GenJam: A Genetic Algorithm for Generating Jazz Solos

John A. Biles

Associate Professor
Information Technology Department
Rochester Institute of Technology
jab@cs.rit.edu

Abstract

This paper describes GenJam, a genetic algorithm-based model of a novice jazz musician learning to improvise. GenJam maintains hierarchically related populations of melodic ideas that are mapped to specific notes through scales suggested by the chord progression being played. As GenJam plays its solos over the accompaniment of a standard rhythm section, a human mentor gives real-time feedback, which is used to derive fitness values for the individual measures and phrases. GenJam then applies various genetic operators to the populations to breed improved generations of ideas.

1 Introduction

As with most problem-solving activities, musical tasks like composition, arranging and improvising involve a great deal of search. Composers search for the right chords to fit a melody or the right melody to fit a chord progression; arrangers search for the right voicings and counterpoint for constituent parts; improvisers search for the right phrases to play over a particular set of chord changes. Certainly, the notion of "right" is individual, but a typical musician "knows what she likes," and this aesthetic sense guides the search through the various problem spaces of notes, chords, or voicings.

Genetic algorithms (GAs) provide a powerful technique for searching large, often ill-behaved problem spaces. A GA solves problems by evolving a population of potential solutions to a problem, using standard genetic operations like crossover and mutation, until an acceptable solution emerges. The minimal requirements for using a GA are that the solutions must map to a string of symbols (preferably bits), and that there be some way of determining how good one solution is at solving the problem, relative to the other solutions in the population.

Given the power GAs possess for searching strange problem spaces and their rather lax requirements for use, it seems natural to apply GAs to musical tasks. This paper describes one such application, GenJam, which searches a large melodic space for "good" material with which to build jazz solos. GenJam's metaphor is an enthusiastic student musician who sits in at jam sessions. When this student plays well, the other musicians respond with "Yeah!" and other classically cool jazz exhortations. When the student plays poorly, the other musicians might respond by "gonging him off," as Jo Jones did to a young Charlie Parker by sailing a cymbal at his feet during a Kansas City jam session. GenJam uses similar, though less dramatic, feedback to guide its search through a melodic space.

2 Background

GAs [Holland, 1975; Goldberg, 1989; Koza, 1992] have been applied to music in the areas of thematic bridging [Horner and Goldberg, 1991], and FM parameter matching [Horner et al., 1993]. Both of these applications employ a "standard" GA in that there is a population of potential solutions that evolves until a single individual emerges whose performance is acceptable. An individual in these populations is represented by a chromosome-like bit string, which maps to a sequence of melodic transforms in thematic bridging and to a set of FM parameter values in the other study. Each individual also has a fitness, which is a numeric indication of the success of that solution is at solving the problem.

Horner et al. [1993] provide an excellent introduction to GAs with a clear musical example. For the uninitiated, a "simple" GA looks like:

```
Initialize the individuals in the population
While not finished evolving the population
    Figure the fitness of each individual
    Select better individuals to be parents
    Breed new individuals
    Build next generation with new individuals
elihW
```

The initialization step is usually random, but one may seed the initial population with individuals that fit some criterion. The loop can be controlled in a variety of ways. Searching can continue until 1) a maximum number of generations has been bred; 2) an individual emerges that meets some criterion for performance; 3) the population converges on a single individual; or 4) you run out of computer time, memory, patience, or funding.

The fitness for an individual is usually determined algorithmically (as in mathematical function optimization problems, where the function being optimized *is* the fitness function). Some GAs have employed human "fitness functions" to generate

images. Sims [1993] evolved images by allowing the user to select favorites from a population to serve as parents for the next generation. Latham and Todd [Haggerty, 1991] allowed the user to select a favorite image in a small population to serve as a single parent. As we shall see, human fitness functions in a musical domain present some interesting problems.

The selection step in the algorithm reflects the evolutionary principle that the fitter individuals in a population tend to survive and mate. Selection in GAs usually involves a random process, biased by the fitness values, so that fitter individuals are more likely, but not guaranteed, to reproduce.

