Machine Learning Techniques for Real-time Improvisational Solo Trading

Belinda Thom
School of Computer Science, Carnegie Mellon University
email: bthom@cs.cmu.edu

Abstract

This paper introduces a new melody generation scheme that enables customized interaction between a live, improvising musician and the computer. This scheme provides a method for integrating a model that was learned to describe the user's tonal, melodic-interval and -contour trends into a stochastic process that, when sampled, produces sequences that exhibit similar trends while also seamlessly integrating with the local environment. This algorithm's musical performance is evaluated both quantitatively, using traditional machine learning techniques, and qualitatively, exploring its behavior in the context of Bebop saxophonist Charlie Parker.

1 Introduction

The vision of my research is to interactively improvise with the computer, using it as a tool when practicing at home alone in order to capture and experiment with my all-too-transient spontaneous musical ideas. I want the computer to serve as my own personal *improvisational music companion* (IMC), playing music with me, improvising with me, and getting to know my own unique musical personality in such a way that it could:

> ... answer me halfway through my phrase, bringing me to the point where I can actually sing what it's going to play next, and then, instead of playing that, it'll play something against it which compliments what I'm singing in my head.

This quote, which inspires my research, paraphrases drummer Keith Copeland's musings about improvising with jazz pianist Ahmad Jamal (as compiled in (Berliner 1994)). Towards this end, I am building *Band-OUT-of-a-Box* (BoB), an agent that trades solos with its user in a musician- and context-specific manner.

1.1 Background

Most interactive music systems rely on an *author-the-aesthetic* paradigm, meaning that ultimately, the artist, programmer, and/or user is expected to configure the system's inner workings, hand-customizing until an interesting musical aesthetic results, e.g., (Dannenberg 1993; Rowe 1993; Pennycook and Stammen 1993). For example, in many of Dannenberg's interactive performances, the computer is programmed to be an extension of his composition, capable of responding to the real-time performance decisions and nuances of the human performer. In this setting, music recognition is either hand-coded and/or trained using supervised learning data, e.g., (Dannenberg, Thom, and Watson 1997), and music generation is generally designed to carry out the composer's, rather than the performer's, goals. While more open-ended improvisational experiences are sought in Wessel's computer-assisted performance settings, again the primary focus is human authored aesthetics. Towards this end, much of their work revolves around organizing and controlling the access of musical material so that human computer musicians can author meaningful musical experiences on-the-fly (Wessel, Wright, and Kahn 1998; Wessel and Wright 2000). When interactive systems do embody a more autonomous sense of musical aesthetic, e.g., (Biles 1998; Franklin 2001), their underlying representations tend to be less general and flexible, for example relying on a specific set of musical scales, hand-tuned fitness functions, and the like.

1.2 Improvised Melodic Companions

In contrast, improvised companionship in the soloing, practice-tool context motivates a different research agenda (Thom 2001; Thom 2000a). Machine learning is used to replace explicit authoring, and this technology is used to configure BoB's computational improvisation model to a particular musician playing at a particular time and in a particular context.

In this setting, we assume that it is more important that user training data be recent — e.g., capturing the musician's playing style right now — than it is to be large. Over-fitting is prevented by simplifying the underlying mathematical model, the assumption being that a less accurate model trained on more relevant data will provide a more intimate response than a more complex model that is trained on a larger hodge-podge of the musician's behavior would.

We also assume that it is more important to handle a wide variety of improvisational settings reasonably well than it is to expertly handle a specific improvisational genre. For example, since I want to be able to use BoB when improvising on top of:

- the rhythmic, modal types of playing one finds in Ali Farque Toure's (guitar) or Dr.Dig's (didjeridoo) music;
- well delineated chord progressions (blues, bluegrass, funk, etc.);
- unaccompanied settings, where melodies are completely unconstrained;

a specialized, knowledge-based approach is avoided. Rather than concentrating on capturing specific improvisational expertise, BoB's main emphasis is providing an extremely flexible and adaptive model of improvised melody that can:

- Leverage off of the spontaneous creativity of its user;
- Transform what the user plays in interesting new ways;
- Reflect the user's performance back at them, that such comingling might result in making the user's spontaneity less transient, more tangible and explorable.

Finally, we assume that training data collection must be natural and non-obtrusive, easily accommodating amateur improvisers with varying skill levels.

