Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness

John A. Biles

Department of Information Technology
Rochester Institute of Technology
Rochester, NY 14623
jab@it.rit.edu

Abstract

This paper focuses on a successful attempt to eliminate the fitness bottleneck in GenJam, an interactive genetic algorithm that performs jazz improvisation, by eliminating the need for fitness. This was accomplished by seeding an initial population of melodic ideas with phrases selected from a database of published jazz licks, and employing an intelligent crossover operator to breed child licks that tend to preserve the musicality of their parents. After a brief overview of the changes made to GenJam's architecture, the paper describes the mapping of licks to measure and phrase individuals in GenJam Normal Form. The intelligent crossover operator on phrases is then described, along with a discussion of measure mutations performed during a solo to insure that repeated measures are altered in interesting ways. The paper concludes with a discussion of the implications of removing fitness from a genetic algorithm and whether the result still qualifies as a genetic algorithm.

1 BACKGROUND

Fitness in most genetic-algorithm-based music and art systems derives from aesthetic judgements. Since aesthetic judgements are not often amenable to algorithmic implementation, these systems usually end up as interactive genetic algorithms (IGAs), which depend on the feedback of a human mentor [Bentley 1999]. The mentor must experience all the individuals in a population of competing works or pieces and somehow indicate each one's artistic or musical merit. In other words, the objective function in these domains is typically a *subjective* function.

IGAs suffer from a bottleneck caused by the need for the mentor to experience and provide feedback for each individual. This *fitness bottleneck* is particularly narrow in temporal domains like music, where individual population members must be experienced one-at-a-time and in real time [?? 1994]. This can set practical limits on population size and number of generations, which in turn often leads to the design of unusual genetic operators, all in an attempt to evolve acceptable individuals before the mentor gives up.

1.1 GenJam REVISITED

This is certainly the case with GenJam [Biles 2001], an IGA that performs jazz improvisations in two modes. In the first mode, GenJam learns to improvise full-chorus solos by evolving hierarchically interrelated populations of measures and phrases under the interactive guidance of a human mentor. Individuals in the measure population map to sequences of eighth-note-length events that make up a measure of melodic content. Three types of events can be represented. *Rest* events turn off the note currently being played, if there is one. *Hold* events hold the current note, if there is one. *New-note* events turn off the current note and begin a new note, using the event number as an index into an array of pitches derived from the current chord in the tune. In this way, GenJam never plays a theoretically wrong note. Individuals in the phrase population map to sequences of four measures from the measure population. Melodic material that has been encoded as measure and phrase individuals is referred to as being in GenJam Normal Form (GJNF).

To create a GenJam soloist, the mentor starts with a random measure and phrase population, listens to GenJam improvise on some tune to the accompaniment of a synthesized rhythm section, and supplies feedback by indicating "good" or "bad" whenever so moved during the improvisation. New measure and phrase individuals are bred in both populations using tournament selection, single-point crossover, musically meaningful mutation, and replacement with a 50% generation gap. The mentor usually evolves several soloists in various musical styles prior to performing, and a typical GenJam gig with the author's Virtual Quintet will feature a dozen or so pre-trained soloists.

In its second improvisational mode, GenJam *trades fours* with a human in real time during a performance by using its genetic representations and mutation operators to evolve the human's last four measures into what it then plays in the next four measures. GenJam listens to the human performer play a four-measure phrase, maps what it heard to a phrase and four measure chromosomes, mutates those chromosomes, and immediately plays the result in the next four measures as its response. The mutation operators, then, act to develop the human's phrase in musically stimulating ways. The details of GenJam's architecture, chromosome structures and genetic operators have been documented elsewhere [Biles 2000 & 2001].