Breeding new individuals is done by combining the parent strings, usually with some form of crossover and mutation. These operations typically occur at the symbol (bit) level and mirror crossover and mutation in chromosomes of natural organisms. However, natural genetics have been more of an inspiration than a constraint, and many clever domain-specific operators have been invented.

In building the next generation, some old individuals, usually the fittest from the past, may survive intact, creating a *generation gap* between them and the new children. The size of this gap, along with the population size, helps control the speed of the search. A large population with no generation gap will cover the most ground but might lose a promising individual. A small population with a large gap will not lose its best individuals, but it will sample the solution space much more slowly.

The most important aspect of designing a GA is the representation of individual solutions. To mirror natural genetics, an individual is represented by a string of symbols. If those symbols are bits, the representation more closely resembles the genes and chromosomes in natural organisms, but many GA applications use non-binary representations. In this paper, however, we will focus only on bit strings.

An individual bit string (genotype) in a population must somehow map to a potential solution (phenotype). A clever representation that efficiently represents alternative solutions, perhaps by excluding clearly unacceptable solutions, will lead to a more efficient search. However, if a representation "cleverly" excludes the best solution, its efficiency is irrelevant. We can easily calculate the size of the solution space by computing the number of different solutions that can be represented by the bit string. This is simply 2^n, where n is the number of bits in the string. The more bits in the string, the more potential solutions, and the larger the search space (each added bit doubles the size). GA designers walk a thin line between too large a search space on one side and inadequately sampled solutions on the other.

We turn now to a brief discussion of computational jazz improvisation systems. Most such systems reported in the literature [Fry, 1984; Levitt, 1981; Giomi and Ligabue, 1991; Ames and Domino, 1992] use knowledge based techniques to derive an "improvised" solo from a given harmonic progression in a constrained rhythmic style. These systems process an abstract progression (set of chord changes) before generating a solo, and usually take the intermediate step of building a sequence of scales from which melodic material will be drawn. This activity mirrors what most "trained" improvisers do when they prepare a tune for improvisation [Haerle, 1989].

The interactive jazz system being developed at McGill University [Penneycook et al., 1993] holds truer to jazz's aural tradition in that it listens to a rhythm section in real time in order to determine what the rhythm section is playing. The philosophy is to give the system as little *a priori* knowledge as possible about jazz harmony and rhythm.

Metamuse [Iverson and Hartley, 1990] also avoids providing explicit *a priori* musical knowledge. It uses autocatalytic set theory to assimilate music fed to it and regenerate new pieces in a similar style. While Metamuse uses string representations, it is not, strictly speaking, a GA.

3 Design of GenJam

GenJam was developed in a Macintosh/Think C environment on top of the CMU MIDI Toolkit [Dannenberg, 1993]. Figure 1 shows GenJam's system architecture and provides a visual overview of its operation. To improvise on a tune, GenJam reads a progression file, which provides it with the tempo and rhythmic style (swing or even eighth notes), the number of solo choruses it should take, and the chord progression. It also reads MIDI sequences for piano, bass and drums, which have been pre-generated using Band-in-a-Box [Gannon, 1991].

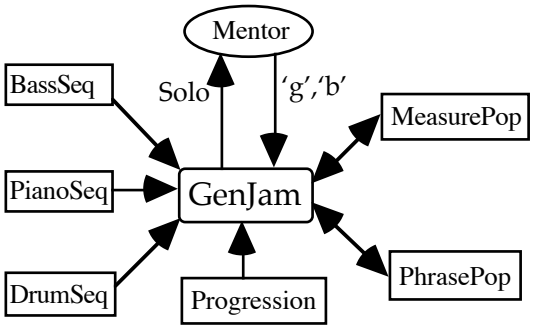

Figure 1. GenJam System Architecture

GenJam improvises on the tune by building choruses of MIDI events decoded from members of the measure and phrase populations. Since, as we shall see, a phrase is implemented as a sequence of four measures, these two populations form a mutually dependent hierarchy of melodic structures.