1.3 Prior Work

BoB's interactive solo-trading model is comprised of two parts, the *Listener* and the *Generator.* The Listener, which has been described elsewhere (Thom 2001; Thom 2000b; Thom 1999), involves two inter-related technologies:

- **Representation:** Encoding solos;
- **Learning:** Automatically configuring the representation to a specific user.

In BoB, the user is merely asked to *warm-up*, first authoring a few high-level controls (tempo and accompaniment harmony), and then improvising on top of a fixed computer accompaniment for about 5 to 15 minutes.[1] With no further user interaction, this warm-up data is fed off-line into an unsupervised learning algorithm, producing a *mixture of Multinomials* (vMn) model that can abstractly map local chunks of a musician's solo into a discrete set of user-specific *playing modes*. On-line interaction requires one more high-level control to be authored — who solos where and when — so that BoB knows how to schedule its listening to the musician's solo *call,* and its search and playback of an appropriate solo *response*.

[1]These same controls are required by *Band-in-a-Box* (PGMusic 1999), the commercial, non-interactive harmonic accompaniment program that many improvisers use to practice on top of.

When solo-trading, the Listener transforms the musician's call into various abstractions, the highest level, via vMn, being user-specific. When replying, the Generator starts with this user-specific abstraction, transforming it into a new user- and context-specific solo. This scheme offers a viable compromise between the conflict that arises when the user, who needs some degree of artistic control, and BoB, who needs an operational method for responding autonomously, interact. The foundation of this compromise relies on synthesizing two types of information:

1. That which the musician directly controls via the *local environment,* i.e., their call;
2. That which they indirectly affect, via the model inferred from their warm-up data.

1.4 Paper Outline

This paper describes a very basic — i.e., primitive — technology for generating short segments of notes using a self-configuring, distributed mechanism that, when sampled, produces a solo that:

1. Accomplishes a particular playing mode goal;
2. Integrates seamlessly with additional constraints imposed by the locally evolving musical environment.

The highly constrained, stochastic generation procedure that is developed here is motivated in part by the philosophy of Johnson-Laird (1991). I believe this procedure's greatest strength is that it provides the computational elements needed to emulate a key aspect of improvisation's essence:

> The ability to improvise [...] seems to come out of the end of one's fingertips [...] its main components are profoundly unconscious. (Sudnow 1978)

This paper begins with a review of the Listener's solo representation and learning architecture, followed by the Generator's technology, which closes the loop between learning and generation, in some sense inverting the learned vMn mapping so that new goal-driven solo responses can be found by the computer in real-time. Finally, the Generator's algorithm is evaluated both numerically and musically.

2 Listener

The Listener's technology is outlined in order to situate the generator-based discussion that follows.

2.1 Representation

In Figure 1, a four bar musician's call is shown (top stave), along with parts of the Listener's results (bottom three rows). BoB encodes solos on a per-bar basis, transforming each bar into a *tree,* v, who's internal structure approximates the bar's rhythmic aspects and whose pitch-based content is approximated by its leaves. A bar's *pitch sequence* (PS) is defined to be the in-order walk of its pitch-valued leaves,[2] mathematically notated via:

$$p = \langle p_1, p_2, \cdots, p_t, \cdots, p_{sz} \rangle.$$

Subscript '$_t$' indicates a leaf's temporal location among the sz pitch-valued leaves in v. Pitch leaves contain *pitch class, octave,* and *slur* information. By default, identical adjacent pitch leaves are tied to form longer notes, allowing v to encode syncopated rhythms.

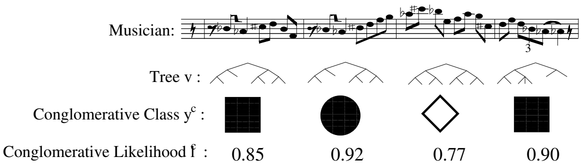

Figure 1: The Listener's Perception of a Musician's Solo

Early on, I chose to focus on automating a procedure for handling PS content in a user-specific manner because v's explicit hierarchical rhythmic handling naturally lent itself to musically sensible transformations — simply *embellish* (i.e., grow) or *simplify* (i.e., collapse) certain leaves in the tree. While musically sensible rhythmic transformations to a musician's call were easily obtained in this way, the pitch values that were assigned to the new leaves that resulted often sounded musically inferior. As such, my current efforts have sought to adequately encode and transform PS content.

In order to capture deeper melodic structure than is available on the bar's immediate surface, three types of histogram *views* for a tree's PS are constructed (these histograms are not shown in Figure 1):

1. **Pitch Class (PC):** histogram h^p summarizes the frequency with which particular pitch classes occur in p.

