A Grammatical Approach to Automatic Improvisation

Robert M. Keller, David R. Morrison
Harvey Mudd College
Claremont, California, USA
keller@hmc.edu, dmorrison@hmc.edu

Abstract— **We describe an approach to the automatic generation of convincing jazz melodies using probabilistic grammars. Uses of this approach include a software tool for assisting a soloist in the creation of a jazz solo over chord progressions. The method also shows promise as a means of automatically improvising complete solos in real-time. Our approach has been implemented and demonstrated in a free software tool.**

Keywords— **jazz, improvisation, educational software, probabilistic context-free grammar, melody generator.**

I. Introduction

Our group set out to develop a tool for assisting improvisors in creating jazz solos in the context of given chord progressions ("changes" in jazz musicians' parlance (JMP)). An initial approach involved creating a large database of existing melodic fragments ("licks" in JMP) from which the user could choose, to help create melodic lines or fill gaps in these lines. One problem with this approach is that there are many different chord progressions, even when normalized to a given tonal center, which would thus require an extremely large database. Moreover, the database needed to be manually created, which was very labor-intensive, albeit instructive to the creator.

An alternate approach that does not have the above difficulties is to have the software dynamically generate novel licks. While this idea has great appeal, it is less than obvious how to make it work so that the results sound like convincing jazz solos. Choices of notes that are largely random, even ones drawn entirely from the chords, tend not to sound good, especially rhythmically. Some additional structuring method is required to make the notes fit together in a manner that will be pleasing to the ear.

We have developed an approach based on the idea of a formal grammar, augmented with certain pragmatic devices, which seems to be largely successful in achieving our objective of creating good-sounding melodic lines. The approach has been realized as a feature of a free software tool for assisting improvisors in the creation of better solos. The tool is called *Impro-Visor* (for "Improvisation Advisor") and has been available on the world-wide web since January 2006 [1]. A preliminary paper describing our tool was presented in [2].

II. Structure of the Paper

Having reviewed the background and motivation of our project, we will briefly review the concept of a context-free grammar, as it occurs in computer science and linguistics. Then we show how a grammar can be used to generate plausible rhythmic and melodic sequences for jazz. Playable examples will be presented that show the effectiveness of the approach. Our approach has been implemented and has been tested in the past year as part of a broader educational software tool, which proved beneficial in a jazz improvisation course taught by the first author and has also been used by various third parties.

III. Related Work

The motivation for our use of grammar was based primarily on implementation needs. It was only during writing the final version of this paper that we became aware of the many references that suggested using grammars in one form or another for various aspects of music analysis and synthesis. Early work includes that of Winograd [3], Lindblom and Sundberg [4], Longuet-Higgins [5] and Roads [6], to name a few. Rader's work [7] used probabilistic grammars to generate traditional melodies. Hughes [8] surveyed the use of grammars for non-western musics, and Bel and Kippen [9] used them for patterns in Indian drumming. Please see Roads and Strawn [10], Cope [11], and McCormack [12] for more thorough surveys, the latter including use of L-systems and Markov chains. Recent work of Assayag and Dubnov [13] demonstrate the use of factor oracles to generate improvised sequences.

Lerdahl and Jackendoff [14] used a generative grammar theory for music analysis, which is referenced by Horowitz [15] as an approach to jazz analysis. Steedman [16] described the use of a grammar for analyzing jazz chord progressions, but not melodies, and Johnson-Laird [17] elaborated on this approach. Earlier work by Johnson-Laird [18] also suggests using regular grammars for explaining rhythmic sequences. Despite this large set of references, we believe that the particular way in which we employ grammars and the meaning we assign to the terminal symbols has adequate elements of novelty. Because we use *probabilistic* grammars, we mention Temperley's recent book [19], which describes wide uses of probabilistic models for music analysis, including both pitch and rhythm. The work of Ulrich [20] and, although it does not use grammars, Grachten [21] seem to share some of the spirit of our own approach.

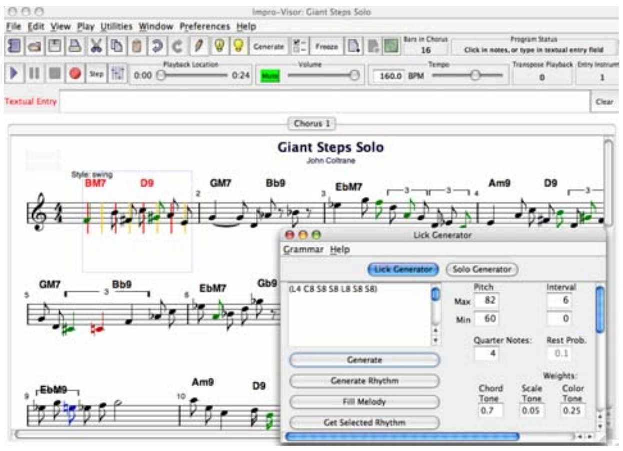


Fig. 1. Screen shot of Impro-Visor

Work on jazz melody generation includes Biles' GenJam [22] and the work of Papadopoulos and Wiggins [23], both of which use genetic algorithms, thus requiring a fitness function to perform genetic selection, and Thom's BoB [24], which used a statistical learning approach. Band-in-a-Box [25] is a commercial tool that will generate choruses without user interaction. Although effective for non-interactive solo generation, its methods are proprietary, and appear to rely mostly on a preconstructed database of melodic fragments.