While listening to a solo, the mentor can type one or more 'g's if a portion is judged to be good, or one or more 'b's if a portion is judged to be bad. The fitness for a given measure or phrase is accumulated by incrementing counters for the currently playing measure and phrase every time a 'g' is typed, and decrementing them every time a 'b' is

typed. The modified fitness values are written back to the population files after the solo terminates.

GenJam runs in one of three modes: learning, breeding or demo. Learning mode is intended to build up fitness values and uses no genetic operators. Phrases are selected at random, ignoring fitness, and presented for feedback. Demo mode is intended to be a "performance." Phrases are selected with a tournament selection process that considers both the phrase fitness and constituent measure fitnesses, and feedback is ignored. In breeding mode the genetic operators are applied, and half of each population is replaced by new offspring before a solo is presented for feedback. The rest of this section details GenJam's representations, genetic operators and fitness procedure.

3.1 Chromosome Representation

As alluded to above, the design of a string representation is critical to a GA's success. GenJam uses a cooperating, two-level, position-based, binary representation scheme. The rest of this subsection will try to explain what that means.

Two major differences exist between GenJam and the simple GA described above. One is that GenJam uses two populations, one of measures and one of phrases. An individual in the measure population maps to a sequence of MIDI events, as will be detailed below. An individual in the phrase population maps to indices of measures in the measure population.

The other major difference is that GenJam uses the entire populations of measures and phrases to build a solo, not just a single "best" measure or phrase. In this way GenJam more closely resembles a classifier system [Goldberg, 1989] or the "musical strata" of Horner [1993b]. It is important to note that GenJam is not trying to evolve the perfect solo on a specific tune; it tries to evolve a workable collection of melodic ideas that it can apply to any tune.

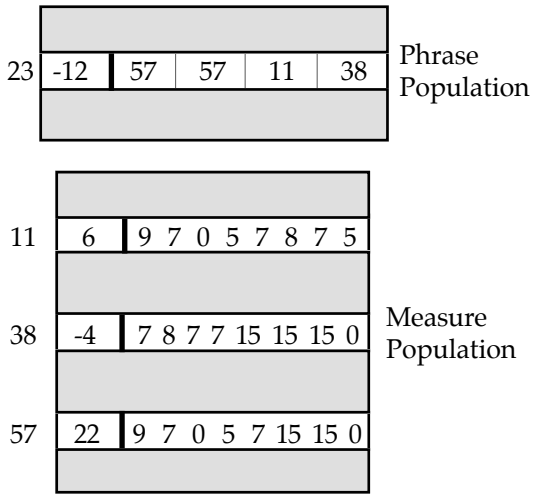

Figure 2. Example Phrase and its Measures

Figure 2 shows a "composed" example phrase, which maps to a rather unhip rendition of the first four bars of Sonny Rollins's Tenor Madness, transposed to C. In both populations, the single number to the left of the heavy line in each individual is the fitness value, and the remaining numbers represent the chromosome.

The example focuses on phrase number 23 and its constituent measures. Phrase 23 has a fitness of -12, which means that it has not been particularly well received by the mentor. Its chromosome is the concatenation of four numbers, each of which is a pointer (array index) into the measure population. The current population sizes for GenJam are 48 phrases and 64 measures. The number 64 is not arbitrary because in order to get maximum efficiency from the phrase representation, the size of the measure population should be a power of two. The reason for this is illustrated in Figure 3.

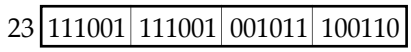

Figure 3. Phrase 23 Chromosome at the Bit Level

It is important that any possible configuration of bits map to a legitimate structure, and it is preferable to isolate specific pointers in specific bit substrings, to make the mappings easier. In this case 64 different measure indices require exactly six bits, and the resulting chromosome of four measure pointers requires 24 bits.

