. The test for virtual octaves (lines 71-82) steps through each cell other than the current note's cell, using OLDCEL to locate the other cell's most recent note. If both notes share the same chromatic degree and if the current cell has no intervening note, then the program rejects the current inflection.

By contrast to the tests just described, the test for conformity to the stylistic matrix illustrated in Figure 14-8 (lines 84-105) is unconcerned with cell numbers. The first step is to locate the two most recent notes in the queue whose pitches are chromatically distinct both from the inflection currently being considered and from each other. The program then feeds the resulting chromatic intervals into the logical array LGLTVL (initialized in the DATA statement spanning lines 14-24) in order to determine if the intervallic sequence is suitable.

The backtracking mechanism for program PITCH first consults array BAKQUE in order to determine the source of an immediate conflict. Sometimes the subprogram determines all of the
inflections considered for a note suitable, only to propagate impasse at later notes in each instance. PITCH was unable to supply information for dependency-directed backtracking in such cases, so the subprogram was forced to grope back note-by-note in order to pinpoint the source of conflict empirically.

14.5 NOTES

1. A similar information structure is used in program PITCH, described under the next heading.

2. The material for Demonstration 11 was itself composed by computer using the techniques described in this chapter.

3. For PITCH'S purposes, array TIMQUE could easily serve this purpose; however, this implementation is designed to handle truly polyphonic applications which allow several notes to start simultaneously.
14.6 RECOMMENDED READING