1.2 IMPROVING GenJam SOLOS

While GenJam improvises quite successfully in both improvisational modes, its performance when trading fours is superior to its performance taking full-chorus solos. This is partly due to the spontaneous interactivity inherent in trading fours--a musical conversation is usually more stimulating than a musical monologue. However, another difference between GenJam's full-chorus solos and the material it plays when trading fours stems from the origin of that material. Individuals that make up the measure and phrase populations of an evolved soloist began their evolution as random strings. After evolving for several generations, these musical ideas reach a level that is competent and often pleasing but seldom compelling. However, the phrase and four measure chromosomes that are mutated when trading fours are a fairly close approximation to a four-measure phrase played by a presumably accomplished human jazz musician.

The "presumably accomplished" part is important because it makes it possible for GenJam to evolve responses without the use of a fitness function in the 15 milliseconds or so allotted to it when trading fours. If the four-measure phrase played by the human is good and the mutation operators are safe, then the mutated phrase will be good, and there is no need to assign fitness. If the human performer is bad, however, then all bets are off, and GenJam's responses will likely sound bad as well. Garbage in, garbage out, as the saying goes.

GenJam's success trading fours prompted the author to make another attempt to ease GenJam's fitness bottleneck for this paper. Previous attempts have included trying to develop neural-network-based fitness functions [Biles et al 1996], which did not work, and seeding the initial measure population with individuals generated by a fractal generator, which produced individuals that statistically resembled measure individuals from a mature, trained soloist [Biles 1998]. The fractal generator certainly helped by making the initial population more musical, which made training easier. However, when soloists were evolved in the absence of feedback, they ended up markedly less pleasing than mentor-trained soloists, so fitness was still deemed necessary [Biles 1999].

This paper describes a successful attempt to create a fitness-free or *autonomous* GenJam. By seeding the initial population with licks derived from human performance, breeding new licks with an intelligent crossover operator, and mutating repeated measures during performance, autonomous GenJam (AutoGenJam) is able to play full-chorus solos that are generally superior to those played by soloists trained by a mentor. We begin with a description of AutoGenJam's revised architecture, then discuss the mapping of licks to GJNF, our intelligent crossover operator, and the role of mutation in keeping a soloist fresh. We conclude with a discussion of whether AutoGenJam is still a genetic algorithm, once fitness is no longer necessary.

2 AUTONOMOUS GenJam

Figure 1 shows AutoGenJam's system architecture. As described elsewhere [Biles 2001], the Rhythm and Head Sequence files are MIDI files that provide rhythm accompaniment and arranged harmony parts for the tune being played, and the Chord Progression and Choruses files describe the structure of the tune. The MIDI Params file provides parameters to configure the tone generator GenJam and the rhythm section will use. The human Performer trades fours with GenJam, as described above. Missing from previous versions of GenJam are the human Mentor and the two Population files, one for measures and the other for phrases. The Mentor is unnecessary because there is no need for fitness, and the stored population files have been replaced by a Licks Database, which supplies four-measure licks from which AutoGenJam creates a measure population of 64 individuals and a phrase population of 48 individuals. These populations use the same chromosome structures as previous versions of GenJam, but they are generated fresh for each tune and are not stored.

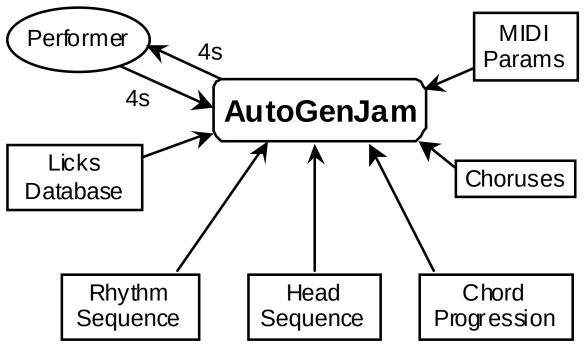


Figure 1: Autonomous GenJam System Architecture

2.1 POPULATIONS FROM LICKS

The creation of the measure and phrase populations from the Licks Database is summarized in Figure 2. The database itself is made up of four-measure licks encoded from a recently published vocabulary of *1001 Jazz Licks*