Individuals in the measure population are made up of a fitness value and a chromosome that is interpreted as a series of eight events, one for each eighth note duration of a 4/4 measure. There are three types of events: a new note, a rest, and a hold. A *new-note* event causes a MIDI note-off followed by a note-on. A *rest* causes a note-off only. A *hold* causes nothing to happen, which has the effect of holding a note already turned on or lengthening a rest. A hold at the beginning of a measure holds whatever ended the previous measure, so rhythmic structures can flow across measure boundaries. If swing rhythm is selected, a 62% swing is used (events on the beat last 62% of the length of a quarter note, while those occurring off the beat last the other 38%). If even rhythm is selected, all events have the same duration, which is appropriate for Latin tunes.

There are 14 different new note events (encoded as 1-14 in Figure 2), one rest (encoded as 0), and one hold (encoded as 15), which adds up to 16 possible events that can occur at each eighth-note position in a measure. An event, then, can be represented in 4 bits and a 4/4 measure in 32 bits, yielding a melodic space of something less than 2^{32} different measures (a rest following a rest and a hold following a rest will sound the same).

The major advantage to thinking in terms of note-off followed by note-on, rather than the reverse, is that note durations can be represented in half a bit per event (two bit permutations out of 16 for each four-bit event). This efficiently unifies pitch and

rhythmic structures in a single representation, as opposed to the more typical approach of treating pitch and rhythmic sequences separately [Ames and Domino, 1992; Giomi and Ligabue, 1991; Fry, 1984].

Two obvious disadvantages to GenJam's scheme are that notes occur only in eighth note multiples, and there are only 14 pitches to choose from at any one time. These certainly would be a noticeable limitation on most human improvisers, but allowing greater rhythmic and chromatic diversity would increase the string lengths needed for measures, thereby exploding the size of the space searched by the measure population.

One other thing to note about Figure 2 is that it is perfectly permissible to repeat a measure in a phrase. When this happens, the listener tends to respond with, "Wow! Thematic development!" In this example, as we shall see in the next section, the chord progression will lead to different scales being used for the two measures, which will result in a slight difference in the actual pitches played.

3.2 Chord/Scale Maps

The 14 new-note events are mapped to actual MIDI pitches through scales suggested by the chord progression being played. As was shown in Figure 1, a progression file is read and processed before the solo is generated. This results in a note map for each half measure of a chorus of the tune (maximum harmonic tempo of two chords per measure). Each note map is an array of 14 MIDI pitches, roughly in the two octaves ascending from middle C.

A given chord is mapped to a scale strictly vertically; that is, the window through which the progression is viewed is only one chord wide. Table 1 summarizes the types of chords currently recognized and the scale mappings for each chord type, using a chord root of C for the examples.

Chord	Scale	Notes
Cmaj7	Major (avoid 4th)	C D E G A B
C7	Mixolydian (~ 4th)	C D E G A Bb
Cm7	Minor (avoid 6th)	C D Eb F G Bb
Cm7b5	Locrian (~ 2nd)	C Eb F Gb Ab Bb
Cdim	W/H Diminished	C D Eb F F# G# A B
C+	Lydian Augmented	C D E F# G# A B
C7+	Whole Tone	C D E F# G# A#
C7#11	Lydian Dominant	C D E F# G A Bb
C7#9	Altered Scale	C Db Eb E F# G# Bb
C7b9	H/W Diminished	C Db Eb E F# G A Bb
Cm7b9	Phrygian	C Db Eb F G A Bb
Cmaj7 #11	Lydian	C D E F# G A B

Table 1. Chord/Scale Mappings

These mappings are a synthesis of several sources in the jazz education literature [Russell, 1959; Hearle, 1980 and 1989; Sabatella, 1992; Coker, 1964]. While this collection of chords is not exhaustive, and the scale choices are certainly debatable, they provide a rich enough set to handle many jazz tunes in a variety of styles.

After a scale is selected, it is extended to 14 tones, beginning at or above middle C. For example, a C7 chord would indicate a C mixolydian scale without the somewhat controversial 4th (C D E G A Bb), and the resulting note map would be (C3 D E G A Bb C4 D E G A Bb C5 D). An F7 chord would indicate an F mixolydian scale (F G A C D Eb), and the resulting note map would be (C3 D Eb F G A C4 D Eb F G A C5 D). If the example in Figure 2 is played against the first four bars of a typical blues progression (C7 F7 C7 C7), the repeated measure 57 maps to two different sets of pitches: E C A C in the C7 measure, and Eb C G C in the F7 measure.

