

Control Improvisation with Application to Music

Alexandre Donzé¹, Sophie Libkind³, Sanjit A. Seshia¹, and David Wessel²

¹ EECS Department, UC Berkeley

² Music Department and CNMAT, UC Berkeley

³ Swarthmore College

Abstract. We introduce the concept of control improvisation, the process of generating a random sequence of control events guided by a reference sequence and satisfying a given specification. We propose a formal definition of the control improvisation problem and an empirical solution applied to the domain of music. More specifically, we consider the scenario of generating a monophonic Jazz melody (solo) on a given song harmonization. The music is encoded symbolically, with the improviser generating a sequence of note symbols comprising pairs of pitches (frequencies) and discrete durations. Our approach can be decomposed roughly into two phases: a generalization phase, that learns from a training sequence (e.g., obtained from a human improviser) an automaton generating similar sequences, and a supervision phase that enforces a specification on the generated sequence, imposing constraints on the music in both the pitch and rhythmic domains. The supervision uses a measure adapted from Normalized Compression Distances (NCD) to estimate the divergence between generated melodies and the training melody and employs strategies to bound this divergence. An empirical evaluation is presented on a sample set of Jazz music.

1 Introduction

In traditional *supervisory control*, the system being controlled (aka “plant”) has some of its transitions disabled by a controller (aka “supervisor”) in order to enforce a safety specification. Such a control strategy, while effective for several applications, is ill-suited when the application imposes certain *additional* requirements. First, in highly dynamic, adversarial environments, simply disabling transitions may disallow most or even all behaviors in the plant. To overcome this, one needs to be able to modify transitions, rather than simply disable them. Second, it is often desirable to have randomness in the control strategy. Randomness can enhance diversity, e.g., to prevent correlated failures of replicated systems, or to prevent an adversary (“attacker”) from easily inferring (and possibly thwarting) the control strategy. Finally, if randomness is employed, one often needs to impose the additional requirement that a trace of the random strategy be “similar” to a reference trace, to maintain some predictability.

We introduce a new concept that formalizes the above variation on the supervisory control problem. Informally, this concept, termed *control improvisation*, is the process of generating a randomized control strategy producing traces similar to a reference sequence and satisfying a given safety specification. We propose a formal definition of the control improvisation problem and an approach to solve it. There are several interesting applications that require control improvisation. One application concerns control in an emergency situation, such as an earthquake, where the environment deviates greatly from its specification [19]. Another application is to home automation, where for example, the lighting in a home can be programmed to switch randomly when occupants are away, but still satisfying constraints (no more than a certain number on at a time), and mimicking typical occupant behavior [20]. In this paper, we demonstrate our ideas with the problem of *music improvisation*, a compelling application that combines all three additional requirements identified above.

Music can be generated either at the audio or at a symbolic level. The former involves processing and synthesizing sound waves, whereas the latter is concerned only with generating *scores*, i.e., sequences of (groups of) symbols, the notes, each of them being an abstract representation of a particular sound that can be instantiated in many different variations by different instruments or, generally speaking, by sound synthesizers. Thus, at the symbolic level, generation of music is the same as generating sequences of letters,

each of which corresponding to a note. Music improvisation is a special case of music generation where one generates a random variant of a given melody (sequence of notes). The field of music improvisation, also termed as *machine improvisation*, has been well studied [24]. One approach to improvisation is *data-driven*, wherein recurrent patterns are inferred from the reference melody, and then replicated and recombined to form the improvisation. Different data structures and algorithms have been proposed for this purpose such as incremental parsing (IP) [14] inspired from dictionary based compression algorithms from the Lempel-Ziv family [3], probabilistic suffix trees (PST) [13], and factor oracles (FO) [2]. Another approach is *rule-based*, where an expert encodes rules in a formal system such as a stochastic context free grammar, using which sequences are generated [18]. Both approaches, however, lack certain desirable properties. First, certain rules need to be enforced always, much like safety properties. Second, it is often desirable to control the amount of “creativity” in the improvisation, using some kind of divergence measure. We present a more detailed discussion of related work in Section 7.

Our definition of control improvisation is thus a good fit for revisiting the problem of music improvisation. Specifically, we consider the scenario of generating a monophonic (solo) melody over a given Jazz song harmonization. The improvised sequence has to be synchronized with another sequence, usually the chord progressions, considered as fixed and called hereafter the accompaniment. The improviser then has to be a function of the training sequence, the accompaniment, and other imposed constraints such as the “safety” rules and divergence measure. We present an approach to solving the control improvisation problem for this specific application. Our approach has three phases. The first phase, *generalization*, learns from the given melody a (non-deterministic) automaton generating a set of melodies containing the original. We implement this phase using factor oracles [2]. The second phase, *safety supervision*, enforces rules on the generalized automaton so that it plays in harmony with the accompaniment. The rules are analogous to “safety properties” that a control system must always obey. The third and final phase, *divergence supervision*, ensures that sequences produced by the improviser automaton lie, with high probability, within a specified “similarity” divergence from the original. This phase is implemented by replacing non-determinism in the improviser automaton with probabilities, which are based on a given divergence measure. For music, several divergence measures have been proposed often in the purpose of genre classifications; amongst these, Normalized Compression Distances (NCDs) have been effectively used [8], and so we employ a variant of an NCD in this paper.

In summary, the main novel contributions of this paper are:

- The notion of *control improvisation*, its formal definition, and an analysis of its computational hardness (Sections 2 and 3);
- An approach to solve the control improvisation problem based on generalization, safety supervision, and divergence supervision (Section 4), and
- An instantiation and application of our approach to improvisation of Jazz melodies (Sections 5 and 6).

2 Control Improvisation

In this section, we define formally the control improvisation problem, which is a variant of a controller synthesis problem. The main application presented in this work is *music* improvisation, and more specifically we consider here only the symbolic aspect of music. This means we leave aside the problems of synthesizing or analysing sound signals, which would require real-valued and continuous-time signal processing, and work with traditional score notation which is based only on discrete sets, namely a discrete set of pitches (e.g., a4, c2, g3, etc) and a discrete set of durations (quarter notes \downarrow , eighth notes \updownarrow , etc). As a consequence, the formal background can be set up in terms of finite state automata.

2.1 Notation and Background

Definition 1. A finite state automaton (FSA) is a tuple $\mathcal{A} = (Q, q_0, F, \Sigma, \rightarrow)$ where Q is a set of states, $q_0 \in Q$ is the initial state, $F \subset Q$ is the set of accepting states, Σ is a finite set called the alphabet and

$\rightarrow \subset Q \times \Sigma \cup \{\epsilon\} \times Q$ is a transition relation for which we use the usual infix notation $q \xrightarrow{\sigma} q'$ to mean that $(q, \sigma, q') \in \rightarrow$, and ϵ is the empty word.