[Shneidman 2000], hereafter referred to as 1001 Licks. Multiple licks databases can be created to reflect different jazz styles. Shneidman includes a generous portion of "essential" licks, along with swing, bebop, hard bop, post bop, non-harmonic, and fusion/funk stylistic licks. All the licks are four measures long (how convenient!) and are presented in the context of specific chord progressions, including licks for the first four measures of 20 standard tunes.

```
Randomly select 16 licks from a licks database
Read those 16 licks to get 64 measure individuals
Initialize first 16 phrase individuals to represent licks
Create 32 child phrases via crossover on original licks
```

Figure 2: Algorithm for Creating a Soloist

Licks that are selected for inclusion in a licks database are mapped to GJNF by hand. Since GJNF represents measures and phrases independent of any chord progression, the items in a licks database are actually more general than the licks in 1001 Licks and can be played against any chord sequence.

Table 1: Chord/Scale Mappings Using C as Chord Root with Restored Avoid Notes in *Italics*

Chord	**Scale**
CMaj7, C6, C	C D E ***F*** G A B
C7, C9, C13	C D E ***F*** G A Bb
Cm7, Cm9, Cm11	C D Eb F G Bb
Cm7b5	C ***Db*** Eb F Gb Ab Bb
Cdim	C D Eb F Gb Ab A B
C#5	C D E F# G# A B
C7#5	C D E F# G# A#
C7#11	C D E F# G A Bb
C7alt	C Db D# E Gb G# Bb
C7#9	C Eb E G A Bb
C7b9	C Db E F G ***Ab*** Bb
CmMaj7	C D Eb F G A B
Cm6	C D Eb F G A
Cm7b9	C Db Eb F G A Bb
CMaj7#11	C D E F# G A B
C7sus	C D E F G A Bb
CMaj7sus	C D E F G A B

The chord-scale mappings used to convert new-note events to actual pitches were altered somewhat for AutoGenJam by restoring some "avoid notes" to accommodate the more standard seven-note scales implied by the licks in 1001 Licks. The modified chord scale mappings are illustrated in Table 1, with italic entries representing notes restored for AutoGenJam.

The two criteria for including a lick in a licks database represent a modest fitness function. Licks are included if they have at least one new-note event in each measure and if the rhythm can be mapped to eighth-note multiples without losing the essence of the rhythmic structure of the lick.

The first criterion eliminates long silences and assures that notes won't be held through too many chord changes. Specifically, a few licks in 1001 Licks end with a long note that begins at the end of the third measure and ties into the last measure. In many four-measure phrases, the chord in the third measure creates dissonance, which is then resolved by the chord in the fourth measure. When GenJam performs notes that are tied across chord boundaries, the scale used for selecting the pitch for that note comes from the measure in which the note began, which in this case often results in a dissonant note being held through a consonant measure. By insuring that each measure has at least one new-note event, any dissonance has a better chance of resolving. 121 of the 1001 Licks were eliminated by this criterion.

The rhythmic criterion for inclusion in the licks database eliminates licks whose rhythmic structure cannot be mapped to the eighth-note multiples that GenJam can represent in its chromosome structure. Most of the licks in 1001 Licks contain only eighth-note multiples and map easily. Some licks contain sixteenth-note or eighth-note-triplet ornaments that can be simplified to eighth notes without losing the essence of the melodic contour. However, some phrases are built on sixteenth or triplet structures that cannot be simplified without losing the essence of the phrase, so those licks were eliminated. This criterion eliminated virtually all the fusion/funk licks, nearly half of the hard-bop licks and occasional licks in other styles. The fusion/funk licks were no great loss to the author because he prefers not to play in that style anyway…