The results of using these note maps is that GenJam can develop ideas to fit different harmonic contexts and will not play a "wrong" note. That is not to say that all of its notes are "right," however, which brings us to the role of the mentor in providing fitness values for the phrases and measures.

3.3 The Fitness Bottleneck

The only really firm requirement for using a GA is a method for determining fitnesses of the individuals in a population. At the very least, a method must exist for determining which of two arbitrary individuals is "better." If a population can be ranked unambiguously, or if an interval or ratio scale can be developed for fitness, then greater sophistication may be used in the genetic operators, particularly selection and replacement.

The unspoken assumption is that this fitness method is algorithmic. If a suitable algorithm can be found, then fitness becomes just another routine in the program, and while it may be computationally expensive, it is at worst a performance bottleneck that leads to days or weeks of computation rather than minutes or hours. Thousands of generations still can be generated with enough patience.

However, what if no suitable algorithm exists for generating fitness, even in the minimal case of deciding which of two individuals is better? After initially considering a neural network trained to respond as I do to pieces of music, I decided to put off the search for an algorithm that implements "I know what I like," and use myself as a fitness function.

As indicated above, this approach has been used successfully in evolving images [Sims, 1993; Haggerty, 1991], but the task for the human rater was made easier by the presentation of several images concurrently. Raters could easily compare images by looking from one to another in whatever order they chose. The presentation of several musical samples, on the other hand, cannot be made concurrently, Charles Ives not withstanding. Furthermore, the presentation of a single image is essentially instantaneous, while a piece of music must be played from start to finish at the proper tempo.

The human who serves as GenJam's mentor, then, is a very narrow bottleneck and is, in fact, the limiting factor on population sizes, number of generations, and size of any generation gap. This is because the mentor must listen carefully to every measure of every phrase to provide their fitnesses.

A typical solo will be three choruses of a 32-bar form, which is about the upper limit for quality feedback from the mentor. This works out to 24 four-bar phrases, which, not coincidentally, is half the size of the phrase population. In breeding mode, GenJam replaces half of both populations (50% generation gap) and then plays the new phrases first in its solo. This leads to the phrase population size of 48 phrases -- the mentor has to listen to all the new phrases, but 24 is the practical limit, so 24 must be half the population size. Feedback for the measures is not a problem because each measure is sampled an average of three times in the phrase population (48 phrases * 4 measures per phrase / 64 measures).

Clearly, the mentor is a critical resource, and GenJam's design reflects the need to minimize the amount of listening required and to make the mentor's interface as simple as possible. While listening to GenJam play a solo, the mentor can type 'g' (for good) or 'b' (for bad) whenever so moved. Each time a 'g' is typed, the fitnesses for the currently playing measure and phrase are both incremented. When a 'b' is typed, both fitnesses are decremented. Fitnesses have a floor of -30 and a ceiling of +30, to guard against successful established individuals overwhelming new ones, and to make it easier for the mentor to thin out a nice lick that becomes overused. The mentor can control the magnitude of the feedback from neutral (no typing) to intense (rapid keystrokes).

To allow time for the mentor to react, empirically derived delays have been built into the feedback mechanisms so that the feedback window for measures is shifted two beats late and the window for phrases one measure late. This means that when a 'g' or 'b' is typed during the playing of beats three or four of a measure, the counter for that measure is incremented or decremented. Feedback typed during beats one or two will affect the previous measure. Similarly, feedback occurring in the first measure of a phrase applies to the previous phrase, while feedback in measures two, three and four affect the current phrase.

3.4 Genetic Operators

Both the fitness bottleneck and the cooperative nature of the two populations heavily influenced GenJam's genetic operators. GenJam applies all its operators only in breeding mode, but selection is also used in demo mode. This section details GenJam's initialization, selection, crossover, mutation, and replacement operators.

