

Assisted Lead Sheet Composition using FlowComposer

Alexandre Papadopoulos^{1,2}, Pierre Roy¹, and François Pachet^{1,2}

¹ Sony CSL, 6 rue Amyot, 75005, Paris, France

² Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, France

`alexandre.papadopoulos@lip6.fr`

`pachetcs1@gmail.com`

`roy@csl.sony.fr`

Abstract. We present FlowComposer, a web application that helps users compose musical lead sheets, i.e. melodies with chord labels. FlowComposer integrates a constrained-based lead sheet generation tool in which the user retains full control over the generation process. Users specify the style of the lead sheet by selecting a corpus of existing lead sheets. The system then produces a complete lead sheet in that style, either from scratch, or from a partial lead sheet entered by the user. The generation algorithm is based on a graphical model that combines two Markov chains enriched by Regular constraints, representing the melody and its related chord sequence. The model is sampled using our recent result in efficient sampling of the Regular constraint. The paper reports on the design and deployment of FlowComposer as a web-service, part of an ecosystem of online tools for the creation of lead sheets. FlowComposer is currently used in professional musical productions, from which we collect and show a number of representative examples.

Keywords: music generation, graphical models, belief propagation, sampling, web-service, user interaction

1 Introduction

Modelling polyphonic music is a particularly challenging task in artificial intelligence. This is probably because music, even in its simplest form, manifests itself under many interdependent dimensions, such as melody (successions of notes in time), harmony (simultaneous notes or chord labels) and meter (constraints on durations of notes making up a bar for instance). Constraint programming has been extensively studied to model polyphonic music [1]. However CSP approaches require expert musicians to encode explicitly the rules (e.g. of harmony) as constraints, and this task is not always possible nor desirable, as these constraints usually correspond to a fixed and slightly outdated musical style.

Recent advances in machine-learning and graphical models have managed to model all these dimensions in single statistical models, such as deep networks [3]. These models have been shown to be able to capture various statistical properties

of musical style. However, they are difficult to control, and have not developed into systems mature enough to be used outside specific demos.

In this paper, we address a specific case of polyphonic music: lead sheets. Lead sheets consist in monophonic melodies, augmented with chord labels (see Figure 5). Lead sheets are routinely used in popular music, including pop, rock, jazz or Brazilian music. Lead sheets also have a strong commercial value as they are the primary asset of music publishing companies. We describe an application called FlowComposer, a lead sheet composition tool. The basic concept is to provide an online lead sheet editor enriched with style imitation generation capabilities. With FlowComposer, users can generate fully-fledged lead sheets based on partially specified information, that conform to the style of a given composer (or set of lead sheets). Generated lead sheets satisfy several types of constraints: 1) user constraints, 2) metrical constraints, and 3) style constraints. Technically, this paper has three contributions. First, we show how to sample metrically constrained Markov sequences, using our recently introduced model [12]. Then, we show how to exploit this framework to enforce stochastic temporal constraints. Finally, we define a two-voice model for chord and note generation: each voice is a metrically constrained Markov sequence, and we synchronise the two voices using stochastic temporal constraints, to enforce harmony. FlowComposer is a working application that is being used in professional music projects (see Section 4.3). The paper describes the technical challenges addressed in designing and deploying FlowComposer and some interesting uses of the system.

2 Background on Constrained Markov Models

Markov models have long been used to generate music in the style of a composer [4,8]. A Markov model can be estimated from a musical corpus, by counting transitions between successive elements. A random walk in a Markov model produces new sequences according to those transition probabilities, but does not, in general satisfy any other desirable property, such as unary constraints (specific values imposed at specific indexes of the sequences) or meter.

We have shown that the formulation of Markov processes as constraints opens the door to fine-grained control over generated sequences [11]. Additional properties such as unary constraints, meter and many others can then be enforced in polynomial time [9,15].

2.1 Enforcing Meter on Markov Sequences

METER is a global constraint that enforces a *metrical structure* on a sequence of temporal events [15]. Let X_1, \dots, X_n be a sequence of temporal events, such as notes, words, tasks, etc. Let $d(X_i)$ be the integer duration of the element assigned to X_i . We define $o(X_i)$, the *onset* or starting time of X_i in the sequence. It is equal to 0 if $i = 1$, or $\sum_{j=1}^{i-1} d(X_j)$ for higher indexes. For example, consider a sequence X_1, X_2, X_3, X_4 of integers [1, 2, 2, 1], where the duration of an element is its own value. The onset of X_1 is 0, the onset of X_2 is 1, the onset of X_3 is $d(X_1) + d(X_2) = 3$, and so on.

Meter Constraint: A METER constraint takes as parameters a *total duration* D , and a *predicate* $\pi(o, e)$, where o is an integer onset, and e an element from the domain of the X_i variables. METER holds on X_1, \dots, X_n if $\sum_{i=1}^n d(X_i) = D$, i.e. the sequence has a total duration of D , and $\pi(o(X_i), X_i)$ holds for every element X_i of the sequence, i.e. it is acceptable to start element X_i at time $o(X_i)$, for all elements of the sequence.