Figure 3 shows two licks from 1001 Licks to illustrate their mapping to GJNF. These licks will also be used to illustrate the intelligent crossover operator described in the next section. The licks selected were numbers 563 and 569, both of which came from Swing Progression # 1. The numbers beneath each lick are the GenJam event numbers encoded from the notes in the licks. 0 represents a rest event; 15 represents a hold event; and 1-14 represent new note events, where 1 maps roughly to D above middle C, 7 maps roughly to C an octave above middle C, and 14 maps roughly to C an octave above that. The specific pitches for each scale degree were given in Table 1. The first lick is structurally interesting in that it is made up of a two-measure motif that is repeated with minor variations. This lick is also diatonic, meaning that all the notes are in scales suggested by the chords in the accompanying progression.

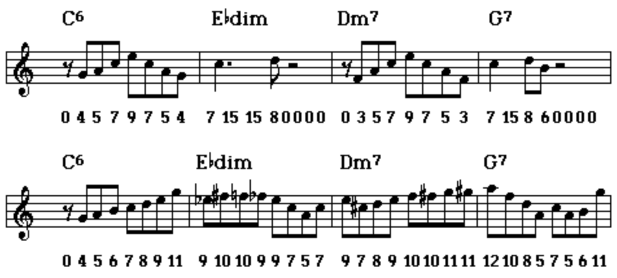


Figure 3: Sample Licks with Event Mappings

The second lick in Figure 3 is chromatic, which means that there are notes that do not come from the scales suggested by the chords. Specifically, there are chromatic runs in the first half of the second measure and the second half of the third measure. Since GenJam maps repeated new-note events to chromatic passing tones when a measure is performed, chromatic runs can be represented in GJNF by repeated new-note events, hence the repeated 10's and 9's in the second measure and the repeated 10's and 11's in the third measure. However, the C# in the third measure, which is a chromatic approach tone not found in the scale suggested by a Dm7 chord, cannot be represented in GJNF, so it was coded as a diatonic approach tone. In this case the C# was encoded as a 7, which will be performed as a C, a note that is in the scale suggested by a Dm7 chord.

0	0	1	2	3
1	4	5	6	7

Phrase Individuals

0	0	6	7	9	11	9	7	6
1	9	15	15	10	0	0	0	0
2	0	5	7	9	11	9	7	5
3	9	15	10	8	0	0	0	0
4	0	4	5	6	7	8	9	11
5	9	10	10	9	9	7	5	7
6	9	7	8	9	10	10	11	11
7	12	10	8	5	7	5	6	11

Measure Individuals

Figure 4: Final GJNF for the Licks in Figure 3

While mapping a lick to GJNF, a transposition often needs to be performed to normalize the lick to a suitable range for GenJam. The fourteen possible new-note events map to just under two full octaves, which is less than the nearly three-octave range used for the licks in 1001 Licks. In our example, the second lick is already in a suitable range, but the first lick is a bit low. To correct that, the first lick was transposed up a third by adding 2 to each new-note event while leaving the hold and rest events unchanged. The resulting phrase and measure individuals for these two licks are shown in Figure 4, which assumes that these two licks are the first two processed in creating the measure and phrase populations, hence the indices beginning at 0.

2.2 INTELLIGENT CROSSOVER

One third of the 48 phrases that represent a soloist are GJNF approximations of licks from a licks database, like our two examples. These 16 original licks are made up of 64 measure individuals, which comprise the entire measure population for the soloist. The remaining two thirds of the phrases are children of pairs of the original 16 phrase individuals, where pairs of children are created via an intelligent crossover operator. Crossover points in phrase individuals can occur only at interior measure-pointer boundaries, which means there are only three potential crossover points in a phrase individual, as illustrated in Figure 5.