2. **Intervallic Motion (INT):** histogram h^i summarizes the frequency of absolute pairwise intervals in p.

3. **Melodic Direction (DIR):** histogram h^d summarizes trends that capture how certain substrings of intervals in p ascend $(+)$, descend $(-)$, or stay the same (o, the unison interval).

Conglomerative histogram h^c is the entire trio of views:

$$h^c = \langle h^p, h^i, h^d \rangle,$$

summarizing a bar's *tonal* (PC), *melodic-continuity* (INT), and *melodic-contour* (DIR) trends. *Conglomerative* (superscript '^{c}') is defined so as to indicate that pitch class ('^{p}'), absolute intervals ('^{i}'), and intervallic directions ('^{d}') are treated independently, in isolation from one another.

Constructing the first two histograms are straight-forward. PC contains 12 bins, one for each pitch class, and INT contains 13 semitone intervals, ranging from the unison interval to all intervals greater than or equal to an octave. The directional histogram's bins are more complicated, each corresponding to one of the 27 transition arcs shown in the *directional Markov Chain* of Figure 2.

This chain has 9 states, one for each *relevant history* r:

$$r \in \{+, ++, +++, -, --, ---, o, oo, ooo\}, \quad (1)$$

where for p_t, r is defined to keep track of how many of the most recent previous directions, ranging from p_{t-3} up to p_t have the same sign $(+, -,$ or $o)$. Whatever the current state, the sign of the next interval, $p_{t+1} - p_t$, determines what arc is taken. Dotted arcs corresponding to the downwards sign, solid arcs to upwards, and dot-dashed arcs to unison. Since it would clutter up the graph to connect every transition to its appropriate input state, the following convention is used: unterminated solid arcs terminate in state $r = -$, solid arcs in $+$, and dot-dashed arcs in o. This chain's structure counts repeated signs of up to length three, resetting back to a relevant history of length one anytime a break in sign occurs.

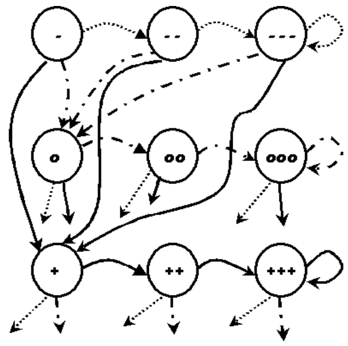

Figure 2: The Directional Markov Chain

While these histograms provide the benefit of mapping variable-sized pitch sequences into fixed-size histograms, this transformation is by definition ignorant of some of p's temporal aspects. In particular, a specific h^c does not tell us how to derive a sequence of pitches that exhibits exactly the specified number of PC, INT, and DIR counts. While trio h^c provides a fair amount of temporal information, including:

- **Zero-order:** PC entirely ignores p's ordering;

- **First-order:** INT captures the ordering between pairs in p;

- **Second-** through **Fourth-order:** DIR's bins consider subsequences in p whose lengths depend on context. Sequences are composed of between three to five items;

[2] Rests are ignored.

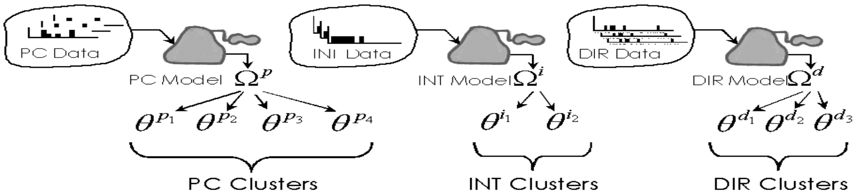

Figure 3: User-Specific Learning

this information does not perfectly memorize the sequence. On the one hand, not memorizing gives h^c its generalization power — p's temporal behavior is captured at several interrelated, more abstract levels, going beyond a mere "string-of-notes" based approach. On the other hand, the temporal information that conglomeration contains is non-trivially interrelated and incomplete, and this is what makes a conglomerative-based generation algorithm difficult.