We interpret letters of the alphabet as observable events of the system under consideration. A *word* w is either ϵ or a finite sequence of letters in Σ , i.e. $w = \sigma_1 \sigma_2 \dots \sigma_k$ for some integer $k \geq 1$. The length of a word is defined inductively as $|\epsilon| = 0$ and $|w\sigma| = |w| + 1 \forall \sigma \in \Sigma$. A word is a *trace* of a FSA \mathcal{A} iff there exists a sequence of states $q_i \in Q$ such that $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_{n-1}} q_{n-1} \xrightarrow{\sigma_n} q_n$. It is an *accepting trace* of \mathcal{A} iff q_n is in F . The *language* of \mathcal{A} , noted $\mathcal{L}(\mathcal{A})$ is the set of accepting traces of \mathcal{A} .

Definition 2. (*Synchronous Product*) Given two FSA with same alphabet $\mathcal{A}^s = (Q^s, q_0^s, F^s, \Sigma, \rightarrow)$ and $\mathcal{A}^c = (Q^c, q_0^c, F^c, \Sigma, \rightarrow)$, the *synchronous product* of \mathcal{A}^s and \mathcal{A}^c , noted $\mathcal{A}^s \parallel \mathcal{A}^c$ is defined as the FSA $\mathcal{A}^s \parallel \mathcal{A}^c \triangleq (Q^s \times Q^c, (q_0^s, q_0^c), F^s \times F^c, \Sigma, \rightarrow)$ where $\forall \sigma \in \Sigma \cup \epsilon$, $(q_i^s, q_i^c) \xrightarrow{\sigma} (q_j^s, q_j^c)$ if and only if $q_i^s \xrightarrow{\sigma} q_j^s$ and $q_i^c \xrightarrow{\sigma} q_j^c$.

Note that here we consider products of FSAs sharing the same alphabet and explicit synchronization of ϵ -transitions. FSAs equipped with the synchronous product are sufficient to define a controller synthesis problem. Our work builds upon existing results from the field of supervisory control of discrete-event systems [7]. Assume that we are given an FSA \mathcal{A}^p , called the *plant FSA*, modeling the behavior of a system and an FSA \mathcal{A}^s , called the *specification FSA* modeling specifications for this system so that accepting traces of $\mathcal{A}^p \parallel \mathcal{A}^s$ represent desired behaviors of \mathcal{A}^p . Typically, the transitions of the plant automaton are classified as being *controllable* (they can be enabled or disabled) or *uncontrollable* (they are always enabled). In the context of our motivating applications, and for simplicity, all transitions can be considered as controllable. Note also that one can view \mathcal{A}^s as a safety specification, since it identifies all finite-length sequences that are good (bad) behaviors of the system.

A *supervisory controller* for \mathcal{A}^p is then an FSA \mathcal{A}^c which, when composed with \mathcal{A}^p , will *disable* (controllable) transitions leading to non-accepting traces of \mathcal{A}^s . In other words, $\mathcal{L}(\mathcal{A}^p \parallel \mathcal{A}^c) \subseteq \mathcal{L}(\mathcal{A}^s)$. A supervisory controller is said to be *non-blocking* if it always allows the composite system $\mathcal{A}^p \parallel \mathcal{A}^c \parallel \mathcal{A}^s$ to reach an accepting state. It is said to be *maximally permissive* when it does not disable more transitions than strictly necessary. There is a simple, well-known algorithm for finding a non-blocking, maximally-permissive, memoryless supervisory controller, when one exists. Informally, the algorithm is based on locating “bad” (blocking) states in the composite automaton and then iteratively pruning away controllable transitions to such states, while marking as “bad” states any uncontrollable predecessors of existing “bad” states or new blocking states. The reader is referred to the book by Cassandras and Lafortune for further details [7].

The framework of supervisory control, while relevant, is not sufficient for our setting of improvisation. There are two main differences:

- (i) *Randomness*: To improvise is to incorporate some randomness (“unpredictability”), whereas traditional supervisory control seeks to find safe, deterministic strategies, and
- (ii) *Bounded Divergence*: The improvisation is created from a *reference trace* w_{ref} , and is typically “similar” to it. The problem definition should capture this constraint.

We therefore define a new controller synthesis problem, termed as the *control improvisation* problem, in the following section.

2.2 Problem Definition

The aim of control improvisation is to randomly generate traces among a family of “safe” traces which are equivalent based on some *creativity measure*. Intuitively, a controller that uniformly samples traces from the safe set is the most creative, and a deterministic controller that replicates the reference trace w_{ref} is the least creative. Our goal is to find a controller of “intermediate creativity.” Of course, creativity is a vague, rather qualitative and subjective notion. Notwithstanding this, we assume that it can be measured by a non-negative function $d_{w_{\text{ref}}}$ on words, such that $d_{w_{\text{ref}}}(w_{\text{ref}}) = 0$ and $d_{w_{\text{ref}}}(w)$ increases as w gets “further” from w_{ref} . The control improvisation problem is then defined with respect to w_{ref} and $d_{w_{\text{ref}}}$ in addition to the plant \mathcal{A}^p and specification \mathcal{A}^s . A controller solving the control improvisation problem resolves the non-determinism in \mathcal{A}^p in two ways:

1. When several transitions of \mathcal{A}^p are safe with respect to \mathcal{A}^s , one is picked following a random distribution in accordance with the creativity criterion;
2. When no safe transition is available, one transition of \mathcal{A}^p is modified (replaced with alternative transitions to the same end state but labeled with a different event) to prevent blocking while still preserving safety.

In addition to this, we require that the process generates accepting words of a minimal length n . This is achieved by running the improvisation for at least n events, and then until an accepting state is reached. Formally, the control improvisation problem is defined as follows:

Definition 3. (*Control Improvisation Problem*) A control improvisation problem \mathcal{P}_I is an eight-tuple $(\mathcal{A}^p, \mathcal{A}^s, n, w_{ref}, d_{w_{ref}}, I, \varepsilon, \rho)$ where \mathcal{A}^p is a (possibly non-deterministic) plant FSA, \mathcal{A}^s is a specification FSA, $n \in \mathbb{N}$, w_{ref} is an accepting trace of \mathcal{A}^s of length n , $d_{w_{ref}}$ its associated creativity measure, $I = [d, \bar{d}]$ is an interval of \mathbb{R} , $\varepsilon \in (0, 1)$, and $\rho \in (0, 1]$. A solution of \mathcal{P}_I is a stochastic process generating words w in Σ^* such that the following conditions hold for each w :

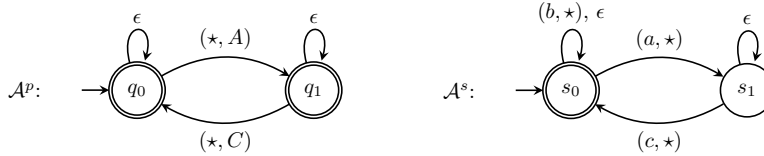
- (a) Minimal Length: $|w| \geq n$;
- (b) Safety: w is an accepting trace of $\mathcal{A}^s \parallel \mathcal{A}^p$;
- (c) Randomness: The measure of w is smaller than ρ , and
- (d) Bounded Divergence: $Pr(d_{w_{ref}}(w) \in [d, \bar{d}]) > 1 - \varepsilon$.