The measure and phrase chromosomes are initialized by generating random bit strings of the appropriate length. Fitness values are initialized to zero. For phrases the strings are uniformly random. For measures the strings are interpreted as the eight four-bit events (0 - 15), with 0 denoting a rest, 15 a hold, and 1-14 new notes. Rests and holds each occur with probability 5/24, and each new note occurs with probability 1/24. This proportion of rests and holds was derived empirically and seeds the initial population with some modicum of "rhythm." The encoding of rest and hold as mutual logical complements protects the population from artificially converging on high or low notes when a crossover point occurs within a rest in one parent and a hold in the other.

Selection and replacement in GenJam are merged in a modified tournament selection process. Four individuals are chosen at random, without regard to fitness, to form a *family*. Of the four individuals in a family, the two with the highest fitness are used as parents, and the two worst are replaced by the two offspring of the parents' mating. In each generation, half of the measure population is thus replaced, and newly created children cannot participate in later families in the same generation.

To insure that the 32 new measures will be heard quickly by the mentor, they are each randomly assigned to one of the first eight new phrases. Each of these "maternity" phrases replaces the loser of a four-phrase tournament, based on phrase fitness. The remaining 16 new phrases are bred in families as described above.

GenJam performs a standard single-point crossover at a random location in the 32-bit measure strings (or 24-bit phrase strings). One of the resulting two children is kept intact, while the other child is mutated by one of several mutation operators, which operate at the event level for measures and the index level for phrases.

In an effort to accelerate learning by creating not just new, but better offspring, these "musically meaningful mutation" operators violate conventional GA wisdom that genetic operators should be "dumb" with respect to the structures they alter. Table 2 summarizes the six mutation operators for measures, using measure 57 from Figure 2 above as an example.

Mutation Operator	**Mutated Measure**
None (Original Measure)	9 7 0 5 7 15 15 0
Reverse	0 15 15 7 5 0 7 9
Rotate Right (e.g., 3)	15 15 0 9 7 0 5 7
Invert (15 - value)	6 8 15 10 8 0 0 15
Sort Notes Ascending	5 7 0 7 9 15 15 0
Sort Notes Descending	9 7 0 7 5 15 15 0
Transpose Notes (eg. +3)	12 10 0 8 10 15 15 0

Table 2. Musically Meaningful Measure Mutations

The rotation operator rotates events a random number of event positions to the right (1 to 7). The inversion operator has the effect of turning rests into holds, holds into rests, and reflecting pitches roughly around C4. The sorting and transposition operators preserve the rhythmic structure of a measure in that

rests and holds are not affected. Transpositions are performed a random number of steps (1 to 4) in the direction of the greater minimum distance between a note and an upper or lower bound (1 or 14). In other words transpositions are done in the direction that "has the most room." If a note is transposed beyond the allowed note range (1 - 14) it is reflected off that bound back into the accepted range. For example, 12 transposed up 3 steps will become 13 (12-13, 13-14, 14-13).

The six mutation operators for phrases are summarized in Table 3, which applies the operators to our sample phrase from Figure 2. The reverse and rotate operators are the same as those used on measures. The genetic repair operator replaces the index of the measure with the worst fitness (in this case measure 38) with a random measure index (in this case 29).

Mutation Operator	Mutated Phrase
None (Original Phrase)	57 57 11 38
Reverse	38 11 57 57
Rotate Right (e.g., 3)	57 11 38 57
Genetic Repair	57 57 11 29
Super Phrase	41 16 57 62
Lick Thinner	31 57 11 38
Orphan Phrase	17 59 43 22

Table 3. Musically Meaningful Phrase Mutations

The super phrase operator generates a completely new phrase by selecting the indices of the winners of four independent three-measure tournaments, where the winners are determined by greatest fitness. This phrase will bear no relationship to its parents, which is mutation in the extreme. Notice, by the way, that measure 57 apparently won a tournament to stay in this phrase.