2.2 Customizing Representation via Learning

Figure 3 shows an instance of user-specific learning. The data in this figure comes from the warm-up session, a trio of histograms built for each bar the musician played. The gray-handled machines represent three independent unsupervised learning algorithms, each estimating its own mixture of k Multinomial components (vMn), whose parameters include:

$$\Omega = \langle \theta^1, \theta^2, \cdots, \theta^c, \cdots, \theta^k \rangle.$$

Notation θ^k is the probability vector of the k-th Multinomial. This learning procedure is also conglomerative and similar superscript notations are again used. For example, $k^p = 4$ indicates that four different PC clusters were learned:

$$\Omega^p = \langle \theta^{p_1}, \theta^{p_2}, \theta^{p_3}, \theta^{p_4} \rangle,$$

each θ^{p_c} specifying a different probability distribution over pitch-class bins. Similarly, each θ^{i_c} specifies a different set of weights over intervallic bins, and each θ^{d_c} a different set of transition probabilities over the directional Markov Chain of Figure 2.

Each clustering function is inferred by assuming that a probabilistic mixture model with k Multinomial components generated the data. The *Expectation-Maximization* algorithm (Dempster, Laird, and Rubin 1977) is used to estimate vMn's parameters, doing so in such a way that the histogram data appears maximally likely. In particular, parameter estimation is MAP-based, a Dirichlet prior is placed on each histogram's bins (Thom 2000c) to combat over-fitting and ensure that all bins have some probability of occurring.

This learned model is used to customize representation h^c by mapping it into its most likely *conglomerative class*:

$$y^c = \langle y^p, y^i, y^d \rangle,$$

where this mapping is based on Bayes rule, individually maximizing each model's posterior, e.g.,

$$y^p = \texttt{Argmax}_c(\theta^{p_c} | h^p, \Omega^p).$$

The purpose of this mapping is to capture the different *playing modes* that the musician used to spontaneously construct their warmup improvisation. For example, one y^c might capture the user's preference for certain tones in some arbitrary scale, realized in the form of a specific set of ascending arpeggios.

Because this model is probabilistic, its estimated class likelihood:

$$l^c = \Pr(h^c | y^c, \Omega^c) = l^p \cdot l^i \cdot l^d,$$

provides a measure of how *non-surprising* (average) a bar's PC, INT, and DIR trends appear given the belief that playing mode y^c was used to generate the bar.

3 Generator

While powerful user-specific abstractions have been shown to emerge using the vMn learning method (Thom 1999; Thom 2001), alone this conglomerative mapping technology cannot be used to generate a new solo response. I now present the crucial technology, a mechanism for closing the learning/generation loop, integrating Ω^c into a stochastic process that, when sampled, produces sequences that tend to exhibit a specific set of PC, INT, and DIR trends. The set of trends that are to be realized is called *generative goal* y^c, and its corresponding Multinomial distributions are contained in *generative hint* θ^c, where

$$\theta^c = \langle \theta^p, \theta^i, \theta^d \rangle$$

and θ^p is the y^p-th component of Ω^p, θ^i is the y^i-th component of Ω^i, and θ^d is the y^d-th component of Ω^d.

A basic call-and-response scenario is shown in Figure 4. The musician has just played a four-bar call and BoB, having just determined and scheduled its 2nd bar of response, is now determining what to play for its 3rd response bar. While there is nothing in BoB's technology that requires such a simple interaction scheme (in future work, richer scenarios will be investigated), this scenario is extremely simple in that the

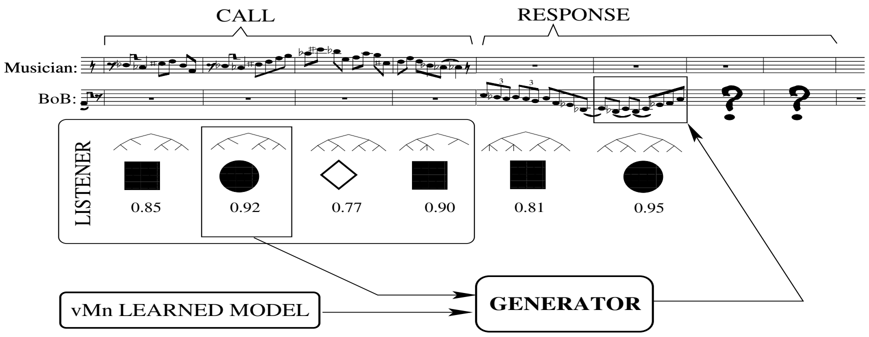

Figure 4: A Trading Fours Scenario

Listener's playing mode for call bar i directly guides the Generator's search for response bar i.[3]

Despite this scenario's simplicity, it does provide BoB an awareness of and an ability to respond to the contrast between tonalities, intervals, and melodic contours that the musician realized during their unfolding call — crucial functionality given that these contrasts are certainly part of what makes their solo interesting! The usefulness of this functionality relies on the following assumptions:

1. Knowledge contained in hint θ^c is sufficient for generating a new solo response that *accomplishes*, i.e., maps into, goal y^c.