Typically, all states of \mathcal{A}^p are accepting states, and therefore the Safety condition (b) is determined by \mathcal{A}^s .

2.3 Running Example

We present below a running example to illustrate the definitions and approach:

- An alphabet composed of two sets of symbols $\Sigma = \Sigma_a \times \Sigma_A$, where $\Sigma_a = \{a, b, c\}$ and $\Sigma_A = \{A, C\}$
- A plant model \mathcal{A}^p and a specification automaton \mathcal{A}^s :



where we use the special symbol \star as a “don’t care” symbol. E.g., (b, \star) represents either (b, A) or (b, C) ;

- A reference word: $w_r = (b, A)(b, C)(a, A)(c, C)$.

The goal is to design a controller that produces variations of w_r satisfying \mathcal{A}^s .

3 Theoretical Hardness

In this section, we analyze the computational hardness of the Control Improvisation problem (CI), as stated in Definition 4. We will show that CI is not just *undecidable*, it is also *not* recursively enumerable (Turing-recognizable). This result follows from the undecidability of the Control Improvisation Verification problem (CIV), which informally is the verification version of CI, formalized below. The undecidability of CIV follows by a reduction from the *string-existence* problem for *probabilistic finite automata* [23,10].

3.1 Definitions and Background

We define here the problem of verifying a candidate solution to CI.

Definition 4. (*Control Improvisation Verification Problem — CIV*) The input to the control improvisation verification problem is a tuple $(\mathcal{A}^p, \mathcal{A}^s, n, w_{ref}, d_{w_{ref}}, I, \varepsilon, \rho, \mathcal{A}^i)$ where \mathcal{A}^p is a (possibly non-deterministic) plant FSA, \mathcal{A}^s is a specification FSA, $n \in \mathbb{N}$, w_{ref} is an accepting trace of \mathcal{A}^s of length n , $d_{w_{ref}}$ its associated creativity measure, $I = [d, \bar{d}]$ is an interval of \mathbb{R} , $\varepsilon, \rho \in (0, 1)$, and \mathcal{A}^i is a stochastic process generating words w in Σ^* . Given this input, the problem is to determine for each generated w whether or not the following conditions hold:

- (a) Minimal Length: $|w| \geq n$;
- (b) Safety: w is an accepting trace of $\mathcal{A}^s || \mathcal{A}^p$;
- (c) Randomness: The measure of w is smaller than ρ , and
- (d) Bounded Divergence: $Pr(d_{w_{ref}}(w) \in [\underline{d}, \bar{d}]) > 1 - \varepsilon$.

It is well known in computability theory that a language is recursively enumerable (Turing-recognizable) if and only if it is *verifiable*. Informally, a language L is verifiable if there exists a decision procedure that, given a problem instance P in L and a candidate solution S to that instance, outputs YES if S is a solution to P , and NO otherwise. From this classic result, it follows that:

Proposition 1. *CI is recursively enumerable if and only if CIV is decidable.*

We therefore turn our focus to analyzing the decidability of CIV. In particular, we show that CIV is undecidable using a reduction from the string-existence problem for probabilistic finite automatas (PFAs) [23,10]. For this, we first introduce the notion of a PFA, a simple stochastic process.

Definition 5. (*Probabilistic Finite Automaton [22]*) A probabilistic finite automaton (PFA) is a 5-tuple (Q, Σ, T, s, f) where Q is a finite set of states, Σ is the input alphabet, T is a set of $|Q| \times |Q|$ row-stochastic transition matrices, one for each symbol in Σ , $s \in Q$ is the initial state, and $f \in Q$ is the accepting state.

The automaton occupies a single state in Q at any given point of time. It begins in state s , transitions from state to state based on the current input symbol and the distribution defined by the corresponding stochastic transition matrix, and halts when it transitions to the accepting state f . It can be assumed that f is an absorbing state, meaning that the automaton stays in f after reaching it.

A PFA accepts a string $w \in \Sigma^*$ if the automaton ends in the accepting state after reading string w , otherwise it rejects it. For any finite-length string w accepted by a PFA, there is an associated probability with which w is accepted. Given these notions, the problem of interest in this paper is the following one:

Definition 6. (*String-Existence Problem for PFAs [22]*) Given a probabilistic finite automaton (PFA), decide whether or not there is some input string $w \in \Sigma^*$ such that the given PFA accepts that string with probability exceeding some input threshold τ .

Paz [23] established the undecidability of this problem. Later, Condon and Lipton [10] gave an alternative proof. More recently, Blondel and Canterini showed that the problem remains undecidable even when the alphabet Σ has just two letters [5].

We also mention here that the above string-existence problem is essentially equivalent to the problem of *probabilistic planning*, as shown by Madani et al. [22] (which is therefore also undecidable). The probabilistic planning problem is relevant to our setting since it is a form of controller synthesis: a plan is a string that is a sequence of actions navigating from an initial state to a goal state.

3.2 CIV is Undecidable

We now give our reduction of the string-existence problem of PFAs to CIV, which yields the following theorem.

Theorem 1. *CIV is undecidable.*

Proof: Consider an instance of the PFA string-existence problem: a pair (\mathcal{P}, τ) , where \mathcal{P} is a PFA.

We create an instance of CIV as a tuple $(\mathcal{A}^p, \mathcal{A}^s, n, w_{ref}, d_{w_{ref}}, I, \varepsilon, \rho, \mathcal{A}^i)$ as follows:

- \mathcal{A}^i equals \mathcal{P} with a slight modification where we add one more letter ξ to \mathcal{P} 's alphabet and direct all strings with any occurrence of that letter to a non-accepting sink state with probability one transitions;
- ε is set to $1 - \tau$;
- Set both \mathcal{A}^p and \mathcal{A}^s to be the universal automaton that accepts all strings in Σ^* ;
- $n = 1$;

- w_{ref} is any string with the letter ξ ;
- Set $d_{w_{\text{ref}}}(w)$ to be 0 if the string w contains ξ , otherwise 1;
- I is set to be $[1, 1]$, and
- ρ is set to be 1.

Note that conditions (a)-(c) are trivially satisfied, so the CIV problem reduces to one of checking (d): that $\Pr(d_{w_{\text{ref}}}(w) = 1) > \tau$. The strings w generated by \mathcal{A}^i satisfying $d_{w_{\text{ref}}}(w) = 1$ are exactly those that are accepted by \mathcal{P} . Therefore, there is a string accepted by \mathcal{P} with probability exceeding τ if and only if condition (d) is satisfied, i.e., iff \mathcal{A}^i is a valid solution to the CIV problem. \square

3.3 Hardness of Control Improvisation

Proposition 1 and Theorem 1 taken together imply the following hardness result:

Theorem 2. *The Control Improvisation Problem (CI) is not Recursively Enumerable.*

We make some observations about this result and its implications:

- First, we note from the proof of Theorem 1 that the hardness stems from the presence of condition (d). As we will show in the following section, it is easy to synthesize an \mathcal{A}^i that satisfies conditions (a)-(c). This indicates that one approach toward making the problem decidable would be to somewhat relax condition (d), for example, by imposing additional conditions on the form of the divergence measure $d_{w_{\text{ref}}}$.
- Second, we note that the class of stochastic process required to obtain the hardness result was relatively weak — a probabilistic finite automaton (PFA). It has been observed that the PFA is similar in expressive power to other models widely used in planning and optimization, such as partially-observable Markov Decision Processes (POMDPs) [22]. Therefore, one approach to make the problem simpler is to restrict the class of stochastic processes even more, to a model that is less expressive than arbitrary PFAs.