IV. Brief Background on Jazz Soloing

A jazz solo consists of a melody improvised over a given chord progression. The progression usually consists of chords chosen to support the original melody of a song, and is usually somewhat standardized, although chord substitutions and embellishments are often introduced. Common practice would be for the group to play the original tune (called the "head" in JMP), then each soloist would improvise some number of "choruses" over the changes, then the group would typically play the head a final time.

The challenge for the soloist is to have sufficient creative ideas to be able to play one or more choruses. Good preparation for this kind of creation entails knowing the melody and changes, as well as the harmonic underpinnings of the changes.

Impro-Visor was designed to assist the soloist, who may not yet have a strong theory background, in the process of creating solos. Our intention is to provide this as a practical end-user tool, not simply a research vehicle, so we have included niceties such as MIDI-playback for usability. Fig. 1 provides a screen-shot of Impro-Visor in action.

V. Automating Melody Generation

We would like our tool to provide a very large repertoire of melodic segments. Our initial approach was to construct these manually or transcribe them from recorded sources. Manual construction is enjoyable, and turns out to be as educational, if not more educational, than using the segments to construct complete solos. However, manual construction is also a slow process. It would be much better if we could automate the generation of these segments, either on-line while the tool is being used, or off-line, saving the segments in the vocabulary file in which such segments currently reside.

On-line generation is somewhat risky. Unless one has a very reliable algorithm for doing it, the results can be poor and therefore misleading to the student. Of course the ear can be used as the ultimate selection device, and it is easy to backtrack to a different sequence, but the beginner's ear is often not sufficiently developed, so it would be better to have every generated segment sound good.

Historically, we intended to create a training corpus for machine learning research. We had originally constructed a "lick triage" device as part of a software improvisation assistant tool, from which we learned some things about the problem, as we now describe. The tool allowed one to specify (unordered) sets of preferred tones over two-chord sequences, although the idea could easily be extended to more chords. The preferred tones are essentially scale tones and color tones, information that is already available in a vocabulary. Minimum and maximum duration for notes were set by the user. When triggered by the user, the tool generated a random sequence of pitches within the realm of the parameter settings. The user could then accept this sequence, or reject it and try another.

One surprise that resulted was that, when we simply avoided repeated notes and used a uniform duration of eighth-notes, most of the segments were acceptable jazz melodies, at a level above 90 percent. When the duration was not uniform, say a mixture of eighth and quarter notes, the acceptance rate was not as high, but still encouraging. Some interesting syncopated melodies were observed, consistent with the intended use for jazz. The acceptance rate deteriorated significantly when longer licks were stipulated, and seemed to suggest that rhythmic aspects are at least as important as pitch aspects when characterizing acceptable jazz melodies. Being encouraged by the results from fairly simple technology, we set out to develop a better tool based on grammars, as next describe.

VI. Formal Grammars

While a review of grammars might not be necessary for the computer scientist or linguist (who can thus skip this section), their use is less common in music, so we briefly present the basic concepts here. There are several categories of grammar, but the type that are perhaps the most manageable while still serving our purpose are the *context-free grammars*, so we restrict ourselves to this family in the current exposition.

The purpose of a grammar is to generate sequences of symbols. In our case, the symbols will be interpreted as rhythmic values and tonal families of notes.

A context-free grammar (CFG) consists of the following components:

1) The **terminal alphabet** is a set of symbols that directly form the objective sequence. In our case,

these symbols are a step away from being rendered as musical notes.

2) The **auxiliary alphabet** (also called **non-terminal alphabet**) is a set of symbols that serve to construct sequences of terminal symbols, but are not themselves terminal symbols. They are analogous to scaffolding in building construction.
3) The **start symbol** is a symbol in the non-terminal alphabet that is used as the root in generating a sequence.
4) **Productions** are rules that create new sequences by replacing symbols in the old sequence with sequences of new symbols.

Here is a simple example of a CFG outside of a musical context. It generates all strings of matched parentheses

1) **terminal alphabet**: { '(', ')' }, the set of parentheses.
2) **auxiliary alphabet**: { S, T }, a set of two symbols.
3) **start symbol**: S
4) **productions**:
   - $S \rightarrow ()$
   - $S \rightarrow (T)$
   - $T \rightarrow S$
   - $T \rightarrow TT$

Below are some sequences generated by the above grammar. At each step, an auxiliary symbol in the previous string that occurs on the left of a production is replaced with the sequence on the right of the same production, leaving the other symbols unchanged. Only strings consisting entirely of terminal symbols are considered to be in the set of strings generated by the grammar.