2. Generating a new solo response that displays the same sequence of abstractions will produce an intimate user- and musician-specific connection, enabling BoB to realize a modicum of musical companionship.

$\boxed{\texttt{generateResponse}(v, y^c, l^c)}$
$\quad v' = v$`.tweak`();
$\quad sz' = v'$`.numPitchLeaves`();
$\quad \theta^c = \left\langle \theta^{p_{y_p}}, \theta^{i_{y_i}}, \theta^{d_{y_d}} \right\rangle$
$\quad p'$ = `empty`
$\quad$ `for` $i = 1 : 1 : num$
$\quad\quad tmp$ = `generatePS`(θ^c, sz')
$\quad\quad p'$ = `saveBestSoFar`(tmp, y^c, l^c)
$\quad$ `end`
$\quad v'$`.assignPitchLeaves`(p');
$\quad$ `return` v'

Table 1: The Per-Bar Generation Algorithm

3.1 Per-Bar Response Generator

BoB uses the algorithm outlined in Table 1 for generating each bar of solo response. This algorithm first transforms call rhythm v into response rhythm v', method `tweak` stochastically embellishing and/or simplifying various leaves in v. Next, a pitch sequence p' comprised of sz' pitch values is generated so as to exemplify playing mode y^c. The elements in p' are then assigned in-order to the leafs in v'. The `for` loop is needed because `generatePS` relies on random sampling, so we can never be sure that a single execution will produce a sequence that accomplishes goal y^c. The hope is that after num tries at least one of these solutions will achieve the goal, in which case function `generateResponse` is considered *successful* (otherwise it is considered a *failure*).

When `generateResponse` is successful, it makes sense to quantify how successful it was, which relates to how easy it was to meet goal y^c when taking a sz'-stepped walk. This value, the *success rate,* is reported in terms of what percentage of the num solutions classified as y^c. Function `saveBestSoFar` determines which of num solutions is best, where if several walks accomplish the same goal, the solution that is chosen is the one whose non-surprise is closest to the call's value, l^c.

[3]An exception to this rule is made when the response's underlying harmony differs from the call's, as happens when trading fours over a 12-bar blues progression. In this case, y^i and y^d are taken directly from the call, but y^p is taken from one of the musician's calls that corresponds to the current harmony.

3.2 Goal-Driven Pitch Sequence Generation

The remaining technical difficulty involves figuring out how to use what was learned (θ^c) in order to generate a new pitch sequence that exhibits the desired PC, INT, and DIR trends.

Part of the difficulty is that we cannot use h^c's generative model directly to obtain a new pitch sequence, for a new h^c does not tell us *how* to derive a particular pitch sequence. In addition, not all values of h^c that might be generated are guaranteed to correspond to a *viable* solution p. For example, our Lagrangian priors ensure that it is possible (albeit perhaps very unlikely) for hints θ^p and θ^i to generate the histograms in Figure 5. The problem with this simple example is obvious — there is no way to choose and order two pitch values so that one of them uses pitch class D♭ and the other uses E♭, while at the same time producing a Tritone or Perfect 4th interval between them.[4] Unfortunately, when h^c is not derived from

[4]The left over interval would connect to the most recent pitch value from

a pitch sequence in the first place, it is not necessarily viable. Although y^c provides a significant amount of abstraction, its generalization is too course. Another method is needed to generate actual pitch sequences.

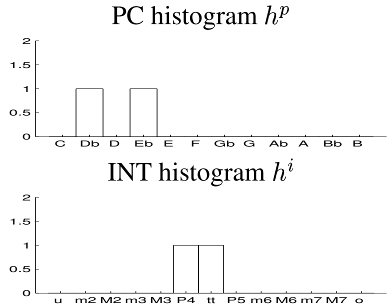

Figure 5: An Infeasible Example

Another non-trivial issue is that certain trends may interfere with others. For example, what is the best way to handle the following type of conflict: together, PC and DIR seem to prefer upwards, semitone motions, yet INT discourages small intervals? While it is easy to come up with situations where in the limit, i.e., as $sz \to \infty$, all three views average behaviors cannot be simultaneously attained, the fact that:

1. We are generating relatively short sequences;[5]
2. The procedure for learning Ω^c was biased so that all histogram bins have some probability of occurring;

means that in practice, solutions that accomplish goal y^c can usually be found.