Given the computational hardness of CI, in the following section we present an approach to solve the problem that works well in practice. Our approach synthesizes a controller \mathcal{A}^i that ensures that conditions (a)-(c) hold, and uses heuristics to satisfy condition (d) in practice (without theoretical guarantees).

4 Approach

Our approach has three components:

1. *Generalization from the reference sequence:* we compute an FSA \mathcal{A}^g that accepts w_{ref} and variations of it;
2. *Safety Supervision:* we compose \mathcal{A}^g with \mathcal{A}^p , adding safe transitions with respect to \mathcal{A}^s where needed, and
3. *Divergence Supervision:* we tune transition probabilities in the modified $\mathcal{A}^g || \mathcal{A}^p$ with respect to a creativity measure that we define.

We now describe each of these components in more depth.

4.1 Generalization using Factor Oracles

The core of our improvisation approach is based on the factor oracle (FO) structure [2]. A factor oracle is a compact automaton representation of all contiguous subwords (factors) contained in a word $w = \sigma_1\sigma_2 \dots \sigma_n$. It has $|w| + 1$ states, all accepting, and its transitions can be categorized into

1. *Direct* transitions of the form $s_i \xrightarrow{\sigma_{i+1}} s_{i+1}$;
2. *Forward* transitions of the form $s_i \xrightarrow{\sigma} s_j$ where $j > i + 1$ and σ is some letter in w ;
3. *Backward* transitions, also called *suffix links*, of the form $s_i \xrightarrow{\epsilon} s_j$ with $j < i$.

The details of the construction of factor oracles, can be found in [9]. Some properties of FOs are as follows:

- An accepting word that takes only direct transitions is a prefix of w ;
- Factors of w are accepting words taking only direct and forward transitions;
- Finite concatenation of factors of w are accepting word taking all three types of transitions.

These properties make the FO a suitable structure to generalize w_{ref} , so a first step to solve the control improvisation problem is to define $\mathcal{A}^g = \text{FO}(w_{\text{ref}})$. In Figure 1, we show the factor oracle obtained from the reference word $bbac$.

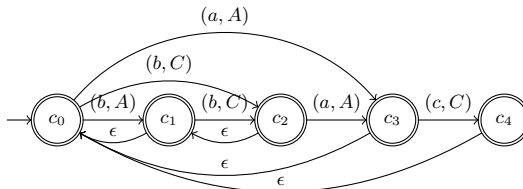


Fig. 1. Factor Oracle improviser obtained from the reference word $bbac$.

4.2 Safety Supervision

Even though w_{ref} is an accepting word for the plant \mathcal{A}^p and specifications \mathcal{A}^s , there is no guarantee that its generalization \mathcal{A}^g composed with \mathcal{A}^p is non-blocking for \mathcal{A}^s . However, assuming that there exists a non-blocking memoryless controller $\mathcal{A}_{\text{max}}^c$ for $\mathcal{A}^p \parallel \mathcal{A}^s$ — something that can be checked using standard supervisory control [7] and which is guaranteed by the existence of w_{ref} — it is always possible to make $\mathcal{A}^p \parallel \mathcal{A}^g \parallel \mathcal{A}^s$ non-blocking by adding transitions as follows. Let (q, c, s) be a blocking state of $\mathcal{A}^p \parallel \mathcal{A}^g \parallel \mathcal{A}^s$. Since $\mathcal{A}_{\text{max}}^c$ is a non-blocking memoryless controller, there exists a non-blocking transition $(q, s) \xrightarrow{\sigma} (q', s')$ in $\mathcal{A}^p \parallel \mathcal{A}^s$ for some $\sigma \in \Sigma$. Hence we can pick some state c' in \mathcal{A}^g and add the transition $c \xrightarrow{\sigma} c'$ to the transition relation of \mathcal{A}^g . This effectively adds the transition $(q, c, s) \xrightarrow{\sigma} (q', c', s')$ in $\mathcal{A}^p \parallel \mathcal{A}^g \parallel \mathcal{A}^s$. This procedure is repeated until no blocking state can be found in $\mathcal{A}^p \parallel \mathcal{A}^g \parallel \mathcal{A}^s$.

To illustrate this construction, consider automaton \mathcal{A}^p , \mathcal{A}^s defined in Sec. 2.3 and \mathcal{A}^g on Fig. 1. Constructing the product $\mathcal{A}^p \parallel \mathcal{A}^s \parallel \mathcal{A}^g$, we find that the run

$$(q_0, s_0, c_0) \xrightarrow{(b,A)} (q_1, s_0, c_1) \xrightarrow{(b,C)} (q_0, s_0, c_2) \xrightarrow{(a,A)} (q_1, s_1, c_3) \xrightarrow{(c,C)} (q_1, s_1, c_0)$$

leads to a blocking state (q_0, s_1, c_0) , as \mathcal{A}^s requires a c transition, whereas \mathcal{A}^g only permits an a or a b . Hence we add the transition $c_0 \xrightarrow{(c,C)} c_1$.

One remaining question is how to pick the un-blocking transition when more than one choice is possible. When this makes sense, we should add a transition which is close to an existing transition of \mathcal{A}^g . For example, in our music application where transitions corresponds to note events, we can pick notes with the closest pitch (or frequency) or duration.

4.3 Divergence Supervision via Probability Assignment

The last step is to define transition probabilities satisfying the Randomness and Bounded Divergence requirements. We begin by concretizing the creativity divergence that we use, which is a variant of the Normalized Compression Distance introduced in [21] and is based on the theory of Kolmogorov complexity. The Kolmogorov Complexity of an object x (denoted $K(x)$) is defined as the length of the shortest compressed code

to which x can be losslessly reduced. The Kolmogorov Complexity of y given x (denoted $K(y|x)$) is the length of the shortest compressed code to which y can be losslessly reduced assuming knowledge of x . In practice, $K(x)$ is not computable, and so is typically approximated by $C(x)$ where $C(x) = \text{length}(\text{compress}(x))$ for some compression algorithm compress and $K(y|x)$ can be approximated by $C(y|x) = C(xy) - C(x)$ [8]. Then the Normalized Compression Distance (NCD) [21] between x and y is defined as

$$NCD(x, y) = \frac{\max(C(x|y), C(y|x))}{\max(C(x), C(y))}.$$

Informally, it estimates 1 minus the mutual information in x and y . In our case however, the amount of information in an improvisation that is not in the reference trace is of more interest than mutual information, hence we define the creativity divergence based on the asymmetric quantity $\frac{C(y|x)}{C(y)}$, as follows.