- $S \rightarrow ()$
- $S \rightarrow (T) \rightarrow (S) \rightarrow (())$
- $S \rightarrow (T) \rightarrow (TT) \rightarrow (ST) \rightarrow (()T) \rightarrow (()S) \rightarrow (()(T)) \rightarrow (()(S)) \rightarrow (()(()))$

and so on

VII. Grammars for Rhythmic Sequences

We use a grammar to generate coherent skeletal rhythmic sequences, then fill those sequences with appropriate notes. For example, suppose we just want to generate an arbitrary number of measures of half notes and quarter notes, with no syncopation. A suitable grammar might be:

- $S \rightarrow M$
- $S \rightarrow MS$
- $M \rightarrow HH$
- $H \rightarrow h$
- $H \rightarrow qq$

(In the Impro-Visor tool, the grammar is actually represented using S-expressions [26]. We present it here in the form shown for greater readability.) Here the terminal alphabet is { h, q }, representing half-notes and quarter-notes respectively. The auxiliary alphabet is { S, M, H } where S represents a sequence of one or more measures, M represents a measure, and H represents a half-measure. Some example derivations are:

- $S \rightarrow M \rightarrow HH \rightarrow hH \rightarrow hh$
- $S \rightarrow M \rightarrow HH \rightarrow hH \rightarrow hqq$
- $S \rightarrow M \rightarrow HH \rightarrow qqH \rightarrow qqh$
- $S \rightarrow MS \rightarrow HHS \rightarrow qqHS \rightarrow qqhS \rightarrow qqhHH \rightarrow qqhqqH \rightarrow qqhqqq$

To allow syncopation, we would need to add to the previous set productions that allow a half-note to straddle the mid-point of a measure and the bar lines, such as in:

- $S \rightarrow HS$
- $S \rightarrow qS$

In this case, a much richer set of possible sequences is generated. In particular, every sequence of half and quarter notes is generated by this grammar. Of course, we could use terminal symbols that represent rests as well as notes.

The advantage of a grammar for representing rhythmic sequences is that a well-constructed grammar can be used to avoid awkard rhythms, such as a sixteenth-note followed by a half-note followed by a sixteenth-note then an eighth-note.

VIII. Probabilistic Grammars

So far, a grammar is completely non-deterministic and void of any sense of distribution of the types of sequences generated. By adding a probability to the application of a production, a semblance of style can be introduced. In other words, the degree to which a sequence is “main-stream” can be controlled. Specifically, sequences that are stylistically on the fringe can be produced infrequently.

In our model, each production is given a *weight*, with the intention that when an auxiliary symbol is selected for replacement, the choice of production is among all productions with that symbol on the left-hand side, with productions having heavier weights being used more frequently. These weights are like probabilities, except that they don’t have to sum to 1. For the user’s convenience, the program normalizes the weights by dividing each by the sum of the given weights for that particular left-hand side to get an actual probability that is used to parameterize random selection.

As an example, if we wished to generate sequences that were only occasionally syncopated, we would designate lower weights for productions that tend to produce syncopated rhythms.

One further augmentation of the basic grammar idea is used to control the length of the sequence generated: certain auxiliary symbols are accompanied by an *argument* giving the length of the sequence to be generated. This argument may be used arithmetically in recursive rules, as will be seen in the example shortly.

IX. Generating Melodic Sequences

Once we have generated a rhythmic skeleton, we can provide a tone for each note. In our framework, which mainly concentrates on jazz at this point in time, we have a classification of tones according to the operative chord:

1) **Chord tones**: tones of the current chord.

2) **Color tones**: complementary tones for the current chord. These are tones that are not in the chord, but which are individually sonorous with it. They are also often called "tensions" in JMP.
3) **Approach tones**: non-chord tones that make good transitions to chord-tones or color tones.
4) **Other**: tones that do not fall into one of the categories above.

These characterizations are similar to ones used by jazz players. The corresponding notes, in the context of chords, provide the tension-release aspect common to professional jazz solos. We use a distinct symbol for each of the first three types of tones, and four additional terminals, as follows:

1) **C** a chord tone.
2) **L** a color tone.
3) **A** an approach tone.
4) **H** a "helpful" tone, defined to be any of one of the above.
5) **S** a scale tone, a member of a scale designated as being compatible with the chord.
6) **X** an arbitrary tone.
7) **R** a rest.

Approach tones currently force the following tone to be a chord tone, even if its designation is otherwise. The definitions of chord, color, approach, and scale tones is given on a per-chord basis in a separate vocabulary specification. All vocabulary information is user specifiable as a text file of S-expressions [26]. The later provide a readable way to communicate structure, in comparison to alternatives such as XML or proprietary formats.

For example, the grammar might generate a sequence (A8 L8 S8 C8 H4. S8). Based on the current chord, the symbols would be replaced with actual pitches, to become a note sequence, such as (f#+8 g+8 e+8 d+8 b4. g8), representing the notes shown in the first staff of Fig. 2 (Note: All staves in this paper are treble clef.). Here 8 refers to eighth-note duration, 4. means a dotted quarter-note duration, and + means to raise the pitch one octave from its default position, the octave just above middle C. Although the second note class in the sequence is specified as a color tone, this is actually over-ridden to be a chord tone, due to the previous note class being an approach tone. (This is the current operative convention, but it could be changed.) The following section elaborates on the overall process.