The last two operators tend to combat the convergence problem, the tendency of GA machinery to converge on slight variations of a "super" individual. In GAs that search for a single best individual, convergence is seldom a real problem, but in GenJam, convergence translates to "the lick that ate my solo." The lick thinner substitutes a random measure for the measure in the phrase that occurs most frequently in the phrase population as a whole. This tends to thin out overly successful measures.

The orphan phrase operator generates a completely new phrase by selecting the winners of four independent three-measure tournaments where the winners are the least frequently occurring measures in the phrase population. This tends to repopulate "orphan" measures that don't appear in any phrase and insure that diversity is maintained.

In demo mode, where only selection is applied and feedback is ignored, phrases are selected with another tournament selection scheme. Three phrases are selected at random, and a combined fitness value is used -- the sum of the phrase's fitness and the average fitness of its constituent measures. Once a phrase has been selected, its combined fitness is halved so that it is less likely to be selected again in the same solo. The result is GenJam's "greatest hits."

4 Training and Performance

After sufficient training, GenJam's playing can be characterized as competent with some nice moments. About two dozen tunes have been prepared for GenJam, either as training tunes or "gig" tunes. The best training tunes have tempos in the range 120 to 180 or so. Faster tunes are too hard for the mentor to keep up with, and slower tunes need sixteenth notes to retain interest. Of the tunes I have prepared, the most frequently used tunes for training are summarized in Table 4, with tempo in beats per minute and the column labeled "C" giving the number of choruses in the solo.

Tune	Form	Tempo	C
Boplicity	32-bar AABA	130	3
Stella by Starlight	32-bar AABA	160	3
Well You Needn't	32-bar AABA	174	3
Bye Bye Blackbird	32-bar AABA	135	3
Gentle Rain	40-bar ABABC	138	3
Lady Bird	16-bar AB	150	4
Bb Blues	12-bar Blues	135	6

Table 4. Tunes Used for Training

The typical training procedure begins by running GenJam in learning mode for three or four solos, to sufficiently sample the populations and provide initial fitness values. Then one can alternate breeding and learning runs until coherent solos begin to emerge. The training tunes are cycled in a fairly random order, with Lady Bird and the Bb Blues being used in learning mode only, since they contain fewer than 24 phrases. Again, the constraint is that 24 new phrases will be generated in a breeding run, and all 24 need to be heard so that they can have some chance of accumulating non-zero fitnesses.

The first few generations of a training session are quite numbing for the mentor. Fitnesses are almost all negative, melodic intervals tend to be large, and the frequency of "nice moments" is very low. Sooner or later, though, a few pleasant licks begin to emerge, and one or two solid phrases tend to appear by the fourth or fifth generation. If a very pleasing measure appears too early, it can become overused in a generation or two and may require the mentor start punishing a previously rewarded lick to thin it out. Typically, at around the tenth generation, a "golden" generation occurs where almost all the new phrases sound reasonable. At this point, the mentor's standards can shift from rewarding anything that sounds musical to rewarding only what really sounds nice.

A preliminary analysis of population statistics gathered over several training runs shows that pitch, interval, and note length distributions shift in expected directions. The initial pitch distribution is

uniform, as would be expected from the initialization procedure, but it gradually "humps up" to look more normal after several generations. Large intervals tend to be bred out fairly quickly, particularly when the note lengths are short. After several generations the interval distribution is heavily skewed toward short intervals, and the average interval shrinks from around seven scale steps to around two. Note and rest lengths also become skewed toward the short end as very long notes and silences are bred out.

5 Extensions and Conclusions

Several enhancements and extensions are under way to improve GenJam's performance. An overhaul of the chord/scale mapping procedure is being designed that will apply knowledge-based techniques to a wider window on the chord progression. This hopefully will correct a few bad scale choices currently made in some progressions.

An attempt will be made to train a neural network to serve as at least a preliminary fitness function for at least the measure population. The strategy will be to extract statistical features correlating with measure fitness to form a feature vector, which will be the input layer to a quick-prop-style neural network. The output layer will be a single node containing the fitness value. The training data will come from the populations that have been saved from the approximately two dozen controlled training runs conducted so far.