Definition 7 (Creativity Divergence $d_{w_{\text{ref}}}$).

$$d_{w_{\text{ref}}}(w) = \frac{C(w|w_{\text{ref}})}{C(w)} + (1 - \frac{C(w_{\text{ref}}w_{\text{ref}})}{C(w_{\text{ref}})})$$

The second term in the sum ensures that $d_{w_{\text{ref}}}(w_{\text{ref}}) = 0$. In our application we used the LZW compression algorithm to compute $C(\cdot)$.

Finally, a simple way to assign probabilities to transitions in a FO is as follows. Recall that traversing the $n + 1$ states in sequence by taking direct transitions reproduces w_{ref} . Improvisation, i.e., variation from the original sequence, is obtained by randomly taking forward transitions or backward transitions. Thus the higher the probability of taking direct transitions, the more similar the output is to the original sequence. In our implementation, we assign the probability p to each direct transition, so that the improviser replicates w_{ref} when $p = 1$, and probability $1 - p$ equi-distributed to other outgoing forward or backward transitions. We assume that p is such that the direct transition is always the most probable. This provides for a simple parameter controlling how different the improvised sequence is from w_{ref} .

The overall supervision process for solving the control improvisation problem is summarized below:

1. Maintain a sequence $(q_0, c_0, s_0)(q_1, c_1, s_1) \dots (q_k, c_k, s_k)$ of states of $(\mathcal{A}^p || \mathcal{A}^g) || \mathcal{A}^s$ and a word $w_k = \sigma_0 \sigma_1 \dots \sigma_k \in \Sigma^k$,
2. If $k \geq n$ and s_k is accepting, return $w = w_k$
3. Else if (q_k, c_k, s_k) has outgoing transitions (non-blocking), assign probabilities according to replication probability p and pick σ_{k+1} and $(q_{k+1}, c_{k+1}, s_{k+1})$
4. Else if (q_k, c_k, s_k) is blocking, pick a safe σ_{k+1} and $(q_{k+1}, c_{k+1}, s_{k+1})$ as defined in Section 4.2.

Theorem 3. *Assume that $\mathcal{A}^p || \mathcal{A}^s$ is non-blocking, $n \geq \frac{\log \rho}{\log p}$ and $d_{w_{\text{ref}}}$ is measurable. Then the stochastic process defined above solves the control improvisation problem $(\mathcal{A}^p, \mathcal{A}^s, n, w_{\text{ref}}, d_{w_{\text{ref}}}, I, \varepsilon, \rho)$ with probability 1 for some ε .*

Proof: Since $\mathcal{A}^p || \mathcal{A}^s$ is non-blocking, Section 4.2 showed that $(\mathcal{A}^p || \mathcal{A}^g)$ augmented with safe transitions is non-blocking as well, which means that an accepting state of $\mathcal{A}^p || \mathcal{A}^g || \mathcal{A}^s$ is always reachable in a finite number of steps. Each transition probability is non-zero, hence an accepting state must be reached in a finite number of steps with probability 1. By step 2, the process then stops and returns an accepting word satisfying the minimal length criterion. As direct transitions have always the highest probability, w_{ref} is returned with the highest probability. This probability is equal to p^n which is smaller than ρ when $n \geq \frac{\log \rho}{\log p}$, hence the randomness criterion is met. Finally, since $d_{w_{\text{ref}}}$ is measurable, so is the event $d_{w_{\text{ref}}}(w) \in [\underline{d}, \bar{d}]$ which means that there exists an $\varepsilon > 0$ such that $Pr(d_{w_{\text{ref}}}(w) \in [\underline{d}, \bar{d}]) > 1 - \varepsilon$. $\square \square$ Note that this does not provide a fully constructive solution to the control improvisation problem, as ε is a part of the problem. However, it provides a reasonable empirical solution: for a given replication probability p , one can generate a populations of improvisations, estimate ε for given $[\underline{d}, \bar{d}]$, and repeat the process tuning p accordingly until obtaining a satisfactory result. In our experiments, we obtained narrow creativity intervals with $\varepsilon = 5\%$ with 100 improvisations (see Section 6).

5 Jazz Control Improvisation

In this section, we apply control improvisation to Jazz music.

5.1 Musical Notations

We start with some musical vocabulary. The *note* is the atomic entity. It has two attributes: a *pitch*, which represents its fundamental frequency, and a *duration*. A pitch is noted with a letter, from **a** to **g**, a number from 0 to 8 denoting the *octave* and an optional *accidental* #, not available for pitches **b** and **e**. Pitches are ordered as: $\{\mathbf{a}0, \mathbf{a}0\#, \mathbf{b}0, \mathbf{c}0, \mathbf{c}0\#, \mathbf{d}0, \dots, \mathbf{e}8, \mathbf{f}8, \mathbf{f}8\#, \mathbf{g}8, \mathbf{g}8\#\}$. We call the difference between two consecutive pitches a *semi-tone*. When the octave is not specified, it denotes the set of all pitches that differs only by their octaves, e.g., $\mathbf{c} = \{\mathbf{c}0, \mathbf{c}1, \dots, \mathbf{c}8\}$. A *rest* can be defined as a note with no pitch, or a silent note. The set of durations is also a finite ordered set $\{\downarrow, \downarrow, \downarrow, \dots\}$ where two consecutive durations differ by a power of two:

$$\downarrow = \downarrow \downarrow = \downarrow \downarrow \downarrow \downarrow \dots$$

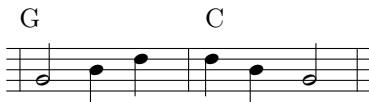
A *chord* is a finite set of pitches, usually noted using a capital letter matching one of its pitch elements called its *root*, and additional letters characterizing its nature (major, minor, dominant, etc).

A piece of jazz music can be simplified into a *melody*, a string of pitched notes and rests, aligned with an *accompaniment*, a looping sequence of chords with given durations. The time unit is the *beat* and the piece is divided into bars which are sequences of 4 beats. We assume that the accompaniment is fixed and our goal is to define an improviser for the melody. Hence, the plant FSA will model the behavior of the accompaniment, without constraining the melody, and the specification FSA will set constraints on acceptable melodies played together with the accompaniment. To encode all events in a score, we use an alphabet composed of the cross-product of four alphabets: $\Sigma = \Sigma_p \times \Sigma_d \times \Sigma_c \times \Sigma_b$, where

- Σ_p is the *pitches* alphabet, e.g., $\Sigma_p = \{\downarrow, \mathbf{a}0, \mathbf{a}\#0, \mathbf{b}0, \mathbf{c}0, \dots\}$,
- Σ_d is the *durations* alphabet, e.g., $\Sigma_d = \{\downarrow, \downarrow, \downarrow, \dots\}$ with $\downarrow = 1$ beat. Note that Σ_d also includes fractional durations, e.g., for triplets, as discussed below;
- Σ_c is the *chords* alphabet, e.g., $\Sigma_c = \{\mathbf{C}, \mathbf{C}7, \mathbf{G}, \mathbf{E}maj, \mathbf{A}dim, \dots\}$,
- Σ_b is the *beat* alphabet. E.g, if the smallest duration (excluding fractional durations) is the eighth note, i.e., half a beat, then $\Sigma_b = \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5\}$.