X. Additional Constraints

We allow the user to apply additional constraints to increase the likelihood that the pitch sequence is melodic. Although this information could be provided with an extension of the current grammar notation, this is left for future exploration. The specific controllable constraints are as follows:

1) Minimum and maximum pitch values.
2) Minimum and maximum interval between pitches.
3) A probability of using an interval outside the above specification, i.e. for a "leap".

Other possible constraints will be considered in the future.

XI. A Sample Grammar

Here is an example of a grammar that seems to be very workable for contemporary jazz licks. It is the current default in our tool. As mentioned earlier, an added advantage of using a grammar is that the user can change it to suit, for example, to make the generated melodies simpler or more complicated. The terminals in the grammar consist of the note class symbols followed by a duration value. These representations are chosen to be natural for the musician, who can also use the same notation in other parts of the software tool, such as in accompaniment style specifications. The duration values are designated as follows:

- **1** a whole-note
- **2** a half-note
- **4** a quarter-note
- **4.** a dotted quarter-note
- **8** an eighth-note
- **16** a sixteenth-note
- **4/3** a quarter-note triplet
- **8/3** an eighth-note triplet
- **16/3** a sixteenth-note triplet

For example A8/3 designates an approach tone with a duration of a eighth note triplet. Note that we did not include dotted eighth-notes and sixteenth-notes, as they don't occur frequently enough in jazz. A pair of eighth notes, however, is usually "swung", that is articulated so that the note on the beat is about 2/3 of a beat long, and the following note about 1/3 of a beat long. This swing ratio is implied from the ambient style, rather than being shown in the grammar explicitly. Hence the same grammar works well for both swing tunes and for latin tunes, where the eighth notes are more nearly equal. If ordinary dotting is desired, a dot after a terminal symbol denotes lengthening the normal value by 1/2.

The terminal set of the example grammar, where each item separated by commas counts as a single symbol, is:

X2, X4, X4., X8, X16, X4/3, X8/3,
H2, H4, H4., H8, H16, H4/3, H8/3,
A4, A8, A16, A8/3, C2, C4, C8,
L4, L8, R4, R8, S4, S8

Auxiliary symbos, such as P below, can be endowed with parameters, to permit generation of melodies to fit a certain number of beats. A parameter of N beats on the left-hand side of a production is sub-divided using arithmetic on the right-hand side to enable a coarse skeletal layout, which is further refined with other productions. The first group of productions below are parameterized. With $P(N)$ as the start symbol, where N is an argument indicating the length of the overall sequence in beats, the productions, with their assigned weights in square brackets, are (where the Q's are auxiliary symbols that expand into a sequence occupying the indicated number of beats):

1) $P(0) \rightarrow \mathit{empty}$ [1]

2) $P(1) \rightarrow Q1$ [1]
3) $P(2) \rightarrow Q2$ [1]
4) $P(3) \rightarrow Q2 \quad Q1$ [1]
5) $P(n) \rightarrow Q2 \quad P(n-2)$ [.25]
6) $P(n) \rightarrow Q4 \quad P(n-4)$ [.75]

An example initial derivation sequence might be

$$P(8) \rightarrow Q4\,P(4) \rightarrow Q4\,Q2\,P(2) \rightarrow Q4\,Q2\,Q2$$

This sequence of auxiliary symbols would then be refined, using productions chosen probabilistically from the following set, where the probabilities have been derived empirically by adjusting them based on generated melodies. Such adjustments may also be done by the end user for customization purposes. The remaining productions follow:

1) $Q4 \rightarrow Q2 \quad V4 \quad V4$ [0.52]
2) $Q4 \rightarrow V8 \quad N4 \quad N4 \quad N4 \quad V8$ [0.01]
3) $Q4 \rightarrow V4 \quad Q2 \quad V4$ [0.47]
4) $Q2 \rightarrow N2$ [0.06]
5) $Q2 \rightarrow V4 \quad V4$ [0.6]
6) $Q2 \rightarrow V8 \quad N4 \quad V8$ [0.12]
7) $Q2 \rightarrow H4. \quad N8$ [0.16]
8) $Q2 \rightarrow H4/3 \quad H4/3 \quad H4/3$ [0.06]
9) $Q1 \rightarrow C4$ [1]
10) $V4 \rightarrow N4$ [0.22]
11) $V4 \rightarrow V8 \quad V8$ [0.72]
12) $V4 \rightarrow H8/3 \quad H8/3 \quad H8/3$ [0.05]
13) $V4 \rightarrow H8/3 \quad H8/3 \quad A8/3$ [0.01]
14) $V8 \rightarrow N8$ [0.99]
15) $V8 \rightarrow H16 \quad A16$ [0.01]
16) $N2 \rightarrow C2$ [1]
17) $N4 \rightarrow C4$ [0.5]
18) $N4 \rightarrow L4$ [0.2]
19) $N4 \rightarrow S4$ [0.5]
20) $N4 \rightarrow A4$ [0.01]
21) $N4 \rightarrow R4$ [0.25]
22) $N8 \rightarrow C8$ [0.4]
23) $N8 \rightarrow L8$ [0.2]
24) $N8 \rightarrow S8$ [0.4]
25) $N8 \rightarrow A8$ [0.01]
26) $N8 \rightarrow R8$ [0.1]

XII. Sample Melodic Sequences

In this section, we demonstrate some of the sequences that are generated by the preceding grammar over selected chords. We show both the terminal sequence of symbols as well as melodic sequences. Note that for a given terminal sequence, there will generally be many distinct melodic sequences. The user can explore the latter while leaving the terminal sequence intact if desired.