Another experiment will seed GenJam's initial population with measures and phrases taken from existing tunes or transcribed solos. A similar exercise will be to merge populations generated in separate training sessions to hopefully get the best of both. GenJam's two levels also could be extended upward to section, chorus, and/or tune levels.

To place GenJam in a larger context, I'll conclude with a brief mention of the role GAs could play in algorithmic composition. GenJam shows that GAs can be a useful tool for searching a constrained melodic space. Other specific compositional tasks should be easy to find, for example, evolving a bass line or percussion sequence or chord progression or series of voicings. These small tasks might even be done concurrently in a more comprehensive system. Evolutionary programming techniques [Koza, 1992], where individuals map to programs or program fragments, also promise to broaden the uses of GAs in music. Finally, the fact that "Genetic Algorithms" is a special topic area at ICMC 94 indicates that others are discovering GAs. It would seem that GenJam is not alone!

References

[Ames and Domino] Cybernetic Composer: An Overview. In M Balaban, K. Ebcioglu and O. Laske (Ed.), *Understanding Music with AI,* AAAI Press, Cambridge, MA, 186-205, 1992.

[Coker, 1964] Jerry Coker. *Improvising Jazz*. Prentice-Hall, Englewood Cliffs, NJ, 1964.

[Dannenberg, 1993] Roger B. Dannenberg. *The CMU MIDI Toolkit, Version 3*. Carnegie Mellon University, Pittsburgh, PA, 1993.

[Fry, 1984] C. Fry. Flavors Band: A Language for Specifying Musical Style. *Computer Music Journal* **8** (4) pp. 20-34, 1984.

[Gannon, 1991] Peter Gannon. Band-in-a-Box. PG Music, Inc., Hamilton, Ontario, 1991.

[Giomi and Lagabue, 1991] Francesco Giomi and Marco Ligabue. Computational Generation and Study of Jazz. *Interface* **20**, pp. 47-63, 1992.

[Goldberg, 1989] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[Haerle, 1989] Dan Haerle. *The Jazz Sound*. Hal Leonard, Milwaukee, WI, 1989.

[Haerle, 1980] Dan Haerle. *The Jazz Language*. Studio 224, Miami, 1980.

[Haggerty, 1991] Michael Haggerty. Evolution by Esthetics, an interview with William Latham and Stephen Todd. *IEEE Computer Graphics and Applications* **11**, pp. 5-9, 1991.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[Horner, et al., 1993a] Andrew Horner, Andrew Assad and Norman Packard. Artificial Music: The Evolution of Musical Strata. *Leonardo* **3**, pp. 81, 1993.

[Horner et al., 1993b] Andrew Horner, James Beauchamp, and Lippold Haken. Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. *Computer Music Journal* **17** (4) pp. 17-29, 1993.

[Horner and Goldberg, 1991] Genetic Algorithms and Computer-Assisted Music Composition. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kauffman, San Mateo, CA, 1991.

[Iverson and Hartley, 1990] Eric Iverson and Roger Hartley. Metabolizing Music. In *Proceedings of the 1990 International Computer Music Conference*, ICMA, San Francisco, 1990.

[Koza, 1992] J. R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1990.

[Levitt, 1981] David Levitt. *A Melody Description System for Jazz Improvisation*. Master's thesis, MIT, Cambridge, MA, 1981.

[Penneycook et al., 1993] Bruce Penneycook, Dale R. Stammen, and Debbie Reynolds. Toward a Computer Model of a Jazz Improviser. In *Proceedings of the 1993 International Computer Music Conference*, ICMA, San Francisco, 1993.

[Russell, 1959] George Russell. *The Lydian Chromatic Concept of Tonal Organization for Improvisation*. Concept Publishing, NY, 1959.

[Sabatella, 1992] Marc Sabatella. *A Jazz Improvisation Primer*. USENET, 1992.

[Sims, 1993] Karl Sims. Interactive Evolution of Equations for Procedural Models. *The Visual Computer* **9** (8), pp. 466-476, 1993.