In order to generate a new PS which achieves goal y^c, a Markov Chain whose states are comprised of two parts, absolute pitch a and relevant history r (Equation 1), is constructed. States are notated via $s \in \langle a, r \rangle$. This chain is defined over a specific range of pitches, $[a_{min}, a_{max}]$, with n_a pitches included in total.[6] The Markov assumption means that:

$$\forall t,\ \Pr(s_t | s_1, s_2, ...s_{t-1}) = \Pr(s_t | s_{t-1}).$$

This set of transition probabilities must be estimated to fully specify the chain. Since there are 9 relevant histories to consider, the Markov Chain contains a total of $9 \cdot n_a$ states. This chain is constructed so as to impose the same directional structure that was shown in Figure 2, except that now instead of three non-zero transition probabilities per state, there are at most n_a, i.e., the chain's computation cost is $\mathcal{O}(9 \cdot n_a^2)$.

BoB uses the following heuristic to integrate learned parameter θ^c into the Markov Chain's transition probabilities:

$$\Pr(s'|s) \propto \quad (2)$$
$$\Pr(\texttt{pc}(a')) \cdot \Pr(\texttt{abs}(a' - a)) \cdot \Pr(r \to \texttt{sign}(a' - a)).$$

the previous bar.

[5]Per-bar, reasonable values for sz' range between 0 to 32.

[6]Using the musician's call bar to set these ranges provides further coupling with the local environment.

Although this heuristic assumes that PC, INT, and DIR behavior can be independently handled, this is not true — views are inter-related. Fortunately, in practice, this heuristic provides adequate guidance, with few searches often producing the desired goal (e.g., $num = 25$). While Steinsaltz and Wessel () are investigating exact methods for estimating transition probabilities on chains that simultaneously converge to several different temporally ordered trends, the heuristic presented here is more appropriate because it meets BoB's real-time needs.

Eliminating the '$\propto$' in Equation 3 requires each state's output probabilities to be normalized. The terms in Equation 3 are easily retrieved from θ^c. Function `pc` returns the pitch class of state s', whose probability is specified in θ^p. Similarly, `abs` returns the absolute interval between the pitches in states s' and s, and `sign` returns its direction. These probabilities are obtained from θ^i and θ^d respectively.

Consider now Figure 6, an example constructed over the range $a \in \{\mathrm{C}, \mathrm{D}\flat, \mathrm{D}, \mathrm{E}\flat\, \mathrm{E}\}$. In this case, there are $n_a = 5$ pitches, or 45 states in total. However, only those states and arcs that are needed to describe the transitions out of states $s \in \langle \mathrm{D}, r \rangle$ are shown. Only those arcs are shown for which Equation 3 is defined are shown (all other transitions have zero probability). Dotted lines correspond to downwards motions, solid lines to upwards motions, and dot-dashed lines to unison intervals. To fit everything on the page, r's notation is simplified (e.g., $+^2$ corresponds to $++$). The arcs in Figure 6 correspond to one step through the chain. When taking a sz'-stepped walk, this graph must be repeated sz' times, e.g., Figure 7. Again, Figure 7 is simplified; only 3 pitches (A, B$\flat$, B) and histories ($+, -, o$) are shown.

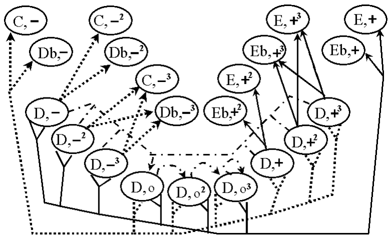

Figure 6: Part of Generative Markov Chain

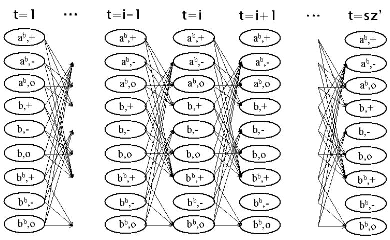

Figure 7: Example: a sz'-Stepped Walk

3.3 Clamping and Other Extensions

Figure 7 is a state transition diagram for which powerful extensions are possible. For example, at any time t we can *clamp* certain states, specifying which one from a particular subset in column t must (or must not) occur. All that is required is for this future knowledge to be propagated backwards through the graph (Thom 2001). The most expensive case is when the last column ($t = sz'$) is clamped, in which case the computation cost is $\mathcal{O}(9 \cdot n_a^2 \cdot sz')$.