The first example is over a single major chord, a C major 9. A terminal sequence generated by the grammar is:

$$A8 \quad L8 \quad S8 \quad C8 \quad H4. \quad S8$$

which designates a sequence of tones as

$$approach, color, scale, helpful, scale$$

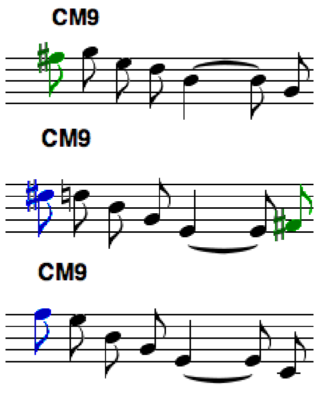


Fig. 2. Licks over CM9 in the pattern (A8 L8 S8 C8 H4. S8)

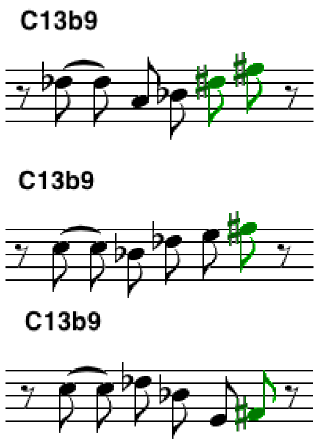


Fig. 3. Licks over C13b9 in the pattern (R8 S4 C8 S8 S8 L8 R8)

Fig. 2 shows three example generated licks over this chord. Our textual chord notation, also used in other aspects of our software for ease in user interaction, uses *M* to abbreviate *major*, *m* to abbreviate *minor*, and the symbols *b* and # to represent musical flats and sharps, respectively.

The next example is also over a single dominant chord, a C13b9. The generated terminal sequence is:

$$R8 \quad S4 \quad C8 \quad S8 \quad S8 \quad L8 \quad R8$$

However, the licks generated are quite different from the previous due to the chord and scale tones being interpreted over different types of chords. Fig. 3 shows three generated licks. Here b9 (D flat) is a chord tone, #9 (D sharp) has been designated as a scale tone, and #11 (F sharp) is a color tone. The melodic notation for the third lick would be

$$r8 \quad c+4 \quad db+8 \quad bb8 \quad e8 \quad f\#8 \quad r8$$

in our tool. The stave display and play-back can be generated automatically from this notation.

Now let's try a two-chord sequence typical of modern jazz: a D minor 9 followed by a Db9#11, the latter representing a "tritone substitution" in JMP. The generated

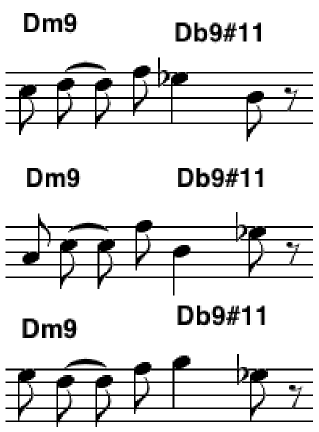


Fig. 4. Licks over Dm9 Db9#11 in the pattern (C8 C4 S8 C8 S8 R8)

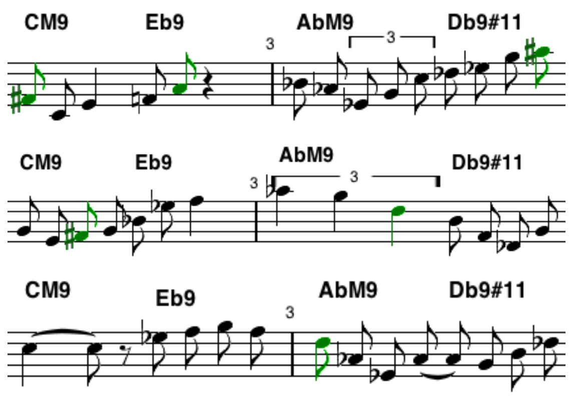


Fig. 5. Licks over CM9 Eb9 AbM9 Db9#11

terminal sequence is:

$$C8 \quad C4 \quad S8 \quad C4 \quad S8 \quad R8$$

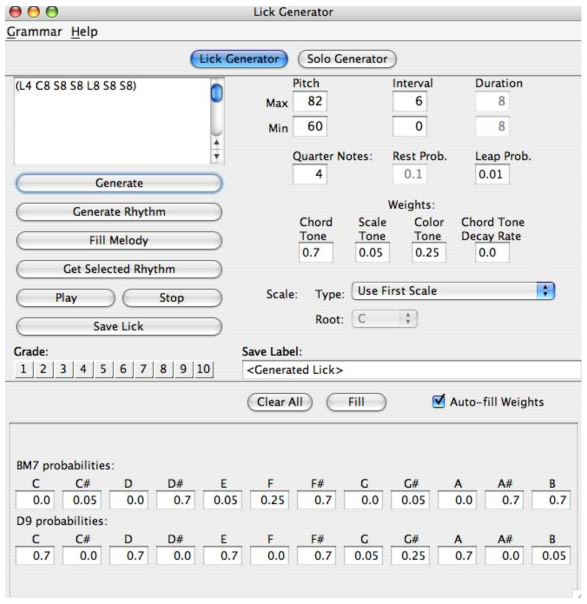
Fig. 4 shows three generated licks corresponding to this terminal sequence.

To show the capabilities of our approach in connecting melodies, we present melodies generated for a four-chord sequence, the famous "Tadd Dameron turnaround", giving three different generated sequences, with respective melodic possibilities in Fig. 5.

Finally we show how an entire chorus can be generated at one time by a single grammatic derivation, in this case for the chord changes to John Coltrane's Giant Steps, is shown in Fig. 7. Solos of this type take about one second to generate.

XIII. Lick Generator Control Panel

Fig. 6 shows the lick generator control panel in Impro-Visor. The terminal string generated by the grammar appears in the upper left text area. It may be used to generate an arbitrary number of melody sequences by replacing the terminals with notes, or the user can press a button to regenerate the terminal sequence. Although it is possible for the user to customize many of the parameters, and even adjust the grammar output to suit, in typical operation, this panel is set automatically based on



(L4 C8 S8 S8 L8 S8 S8)

	Pitch	Interval	Duration
Max	82	6	8
Min	60	0	8

Quarter Notes: 4 Rest Prob. 0.1 Leap Prob. 0.01

Weights:

Chord Tone	Scale Tone	Color Tone	Chord Tone Decay Rate
0.7	0.05	0.25	0.0

Scale: Type: Use First Scale

Root: C

Grade: 1 2 3 4 5 6 7 8 9 10

Save Label: <Generated Lick>

[✓ Auto-fill Weights]

BM7 probabilities:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0.0	0.05	0.0	0.7	0.05	0.25	0.7	0.0	0.05	0.0	0.7	0.7

D9 probabilities:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0.7	0.0	0.7	0.0	0.7	0.0	0.7	0.05	0.25	0.7	0.0	0.05

Fig. 6. Lick Generator Control Panel

chord and scale parameters that are part of the standard vocabulary specification. The casual user simply presses a button in the main window to generate melodies, without necessitating opening this panel. Other buttons in the panel allow us to assign a grade to licks and save them. The corpus of saved licks will be used to train a computational filter, such as a neural network or support vector machine, so that only licks satisfying subjective standards of a user will pass. We intend to incorporate such filtering into a future version of the tool.

XIV. Implementation

Impro-Visor is implemented entirely in Java, version 1.5. S-expression input and output for vocabulary elements is handled using the Polya library [27] which provides Lisp-like functionality [26] in Java. Some of the musical representation aspects were constructed using a modified version of jMusic [28]. GUI development was done using NetBeans [29].

XV. Evaluation

Currently evaluation is mostly subjective. As a moonlighting instructor of jazz improvisation, the first author assesses that the grammar-based approach generates melodies that compare favorably with those played by college-level jazz students of at least an intermediate playing level, if not better. The reader is invited to listen to examples played automatically on the website [1], which include both human-composed and machine-composed improvisations, and is encouraged to download Impro-Visor and try it for him/herself. Because the grammar is user-modifiable, there is plenty of opportunity to try to modify the grammar to generate melodies to individual taste.

XVI. Future Work

We plan to explore further other types of terminals in the grammar, such as one that specify particular forms of scalar sequences, including, but not limited to, chromatic approach sequences and other common jazz idioms. In addition to the generation of melodies, Impro-Visor generates accompaniment in the form of percussion, bass line, and chordal comping. All of these are specified in a style language similar to the one used to express melodic sequences. We intend to describe the algorithms for generating accompaniment from style specifications in a future paper.

Work is also in progress to construct critic filters for licks based on machine learning. This would be done by attaching grades to a large number of licks generated by the lick generator, which are then saved as a training corpus. Once trained, the filter can be imposed to improve even further the quality of generated licks, rejecting ones that are below a specified level and regenerating in such a case. Another possible line of work envisioned is to use machine learning to learn grammars from a corpus, including possibly separate grammars for different player styles.

XVII. Conclusion

We have described the use of probabilistic context-free grammars for generating convincing jazz sequences. Although the use of grammars has been suggested be many others, we believe that the key idea that makes our approach work well is the interpretation of the terminal symbols as chord tones, color tones, approach tones, etc. This provides the linkage between pitch classes and chords without requiring the grammar to be too specific as to pitches.