All automata in the following will use implicitly the full alphabet Σ . However, each component alphabet is meant to address one particular aspect of the music formalization, and we will construct the specification automaton by the composition of different sub-automata using these different component alphabets. Also, note that this encoding of music is of course not unique nor meant to be canonical, and other types of alphabets can be used in replacement to or to complement the one we propose and used. E.g., we do not consider here note *velocity* (i.e., the intensity of the sound of the note).

Example 1. we provide an example encoding of a simple score using the formalism defined in Section 5. Consider the following extract:



It contains a melody and a chord progression which is represented by the following word in our alphabet:

$$(\mathbf{g}4, \downarrow, \mathbf{G}, 0) (\mathbf{b}4, \downarrow, \mathbf{G}, 2) (\mathbf{d}5, \downarrow, \mathbf{G}, 3) (\mathbf{d}5, \downarrow, \mathbf{C}, 0) (\mathbf{b}5, \downarrow, \mathbf{C}, 1) (\mathbf{g}4, \downarrow, \mathbf{C}, 2)$$

5.2 Encoding Chord Progressions

The harmonic context of the melody is given by the chord progression (accompaniment). The plant FSA \mathcal{A}^p then encodes the events of specified chords at specified times. The basic idea of the encoding is to define as many states as there can be events of the minimal possible duration in a bar, i.e., in four beats, and replicate those states for as many bars as needed. Then transitions from one state q to another state q' is possible with a note of the proper duration is possible and if in the duration of this note, there is no chord change. This construction is illustrated in Figure 2.

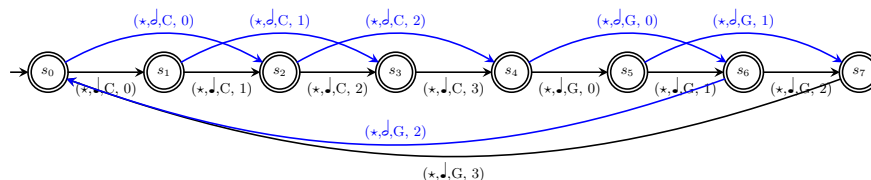



Fig. 2. Chords progression automaton \mathcal{A}^p of the example. It consists in an accompaniment looping on chord C during 4 beats (1 bar) and chord G during 1 bar, with duration alphabet restricted to quarter notes and half notes.

5.3 Rhythmic and Harmonic Specifications

The specification FSA encodes rhythmic and harmonic constraints involving notes in the melody which enforce some general structure and basic musical consistency. The following specifications are adapted and simplified from the generic guidelines found in [17]. We structure Jazz melodies into *licks* defined informally as short melodic phrases of pitched notes separated by either rests or long notes. Then we impose that licks start on specific beats. E.g., start beats can be 0.5, 1.5, 2.5 or 3.5, i.e., *off-beats*. This specification can be encoded in the automaton (a) on Fig. 3.

The second specification has to do with durations which are not multiple of the smallest duration. In that case, we require that it be repeated until the total duration is such a multiple. The typical example

of this situation is the triplet, e.g., , which is the concatenation of three notes of duration $\frac{1}{3}$, noted $\frac{3}{\text{♪}}$. Without loss of generality, we model only this case, shown as the FSA (b) in Fig. 3, as other fractional durations are dealt with in a similar manner.

Finally, we define constraints on the pitches of the notes in the melody. The pitched notes are classified based on their accompanying chord. We follow the three primary tone classifications as described in [17]:

- **Chord tone:** a pitch belonging to the current chord;
- **Color tone:** a pitch that does not belong to the current chord but complements and creates euphony with the current chord;
- **Approach tone:** neither a chord nor color tone that is followed by pitched note that differs by exactly 1 semitone;

This classification provides a set of “good” pitches for each chord. Color tones can be defined by a scale, i.e., a set of pitches, which is overall “compatible” with the whole song, and to which we remove potential “avoid” notes for the current chord. As an example, consider a song in the key of C. All notes in the major scale $\{c, d, e, f, g, a, b\}$ are safe to be played in general, however if an F chord is played (composed of pitches $f, g\#, c$), we need to avoid b which is highly dissonant with f . Hence the set of good notes in that situation

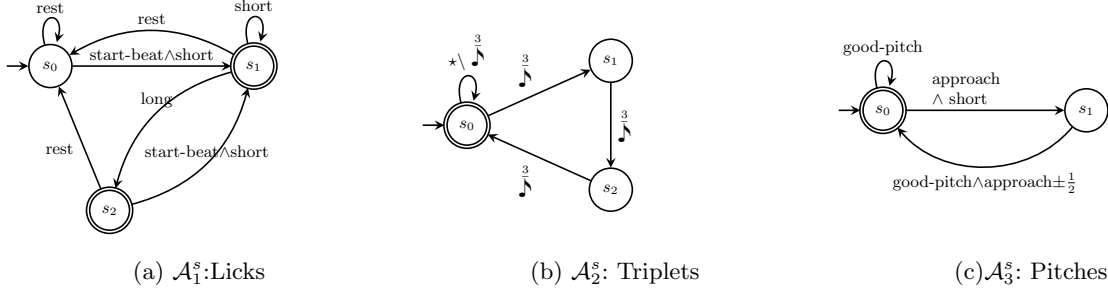
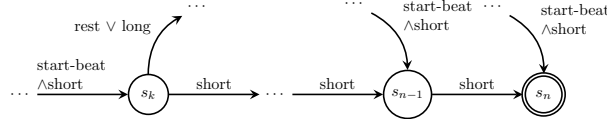


Fig. 3. Specification automata $\mathcal{A}^s = \mathcal{A}_1^s || \mathcal{A}_2^s || \mathcal{A}_3^s$. “rest” indicates a rest in the melody of any duration. “start-beat” indicates a label of the form (\star, \star, \star, b) where b is a beat value for which a lick can start. “short” indicates a note in the melody of short duration, e.g., of duration less or equal to a beat (\downarrow). Conversely, “long” indicates a note of a longer duration, e.g., strictly more than \downarrow . “good-pitch” indicates a note with a pitch which is either a chord or a color tone. “approach” indicates an approach tone. “approach $\pm \frac{1}{2}$ ” indicates an approach tone plus or minus a semi-tone.

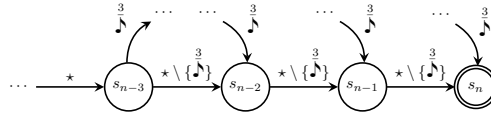
is $\{c, d, e, f, g, g\#, a\}$. The approach tones make it possible to deviate “temporarily” from these good notes: if a note not classified as good is played, it must be short and followed by a good note immediately and not further than a semi-tone away from it. We simplify this into the automaton (c) in Fig. 3.