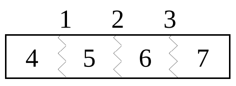


Figure 5: Three Potential Crossover Points in Phrase 1

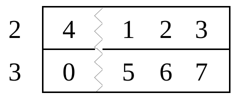
The crossover operator performs either single or double-point crossover at one or two of the potential crossover points. The six possible crossovers, then, are three single point crossovers at points 1, 2 and 3, and three double-point crossovers at points 1 and 2, 1 and 3, and 2 and 3. The choice of which of these possibilities will be performed is determined by the algorithm in Figure 6.

```
At each potential crossover point
    Compute horizontal intervals in both parents
    Compute horizontal intervals in both children
    Compute interval differences both ways
    Return smaller difference as "fitness" for that point
Select crossover point with smallest difference
If a second crossover point has same difference
    Select it too
Perform crossover(s) at selected point(s)
```

Figure 6: Algorithm for Intelligent Crossover

The algorithm determines which crossover point(s) produce children that most closely preserve the horizontal intervals at the crossover point(s) in the two parents. A *horizontal interval* is the interval between two adjacent notes and is defined here to mean the difference between the first new-note event of the measure following the crossover point and the last new-note event of the measure preceding the crossover point. Since there are two parents and two children, there are two possible comparisons between parents and children. Both comparisons are computed, and the one that yields the smaller total difference is selected as the difference for that crossover point.

The differences for the three potential crossover points are then used to select the actual crossover point(s). If one difference is less than the other two, it is selected as the only crossover point, and a single-point crossover is performed at that point. If two crossover points tie for the smallest difference, they are both selected, and a double-point crossover is performed. If all three differences are identical, a double-point crossover is performed at points 1 and 3.



2	4	1	2	3
3	0	5	6	7

Figure 7: Child Phrases Showing Crossover Point

In our example, which can be worked out by the reader with not enough to do, the differences are 0 for crossover point 1, 6 for point 2, and 6 for point 3, so a single point crossover was performed at crossover point 1. The resulting phrase individuals are shown in Figure 7, and the final child licks, as played over the original chords are shown in Figure 8. Notice that the chromatic runs mentioned earlier have been altered slightly in the second phrase due to GenJam's somewhat idiosyncratic handling of repeated new-note events as chromatic passing tones.

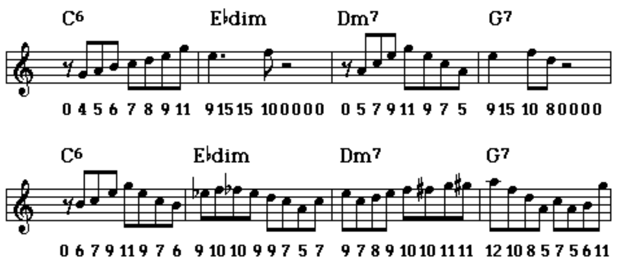


Figure 8: Child Phrases Played Over Original Chords

2.3 MUTATIONS TO INSURE FRESHNESS

Since a given measure occurs in three different phrases in a soloist, it may be repeated during the course of a solo on a specific tune. Furthermore, since building blocks of two or three measures tend to survive in child phrases, sequences of two or three measures could be repeated in a solo. This is more likely to occur in tunes with longer forms or tunes where GenJam is playing double time, but it often occurs even in standard 32-bar forms, which only consume eight of the 48 available phrases per chorus. Even though the chords, and thus the scales used to determine actual pitches, will likely be different the second or third time a measure individual is played, the listener may detect the repetition and conclude that the soloist has run out of new ideas and is simply reusing old ones.

To combat this, AutoGenJam mutates measures that have already been played as part of an earlier phrase in the tune. The mutations used are the subset of GenJam's musically meaningful measure mutations that are used when trading fours with a human [Biles 1998]. The specific operators used are a random transposition up or down one to three steps, a range-corrected inversion that essentially plays the measure upside down, a retrograde that reverses the events in the measure, and a retrograde inversion that performs both the inversion and reverse.

These mutations are some of the same compositional devices used by human composers and arrangers to develop melodic material. As such, they tend to develop measures in musically pleasing ways, which is important in an environment with no fitness available to identify and weed out poor mutations. These mutation operators are considered *safe*, then, because they reliably develop good measures into good measures, thereby preserving the musicality of the original measures.