The original intended use of this approach is in an instructive tool for suggesting melodic ideas to improvisers. An advantage of the grammar approach in this context is that the user of a software tool based on grammar can change the grammar to suit. For example, a simple grammar can be used for beginners while a more sophisticated grammar can be used for advanced players and ones who are more technically accomplished. Although not the original intent, our approach could also be the basis of a real-time improvising companion, since in its current state, our tool will generate entire choruses in one second or less, but even with slower processing, the processing could, in principle, be overlapped with playback in a separate thread.

Acknowledgment

This work was supported by a Faculty Enhancement grant from the Mellon Foundation and by the National Science Foundation REU Program under grant Award No. 0451293 to Harvey Mudd College. We wish to thank Stephen Jones, Martin Hunt, and Steve Gomez for contributing to other aspects of our project that have help amplify the usefulness of our work.

References

[1] *Impro-Visor*, http://www.cs.hmc.edu/˜keller/jazz/improvisor.
[2] Robert Keller, Stephen Jones, David Morrison, Belinda Thom, and Aaron Wolin, "A computational framework enhancing jazz creativity", *CDROM Proceedings, Third Workshop on Computational Creativity, ECAI 2006*, Riva del Garda, Italy.
[3] Terry Winograd, "Linguistics and computer analysis of tonal harmony", *Journal of Music Theory*, volume 12, pp. 2-49, 1968.
[4] B. Lindblom and J. Sundberg, "Towards a generative theory of melody", *Swedish Journal of Musicology*, volume 52, pp. 77-88, 1970.
[5] H.C. Longuet-Higgins, "The Grammar of Music", *Interdisciplinary Science Reviews*, volume 3, no. 2, pp 128-156, 1978 (reprinted in Longuet-Higgins, *Mental Processes*, MIT Press, 1987).
[6] Curtis Roads, "Grammars as Representations for Music", *Computer Music Journal*, volume 3, no. 1, 1979, pp. 48-55.
[7] Gary M. Rader, "A Method for Composing Simple Traditional Music by Computer", *Comm. ACM*, volume 17, no. 11, pp. 631-638.
[8] David W. Hughes, "Grammars of Non-Western Musics: A Selective Survey", in Peter Howell, et al., *ed.*, *Representing Musical Structure*, pp 327-362, Academic Press, 1991.
[9] Bernard Bel and Jim Kippen, "Modelling music with grammars: formal language representation in the Bol Processor", in Alan Marsden and Anthony Pople *eds.*, *Computer Representations and Models in Music*, pp 207-238, Academic Press, 1992.
[10] Curtis Roads and John Strawn, *eds.*, *Foundations of Computer Music*, MIT Press, 1985.
[11] David Cope, *Computers and Musical Style"*, A-R Editions, Inc., Madison, WI, 1991.
[12] Jon McCormack, "Grammar-Based Music Composition", *Complexity International*, volume 3, (http://www.complexity.org.au/ci/vol03/mccorm/mccorm.html), 1996.
[13] Grard Assayag and Shlomo Dubnov, "Using Factor Oracles for Machine Improvisation", *Soft Computing*, volume 8, no. 9, pp 604-610, 2004.
[14] F. Lerdahl amd R. Jackendof, *A Generative Theory of Tonal Music*, MIT Press, Cambridge, Mass., 1983.
[15] Damon Horowitz., "Representing Musical Knowledge in a Jazz Improvisation System", In *Proceedings of Artificial Intelligence and Music, IJCAI workshop*, pp 16-23, August, 1995.
[16] M.J. Steedman,"A generative grammar for jazz chord sequences", *Music Perception*, voume 2, no. 1, pp. 52–77, 1984.
[17] P.N. Johnson-Laird, "How Jazz Musicians Improvise", *Music Perception*, volume 19, no. 3, pp 415-442, 2002.
[18] P.N. Johnson-Laird, "Jazz Improvisation: A Theory at the Computational Level", in Peter Howell, et al., *eds.*, *Representing Musical Structure*, Academic Press, 1991.
[19] David Temperley, *Music and Probability*, MIT Press, Cambridge, Mass., 2007.
[20] John Wade Ulrich, "The analysis and synthesis of jazz by computer", *Proceedings 5th IJCAI*, pp 865-872, 1977.
[21] Maarten Grachten, "JIG: Jazz Improvisation Generator", *Workshop on Current Research Directions in Computer Music*, pp 1-6, Audiovisual Institute-UPF, 2001.
[22] J. A. Biles, "GenJam: a genetic algorithm for generating jazz solos", *Proceedings of the 1994 International Computer Music Conference*, Aarhus, Denmark, pp. 131–137.
[23] George Papadopoulos and Geraint Wiggins, "A genetic algorithm for the generation of jazz melodies", *Proceedings of STeP 98*, Jyvskyl, Finland, 1998.
[24] Belinda Thom, "BoB: an interactive improvisational music companion", *Proceedings of the Fourth International Conference on Autonomous Agents*, ACM Press, Barcelona, Catalonia, Spain, pp. 309–316, 2000.
[25] PG Music, *Band in a Box*, http://www.band-in-a-box.com.
[26] John McCarthy, "Recursive functions of symbolic expressions and their computation by machine", *Comm, ACM*, volume 3, no. 1, pp 184-195, 1960.
[27] Robert Keller, Polya Java library, http://www.cs.hmc.edu/keller/polya/.
[28] Andrew Sorensen and Andrew Brown, jMusic Java library, http://jmusic.ci.qut.edu.au/.
[29] NetBeans, http://www.netbeans.org/.