5.4 Specifications Controllability

In this section, we provide a supervisory controller for a plant automaton \mathcal{A}^p modeling Jazz accompaniment and a specification $\mathcal{A}^s = \mathcal{A}_1^s || \mathcal{A}_2^s || \mathcal{A}_3^s$ defined as above. We first note that in order to satisfy \mathcal{A}_1^s , a necessary condition is that a state of \mathcal{A}^p with outgoing transition labeled with (\star, \star, \star, b) where $b \in$ start beat is accessible in k steps with $k \leq n - 1$. Then a controller for \mathcal{A}_1^s can take the following form:

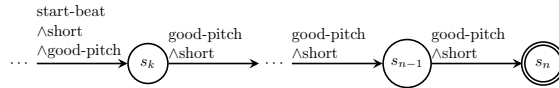


Regarding \mathcal{A}_2^s , there is no need to make any assumption for the existence of a controller, as a conservative strategy simply avoiding notes with fractional durations is always accepting. It is then easy to derive the most permissive controller which can additionally use fractional durations, provided it remains enough steps with the same fractional duration to resolve to an integer duration before step n :



Similarly, for \mathcal{A}_3^s , a conservative strategy consisting in always picking “good” pitches is guaranteed to produce an accepting trace. This requires that the set of good pitches is never empty. This is the case as this set includes in particular the chord tones, i.e., the pitches included in the current chord. A more permissive controller could also take detours through approach tones at every step except for the last one.

Clearly by composing the three strategies, it is easy to extract a valid controller for \mathcal{A}^s . One possibility is the following:



In words, we use rests with appropriate durations for k steps until a starting beat is available, then we use only short notes with “good” pitches until producing a length n trace.

5.5 Jazz Improviser Architecture

The automaton obtained by composing the specifications above with the accompaniment automaton is non-blocking; thus, we can apply the approach proposed in Section 4. However, our early experiments showed that a single viewpoint system in which the model predicted note duration and pitches together was too inflexible, in that the supervisory control phase would have to many edges to the factor oracle generator, therefore we adopted a multiple viewpoint system which improvises rhythms and melodic pitches separately. The architecture presented in Figure 4 has been implemented in Python, using the Music21 library.⁴ We present some results in the next section, as well as on a dedicated webpage⁵.

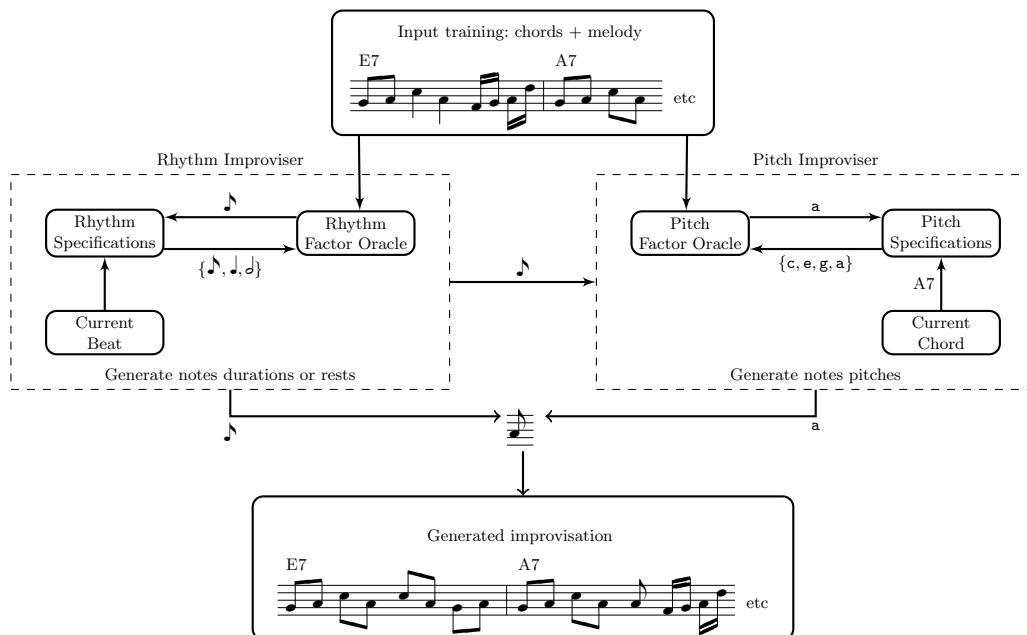


Fig. 4. Architecture of the improviser with multiple view points.

6 Results

We evaluated our improviser using a melody generated by Improvisor ([18]) over the standard 8-bar blues chord progression. Using this melody as the reference trace, we generated an improvisation from a supervised factor oracle and an improvisation from an unsupervised factor oracle both with probability of direct transitions assigned to be $p = 0.8$ (Figure 5). The reference melody and the supervised improvisation share several similarities, however the improvisation also deviates sufficiently from the original to be considered unique.

⁴ <http://web.mit.edu/music21>

⁵ http://www.eecs.berkeley.edu/~donze/impro_page.html



Fig. 5. Training melody (a), improvisation generated by supervised factor oracle (b), improvisation generated by unsupervised factor oracle (c). Black notes are chord tones, green notes are color tones, blue notes are approach tones, and red notes are other (undesirable) tones.

The unsupervised improvisation contains several notes (highlighted in red) that are not chord, color, or approach tones and which are therefore undesirable. Conversely, supervisory control preserves the melodicism of the improvisation as evidenced by the lack of red notes in the supervised improvisation.

We also evaluate the supervised factor oracle improviser by the creativity divergence defined in Section 4.3. In particular, we evaluate improvisational creativity with respect to rhythm (Figure 6). Overall, the plot shows an inverse relationship between replication probability and average creativity, confirming that creativity correlates with how similarly an improvisation mimics the original melody and suggesting that it is possible to bound creativity by an appropriate choice of replication probability.

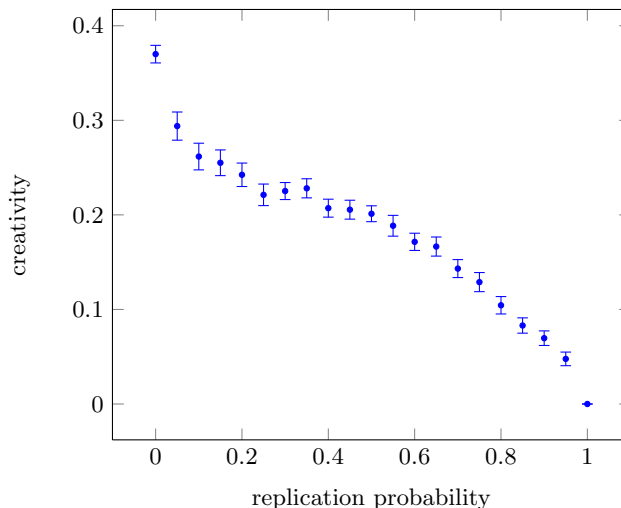


Fig. 6. Average creativity with respect to rhythm of 100 improvisations generated by factor oracle, with $\varepsilon = 5\%$ confidence intervals.

7 Related Work

To our knowledge, a concept like control improvisation has not been introduced in the literature before. We survey the most closely related work in the areas of music improvisation and control theory.