For example, suppose we want to create sequences of integers in $\{1, 2, 4\}$ (again, their duration is their own value), summing to 8, and sectioned into groups summing to 4. A solution is $[1, 2, 1, 2, 2]$: the total duration is 8, and it can be sectioned into $[1, 2, 1]$ and $[2, 2]$, each lasting 4. Another solution is $[4, 4]$. Conversely, $[2, 4, 2]$ is not a solution because it cannot be sectioned into subsequences summing to 4. We can encode this problem with METER, by setting $D = 8$, and defining $\pi(o, e)$ as follows:

$$\pi(o, e) \equiv \left(\left\lfloor \frac{o}{4} \right\rfloor = \left\lfloor \frac{o+e}{4} \right\rfloor \right) \vee \left(o+e = 4 \cdot \left(\left\lfloor \frac{o}{4} \right\rfloor + 1 \right) \right)$$

Intuitively, the predicate accepts value e at onset o only if e starts and ends in the same section (first case of the disjunction), or if e ends exactly at the start of the next section (second case). In general, sequences summing to 8 can involve a varying number of elements. In order to encode such sequences with a fixed number of variables, we choose a length sufficient for the longest sequence of total duration D , and we introduce a dummy element of zero duration, hereafter called *padding element*, used to fill the remainder of a sequence when the target duration is reached with fewer elements. In our example, we need at most 8 variables, to represent the sequence $[1, 1, 1, 1, 1, 1, 1, 1]$. The previous solutions are encoded as $[1, 2, 1, 2, 2, 0, 0, 0]$ and $[4, 4, 0, 0, 0, 0, 0, 0]$, respectively. In order to restrict the padding element to the end of the sequence, we need the predicate to also satisfy the condition $(o = D) \Rightarrow (e = 0)$.

We illustrate METER with “Frère Jacques”, a French nursery rhyme, also known in English as “Brother John”. We show the first 4 bars of this melody on Figure 1. This melody satisfies the following metrical constraints: its total duration, determined by the number of bars and the time signature, is equal to 16 beats (4 bars of 4 beats each), and notes do not cross bar lines.



Fig. 1. The first 4 bars of the Frère Jacques melody

In [15], we showed how to propagate METER as a global constraint in a CSP. However, an important observation underlying this work is that METER can also be formulated as a REGULAR constraint. As a result, we can apply the technique described in the next section to correctly sample metrically constrained Markov sequences, a novel result in this paper.

2.2 Markov models and Regular constraints

Recently, we generalised Markov constraints to REGULAR constraints [13], specifying that a Markov sequence X_1, \dots, X_n should additionally form a word from a *regular language* $\mathcal{L}(\mathcal{A})$, recognised by an imposed *finite-state automaton* \mathcal{A} . We use a factor-graph based model to encode this constrained Markov model, and use *belief propagation* to sample, with unbiased probabilities, Markov sequences satisfying REGULAR, in polynomial time [12]. Belief propagation generalises constraint propagation, where instead of propagating information on value consistency, we propagate probabilities associated with values.

A Markov model is a stochastic process, where the probability for state X_i , a random variable, depends only on the last state X_{i-1} . Each random variable X_i takes values amongst an *alphabet*, denoted \mathcal{X} . A Markov model produces sequence X_1, \dots, X_n with probability $P(X_1) \times P(X_2|X_1) \times \dots \times P(X_n|X_{n-1})$. Given additional unary constraints $P_i(X_i)$ and a REGULAR constraint specified by an automaton \mathcal{A} , the problem of sampling a Markov sequence subject to REGULAR is defined as the problem of sampling from the following distribution:

$$p_{target}(X_1, \dots, X_n) \propto \begin{cases} \prod_{i=2}^n P(X_i|X_{i-1}) \times \prod_{i=1}^n P_i(X_i) & \text{if } X_1 \dots X_n \in \mathcal{L}(\mathcal{A}) \\ 0 & \text{otherwise} \end{cases}$$

The symbol \propto (“proportional to”) indicates that the equality holds after normalisation. The first case indicates that the regular constraint holds (i.e. the sequence belongs to the specified language), and in the expression, the *unary factors* $P_i(X_i)$ are distributions that generalise unary constraints on the variables X_i . A unary factor merely biases the probability of the overall sequence, but does not necessarily correspond to the marginal distribution on X_i of the resulting distribution. In order to sample p_{target} , we reformulate it into a distribution p_{reg} of Y_1, \dots, Y_n , where the new Y_i variables take values (e, q) , where $e \in \mathcal{X}$ is a state of the Markov chain, and q is a state of the automaton \mathcal{A} that defines the REGULAR constraint. Sampling p_{target} is equivalent to sampling p_{reg} , and projecting each resulting sequence $(e_1, q_1), \dots, (e_n, q_n)$ to e_1, \dots, e_n . We show in [12] that p_{reg} can be represented as a tree-structured factor-graph, and therefore that we can use belief propagation to sample p_{reg} in polynomial time. The time complexity of this procedure is in the size of the alphabet of Y_i times the length of the sequence, i.e. $O(|\mathcal{X}| \cdot |Q| \cdot n)$.