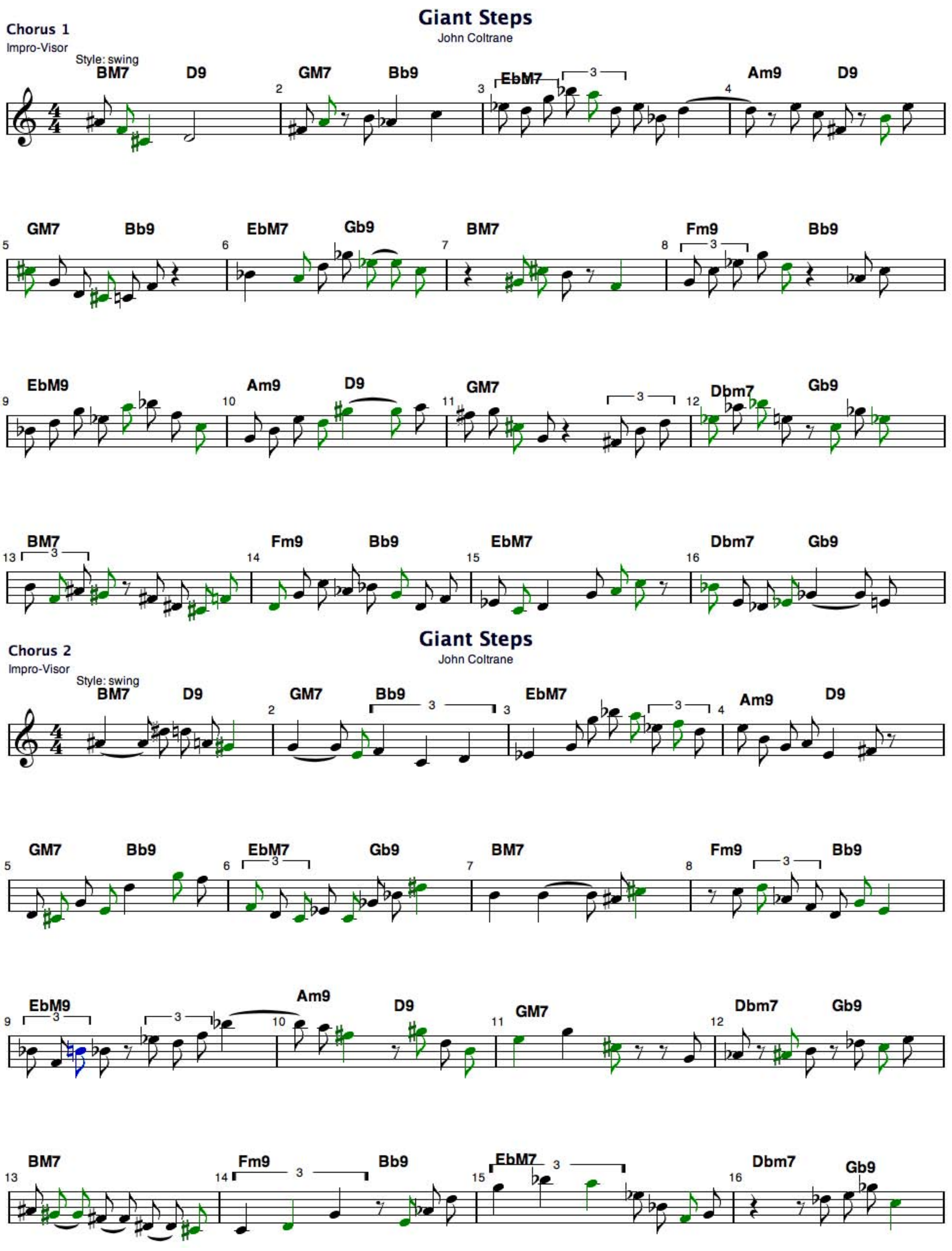

Chorus 1
Impro-Visor
Style: swing
Giant Steps
John Coltrane
BM7 D9 GM7 Bb9 EbM7 Am9 D9
GM7 Bb9 EbM7 Gb9 BM7 Fm9 Bb9
EbM9 Am9 D9 GM7 Dbm7 Gb9
BM7 Fm9 Bb9 EbM7 Dbm7 Gb9
Chorus 2
Impro-Visor
Style: swing
Giant Steps
John Coltrane
BM7 D9 GM7 Bb9 EbM7 Am9 D9
GM7 Bb9 EbM7 Gb9 BM7 Fm9 Bb9
EbM9 Am9 D9 GM7 Dbm7 Gb9
BM7 Fm9 Bb9 EbM7 Dbm7 Gb9

Fig. 7. Typical choruses generated on John Coltrane's Giant Steps