Actually, these mutations are the last in a series of steps that insures a fresh soloist for every tune. The 16 licks chosen for a soloist are selected randomly from a database of two to several times that number, so a different subset is chosen for each tune. Once a set of 16 licks is chosen to be 1/3 of the phrases, the other 2/3 of the phrases are created by randomly breeding pairs of the original 16 phrases. This would result in different children phrases even if the same 16 original licks were chosen over again. Finally, measures that are repeated in a solo are mutated the second and third time they are used so that they will be less recognizable but still acceptable when repeated.

2.4 AutoGenJam VS. GenJam

Anecdotal evaluations by listeners indicate that the solos played by AutoGenJam are generally superior to those played by GenJam using well-trained soloists. AutoGenJam solos were characterized as more interesting, more coherent, more human, less random, and less mechanical. The author also finds AutoGenJam's solos to be more engaging, and definitely less repetitive. The author plays several gigs per month and practices with GenJam frequently, so he knows his trained soloists all too well. This induces in the author a sense of boredom with GenJam solos that, fortunately, is not usually shared by audience members who haven't heard GenJam before. However, in most gigs a certain song style will be played more frequently than the others, which results in the soloist trained on that style being heard more often. This can lead to a sense of déjà vu, even in audience members who have not heard GenJam before. While the author does train new GenJam soloists occasionally to supplement or replace soloists that have gotten too familiar, doing so takes time, which the author typically doesn't have when a gig is imminent.

AutoGenJam provides a vastly expanded pool of superior melodic ideas without having to evolve them from scratch. Since a new soloist is generated for each tune from this much larger pool, the solos are fresher from one tune to the next, even if the same licks database is used over and over. In addition, the licks used to generate each soloist are more coherent and musical than the evolved licks of a mature trained GenJam soloist, so the solos are not only fresher but better.

One nagging issue, at least to the author, is the use of non-original licks to seed AutoGenJam's populations. While the standard pedagogical practice for apprentice jazz musicians is to listen to, transcribe, and memorize classic jazz solos, the author has always eschewed this tradition in an effort to avoid sounding like someone else. Since GenJam starts with random strings, any licks that it evolves are its own, or at least the mentor's own, since the mentor must have liked the survivors enough to not breed them out. AutoGenJam is more reminiscent of some jazz musicians the author has encountered at jam sessions who seem to have memorized every solo by their personal favorite and play solos that are essentially strings of quotes from their hero. These musicians often get a good response from the audience, usually because they are quoting great music, but their abilities as an improviser are suspect. AutoGenJam at least develops its memorized licks, both harmonically, with its chord-scale mappings, and melodically, with its intelligent crossover and mutation operators.

An alternate way of seeding the initial population is currently being developed by the author to address this originality issue. When the human performer takes a full-chorus solo, AutoGenJam will listen to the human's solo and build measures and phrases in the same way it does when trading fours. It then can mate these with each other or with licks from 1001 Licks and generate populations that integrate ideas from the human's solo into the mix. AutoGenJam would still be memorizing the licks of others, but at least it wouldn't be doing it strictly by the book.

The biggest problem being worked on now is, yes, a fitness function to determine if a human's four in a solo is worth keeping and breeding. This is not an issue when trading fours because the occasional bad four is gone as soon as it is played, but a bad lick breeding in a fitness-free environment could ruin a soloist. Hopefully, future papers will document the success of this approach.

3 CONCLUSIONS

One question remains: Is AutoGenJam still a genetic algorithm? Most authorities list fitness evaluation as a requisite step in a genetic algorithm. Bentley [1999], for example, lists "initialisation, mapping, evaluation, selection, fertility, reproduction and termination" as stages used by a simple genetic algorithm. GenJam clearly has all these stages, but AutoGenJam has no evaluation stage, and the selection and fertility stages are uniform and random rather than preferential.