Clamping opens up exciting possibilities for future work. For example, to what degree can harmonic knowledge improve BoB's performance in certain settings? This can be investigated by developing a language for clamping certain states to key chord tones. Similarly, we can control when various intervals and/or contour changes occur, providing powerful mechanisms for re-coupling rhythm and phrasing heuristics back into the pitch generation procedure.

Some important details concerning the generation algorithm were not mentioned in the previous section for space reasons. For example, the interval at time $t = 1$ always links back into the most recent pitch played in the previous bar, i.e., $prev$. Unison intervals are also disallowed in certain walk locations in order to ensure that when p' is reassigned to v' the tree makes sense. Different PC weightings are also used when sampling states that are on- versus off-the-beat. As detailed in Thom (2001), these extensions are handled by calculating several different transition matrices up-front. When walking through the graph, t's location is used to select which matrix should be used to weight that column of states' arcs.

3.4 Results

The first result presented is numerical, quantifying the degree to which Equation 3's heuristic helps BoB achieve its goals. Simulated datasets were generated using two different musician's improvisations, Bebop saxophonist Charlie Parker and Swing jazz violinist Stephane Grappelli. Simulation first involved learning a different vMn model for each musician's warm-up data (120 and 129 bars over a given tune respectively). Once these models were learned, each was used to perceive its own warm-up data. Each bar's abstract perception was then fed into its respective musician's Generator as goal y^c. In this way, a novel response was produced for each bar of each training set.

As shown in Figure 2, several different simulations were run (one per row). Each experiment was defined by several different options. Column 'Musician' indicates what musician's data was used as the call. When `generatePS` was guided by learned knowledge (θ^c), the value in the 'Learn' column is 'True'. In contrast, 'False' indicates that each arc was uniformly weighted, i.e., no customization was employed. Column 'Clamp' indicates whether or not the last pitch of each call bar was used to clamp the last pitch of each response.[7] The last three columns quantify how well `generateResponse` performed, 'Success Rate' reporting the average percentage obtained for each successful call's rate when normalized over $num = 150$.

Musician	Learn	Clamp	Num Successes	Num Fails	Success Rate
Parker	False	False	92	28	3.7%
Parker	True	False	120	0	40%
Parker	False	True	98	22	4.3%
Parker	True	True	120	0	47%
Grappelli	False	False	59	70	5.2%
Grappelli	True	False	122	7	45%
Grappelli	False	True	66	63	7.5%
Grappelli	True	True	125	4	56%

Table 2: Uniform versus Heuristic Simulations

Without learned knowledge, many more bars failed, the worst case being 63 out of 129 failed goals. Also, clamping improved uniform performance, eliminating 6 of the failures in Parker and 7 in Grappelli. This improvement is not surprising given that clamping more tightly couples generation with the user. When the heuristic was used, all of Parker's 120 calls were successfully responded to, and only a few of Grappelli's goals failed (clamping reduced the number of failures from 7 to 4).

Additional simulations are described in Thom (2001). For example, the simulated datasets described above were used to re-learn the vMn model. By comparing the differences between the learned model that drove generation and the model that was learned from the simulated data, `generateResponse` was demonstrated to successfully reproduce much of the original model's structure. Experiments that verified human listeners' abilities to recognize when solos were generated with different goals are also described in Thom (2001).

The remaining results are musically evaluated in order to demonstrate `generateResponse`'s flexibility and power in the IMC domain. Consider each of the examples BoB generated in Figure 8. Bar (a) is a call taken from Parker's warmup. In all of the response scenarios, (b)-(f), the call's rhythm was fed into `generateResponse`. In all cases, `generateResponse` was also constrained to clamp to the call's final pitch (middle C) and link into Parker's previously played pitch ($prev$ was one octave above middle C). The only aspects that varied between generations were whether or not rhythm was tweaked and which goal y^c was used. Rhythmic tweaking occurred in Bars (c) and (f), simplifying (the final quarter) and embellishing (the sextuplet) respectively. The goal that drove Bars (b) and (c) was quite different from the playing mode associated with Parker's call. In particular, $y^c = ii$ corresponds to a playing mode where the following are preferred: pitch classes D♭, D and E♭; the unison, Perfect 4th, and minor 6th intervals; and downwards motion

[7] Thus ensuring that each response bar has the same $prev$ that its respective call had.