Broadly speaking, there are two approaches to automatic music improvisation: *rule-based* and *data-driven*. Rule-based approaches attempt to define the rules of “good” improvisations and generate pieces of music that follow those rules. However, it has been observed that it is difficult to come up with the “right” rules, resulting in systems that are either too restrictive, limiting creativity, or too relaxed, thereby allowing musical dissonance [12,1,18]. Consequently, recent musical improvisers tend towards *data-driven* or “predictive” approaches that employ machine learning. These approaches learn a probabilistic model from music samples, and use that model to generate new melodies. Examples of such models include stochastic context-free grammars (SCFGs) [15,17], hidden Markov models (HMMs) [16], and universal predictors [13,1,2,6]. Some approaches combine rule-based and data-driven approaches; e.g., the Improvisor system [18] based on SCFGs has rules learned from training licks through the grammatical inference [15].

It has been found that certain universal predictors outperform other stochastic models in producing stylistically appropriate music [1]. Universal predictors vary based on the data structures and algorithms used, such as incremental parsing (IP) [14] inspired from dictionary based compression algorithms from the Lempel-Ziv family [3], probabilistic suffix trees (PST) [13], and factor oracles (FO) [2]. Amongst these, it has been found that the latter has some advantages. Unlike both IP and PST, the factor oracle is both complete

(contains all factors of the given word) and can be constructed on the fly. Due to this, factor oracles are at the core of the OMax improvisation system ⁶ developed at IRCAM and which has been used in a number of performances.

Our approach extends this state of the art by providing a way to (i) enforce certain “safety” rules on the generated melody, and (ii) bounding the “creativity” divergence from the original melody. Also, many of the improvisers discussed rely on a single viewpoint system. In other words they attempt to encapsulate and improvise all aspects (rhythm, pitches, volume, etc.) of an improvisation simultaneously. For example, in [1] the alphabet of the prediction model is the cross product of the beat each note starts on, the note’s pitch, and the note’s duration. Following Conklin and Cleary [11] we implemented a more flexible multiple viewpoint approach to music generation in which note aspects are predicted separately and then aggregated.

In the area of control, our problem varies from traditional supervisory control as noted in Sections 1 and 2. Perhaps the closest existing notion is that of *adaptive control* [4], in which the controller deals with a highly-dynamic environment by learning parameters of an environment model, and adapting the control strategy to changing parameter values. While control improvisation does involve generalization (learning), it must also meet additional constraints on randomness and divergence with respect to a reference trajectory. Further our work is so far limited to the purely discrete setting, whereas much of adaptive control is in the continuous-time setting.

8 Conclusion

We introduced the concept of control improvisation and presented an approach to solve it. Our approach shows promise for automatic improvisation of Jazz music. We believe this paper is just a first step, and there is plenty of room for further work on both theory and applications of control improvisation. In particular, this work can be seen as a variation of discrete supervisory control with additional randomness and improvising requirements in the solution. Other types of controller synthesis problem could be adapted similarly such as, e.g., reactive controller synthesis with LTL specifications.

More work is required to investigate the full space of possible creativity divergence measures in the context of various applications. Moreover, there is room for improvement over the base approach we present in this paper, e.g., to provide stronger theoretical guarantees for the “bounded distance” condition.

For the application to Jazz improvisation, one can consider inferring the specification automaton from examples of “good” and “bad” melodies. Further, it would be interesting to consider real-time improvisation and improvising collectively on a set of melodies rather than just a solo piece. Finally, while this paper presented a musical application of the factor oracle combined with creativity selection, there are other potential application domains to be explored, including home automation, software testing (generating strings to test a program), secure control systems, and emergency management.

Acknowledgments

This research was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. The third author was also supported in part by NSF Expeditions grant CCF-1138996 and by an Alfred P. Sloan Research Fellowship. The second author began this work as a participant in the SUPERB summer research program at UC Berkeley.

References

1. G. Assayag and S. Dubnov. *Mathematics and Music: A Diderot Mathematical Forum*, chapter Universal Prediction Applied to Stylistic Music Generation. Springer, Berlin, 2002.

⁶ <http://repmus.ircam.fr/omax/home>

2. G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–610, 2004.
3. G. Assayag, S. Dubnov, and O. Delerue. Guessing the composers mind: Applying universal prediction to musical style. In *Proceedings of the International Computer Music Conference*, pages 496–499, 1999.
4. K. J. Åström and B. Wittenmark. *Adaptive Control*. Courier Dover Publications, 2008.
5. V. D. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36(3):231–245, 2003.
6. G. Cabral, J.-P. Briot, and F. Pachet. Incremental parsing for real-time accompaniment systems. In *The 19th International FLAIRS Conference, Special Track: Artificial Intelligence in Music and Art*, Melbourne Beach, USA, 2006.
7. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
8. R. Cilibrasi, P. Vitányi, and R. De Wolf. Algorithmic clustering of music. In *Web Delivering of Music, 2004. WEDELMUSIC 2004. Proceedings of the Fourth International Conference on*, pages 110–117. IEEE, 2004.
9. L. Cleophas, G. Zwaan, and B. W. Watson. Constructing factor oracles. In *In Proceedings of the 3rd Prague Stringology Conference*, 2003.
10. A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–467. IEEE Computer Society, 1989.
11. D. Conklin and J. G. Cleary. Modelling and generating music using multiple viewpoints. In *Proceedings of the First Workshop on AI and Music*. University of Calgary, 1988.
12. D. Cope. *Computers and Musical Styles*. Oxford University Press, 1991.
13. S. Dubnov, G. Assayag, and O. L. G. Bejerano. A system for computer music generation by learning and improvisation in a particular style. *IEEE Computer*, 10(38), 2003.
14. S. Dubnov, G. Assayag, and R. El-Yaniv. Universal classification applied to musical sequences. In *Proceedings of the International Computer Music Conference*, pages 332–340, 1998.
15. J. Gillick, K. Tang, and R. M. Keller. Learning jazz grammars. In *Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, Porto, Portugal, 2009.
16. J. R. Gillick. A clustering algorithm for recombinant jazz improvisations, 2009.
17. R. M. Keller. *How to Improvise Jazz Melodies*, 2012.
18. R. M. Keller and D. R. Morrison. A grammatical approach to automatic improvisation. In *Proceedings SMC'07, 4th Sound and Music Computing Conference*, pages 330 – 337, Lefkada, Greece, 2007.
19. J. M. Kendra and T. Wachtendorf. Improvisation, creativity, and the art of emergency management. In H. Durmaz, B. Sevinc, A. S. Yala, and S. Ekici, editors, *Understanding and Responding to Terrorism*, pages 324–335. IOS Press, 2007.
20. E. A. Lee. Personal Communication, 2013.
21. M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi. The similarity metric. *Information Theory, IEEE Transactions on*, 50(12):3250–3264, 2004.
22. O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 541–548. AAAI Press / The MIT Press, 1999.
23. A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
24. R. Rowe. *Machine Musicianship*. MIT Press, 2001.