Goldberg [1999] lists encoding to chromosomes, determining "fitness to purpose," using a "population of encoded solutions," employing "selection, recombination and mutation," and preferring "better solutions to worse ones." Again, GenJam clearly does all this, but AutoGenJam does not determine fitness, other than the elimination of licks that do not meet the two criteria for inclusion in a licks database. AutoGenJam also does not prefer better solutions to worse ones because all solutions are deemed good, once they make it into a licks database.

If genetic algorithms are viewed from the perspective of a generate-and-test strategy, however, AutoGenJam begins to look more like a GA. Genetic algorithms use initialization, recombination, and mutation to *generate* new solutions that are then *tested* by fitness. In classic GA's, these operations are random or at least "dumb," and any knowledge of the problem domain resides primarily, if not entirely, in the fitness function. This is what makes GA's attractive for problems that are not well understood, because all that is usually necessary for the GA machinery to work is the ability to tell whether one individual is better than another. However, many GA developers seed their initial populations with promising individuals, and knowledge-based crossovers and mutations are certainly nothing new. These techniques effectively decentralize domain knowledge and distribute it across initialization, recombination and/or mutation. AutoGenJam simply takes this to the logical extreme by removing all domain knowledge from fitness and placing it in the initialization, crossover, and mutation operators. Fitness, then, serves no useful purpose and can simply be eliminated.

The lack of a fitness function and the resulting lack of selection pressure in AutoGenJam also serves a useful purpose in that it eliminates the convergence problem that has plagued GenJam [Biles 1998]. Since the search in both systems is for a good soloist, not for a good solo, and certainly not for a super phrase or measure, convergence on a small number of highly fit phrases and measures is disastrous. Without selection pressure, convergence is no longer a problem.

Our conclusion, then is that AutoGenJam is still a genetic algorithm, albeit an unusual one. It uses the evolutionary paradigm to explore a melodic space of jazz licks in such a way that it never takes a truly wrong turn. If only it were so easy for aspiring human jazz musicians…

References

Peter Bentley (1999). An Introduction to Evolutionary Design by Computers. In Peter J. Bentley (ed.), *Evolutionary Design by Computers*, pp. 1-73. San Francisco, Morgan Kaufmann.

John A. Biles (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos. In *Proceedings of the 1994 International Computer Music Conference*, pp. 131-137. San Francisco: ICMA.

John A. Biles, Peter G. Anderson, and Laura W. Loggi (1996). Neural Network Fitness Functions for a Musical IGA. In *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA'96) and Soft Computing (SOCO'96)*, pp. B39-B44, Reading, UK: ICSC, Academic Press.

John A. Biles (1998). Interactive GenJam: Integrating Real-time Performance with a Genetic Algorithm. In *Proceedings of the 1998 International Computer Music Conference*, pp. 232-235. San Francisco: ICMA.

J. A. Biles (1999). Life with GenJam: Interacting with a Musical IGA. In *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 652-656, Tokyo: IEEE.

John A. Biles (2000). GenJam in Perspective: A Tentative Taxonomy for Genetic Algorithm Music and Art Systems. In *Proceedings of the 2000 GECCO Workshop Program*, pp. 133-135, Las Vegas: ISCEG.

John A. Biles (2001). GenJam: Evolution of a Jazz Improviser. In P. J. Bentley and D. W. Corne (ed.), *Creative Evolutionary Systems*. San Francisco: Morgan Kaufmann.

David Goldberg (1999). The Race, the Hurdle, and the Sweet Spot: Lessons from Genetic Algorithms for the Automation of Design Innovation and Creativity. In Peter J. Bentley (ed.), *Evolutionary Design by Computers*, pp. 105-118. San Francisco: Morgan Kaufmann.

Shneidman, Jack (2000). *1001 Jazz Licks*. New York: Cherry Lane Music Company.