Figure 8: Generative Examples

combined with no change in melodic contour. In contrast, the playing mode of Parker's call, $y^c = i$, prefers: all pitch classes but D♭, E, A♭, and C; small intervals, especially the Major 2nd, but not the unison; and runs of downwards motion interspersed by an upwards interval or two.

The first thing to note is that multiple responses (e.g., Bars (b)-(c) or Bars (d)-(f)) are only possible because `generateResponse` is stochastic, different random numbers producing different results. Also compare how different the bars driven by Goals i and ii are. For example, ii's responses are syncopated and jump around, whereas i's are continuous with no syncopation. In terms of pitch class, D♭ (Goal ii) and A (Goal i) are noticeably present and distinctive. It is also interesting that even though Goal ii's tonality did not prefer pitch class C, it was able to achieve its goal even though its last tone was clamped to this value. Fortunately, in all the experiments I have run, this type of cooperation between constraints and goals has been observed.

4 Conclusion

The ability to control a solo's tonality, melodic continuity, and melodic contour provides a powerful computational model for generating user- and context-specific solos. Furthermore, the ability to clamp aspects of a solo's content within this framework provides mechanisms for incorporating higher level domain knowledge, authoring control, etc. While attempting to simultaneously control these three different temporal views of a pitch sequence's behavior is difficult, each view contributes to the musical salience of the result. The heuristic algorithm presented here has been demonstrated to work well in practice, successfully integrating these views to produce coherent solo responses.

Acknowledgements This research was sponsored in part by the Computer Science Dept. at Carnegie Mellon University and by a DARPA and US Air Force contract under Cooperative Agreement No. F30602-98-2-0135. The views and conclusionsin this paper are those of the author and should not be interpreted as necessarily representing the views of the sponsors.

References

Berliner, P. F. (1994). *Thinking in Jazz, The Infinite Art of Improvisation*. University of Chicago Press.

Biles, J. (1998). Interactive GenJam: Integrating real-time performance with a genetic algorithm. In *Proceedings of the 1998 ICMC*. International Computer Music Association.

Dannenberg, R. B. (1993). Software design for interactive multimedia performance. *Interface - Journal of New Music Research 22*(3), 213–218.

Dannenberg, R. B., B. Thom, and D. Watson (1997). A machine learning approach to musical style recognition. In *Proceedings of the 1997 ICMC*. International Computer Music Association.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*.

Franklin, J. A. (2001). Multi-phase learning for jazz improvisation and interaction. In *Proceedings of the Eighth Biennial Symposium for Arts & Technology*.

Johnson-Laird, P. N. (1991). Jazz improvisation: A theory at the computational level. In P. Howell (Ed.), *Representing Musical Structure*, pp. 291–325. Academic Press, Inc.

Pennycook, B. and D. Stammen (1993). Real-time recognition of melodic fragments using the dynamic timewarp algorithm. In *Proceedings of the 1993 ICMC*. International Computer Music Association.

PGMusic (1999). Band-in-a-box. http://www.pgmusic.com.

Rowe, R. (1993). *Interactive Music Systems : Machine Listening & Composing*. MIT Press.

Steinsaltz, D. and D. Wessel. The markov melody engine: Generating random melodies with two-step markov chains. Technical report, Deptartment of Statistics, University of California at Berkeley. In Progress.

Sudnow, D. (1978). *Ways of the Hand, the Organization of Improvised Conduct*. Harvard University Press.

Thom, B. (1999). Learning melodic models for interactive melodic improvisation. In *Proceedings of the 1999 ICMC*. International Computer Music Association.

Thom, B. (2000a). Artificial intelligence and real-time interactive improvisation. In *Proceedings from the AAAI-2000 Music and AI Workshop*. AAAI Press.

Thom, B. (2000b). BoB: an interactive improvisational music companion. In *Proceedings of the Fourth International Conference on Autonomous Agents*.

Thom, B. (2000c). Unsupervised learning and interactive jazz/blues improvisation. In *Proceedings of the AAAI-2000*. AAAI Press.

Thom, B. (Expected date, September 2001). *BoB: An Improvisational Music Companion*. Ph. D. thesis, Computer Science Department, CMU.

Wessel, D. and M. Wright (2000). Problems and prospects for intimate musical control of computers. In *CHI '01 Workshop New Interfaces for Musical Expression*. ACM SIGCHI.

Wessel, D., M. Wright, and S. A. Kahn (1998). Preparation for improvised performance in collaboration with a Khyal singer. In *Proceedings of the 1998 ICMC*. International Computer Music Association.