2.3 Sampling Metrically Constrained Markov Sequences

In order to sample metrically constrained Markov sequences, we need to build the METER automaton $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ as follows:

- Q is the set of states: for each possible temporal position i , from 0 to the target duration D , we build a state q_i ;
- q_0 is the initial state, corresponding to temporal position 0;

- F is the subset of Q of accepting states: it contains only the state q_D corresponding to the target duration D ;
- Σ is the alphabet of the automaton, and contains the values e in the domains of variables X_i (i.e. musical events);
- δ is the transition function, mapping a state from Q and a symbol from Σ to a destination state: for a state q_o , corresponding to temporal position o , and an element e , we reach state q_d (i.e. $\delta(q_o, e) = q_d$), corresponding to temporal position d , iff $d = o + d(e)$ and $\pi(o, e)$ holds.

Figure 2 shows the METER automaton recognising all two-bar sequences we can build using the notes from Figure 1, and subject to the same metrical constraint. Since METER expects integer durations, durations are rescaled to match integer values, i.e. quarter notes have a duration of 1, half notes have a duration of 2, bars have a duration of 4 and the full melody has a duration of 8.

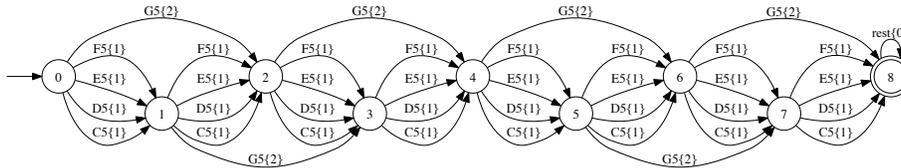


Fig. 2. The automaton accepting 2-bar melodies with the five notes from the melody on Figure 1, with meter. Discretised durations are shown between curly braces.

The procedure for sampling a metrically constrained Markov sequence is quadratic in the total duration D : the time complexity for sampling a Markov sequence subject to REGULAR is $O(|\mathcal{X}| \cdot |Q| \cdot n)$, as mentioned in Section 2.2. We need at most D variables to represent a sequence of total duration D (if the smallest duration is 1), and since each state of the METER automaton corresponds to a temporal position bounded by D , we have $|Q| = D + 1$, and therefore the overall time complexity is $O(|\mathcal{X}| \cdot D^2)$.

3 A Two-Voice Statistical Model of Lead Sheets

FlowComposer is based on a two-voice model of lead sheets, which captures stylistic information concerning the melody, the harmony, and the interaction between harmony and melody (see Figure 3). The chord model and the melody models are both based on a representation of music meter as a regular constraint as described in the preceding section.

Lead sheets are generated with the following procedure:

1. Generate a chord sequence by sampling the Markov+Meter model on chords, taking into account imposed sections of melodies as harmonic constraints.

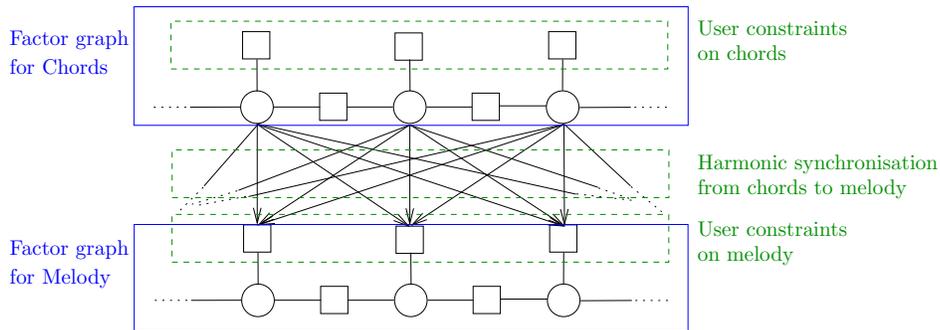


Fig. 3. The two-voice model for lead sheet generation

Additionally, when sampling, instead of drawing chords with their exact marginal distribution, we use a *variable order heuristic* that favours the chords that tend to replicate longer chord sequences from the corpus [2]. In practice, for each context size, we compute the entropy of the distribution on the chords that the context allows, then we choose a context length randomly, with a probability proportional to this entropy. This increases the impression of style imitation for the chord sequence, while avoiding risk of downright plagiarism, since orders with a low entropy are effectively discarded.

2. Generate a melody by sampling the Markov+Meter model on melody, imposing the chord sequence generated in the first step as a *harmonic constraint*. Here we do not use variable order, since each chord typically covers many notes, and the harmonic constraint introduces an amount of higher order correlation between all notes under a certain chord.

3.1 Markov+Meter Model for Chord Sequence

We generate a Markov sequence X_1, \dots, X_n , where X_i is assigned a chord, represented by a chord label and an integer duration. We train an order 1 Markov model on the corpus represented as sequences of chords. We use METER to impose a total duration equal to the duration of the lead sheet to compose, and to forbid chords to cross bar lines.

3.2 Markov+Meter Model for Melody

We generate a Markov sequence X_1, \dots, X_n , where X_i contains a note, represented by its MIDI pitch and integer duration. We obtain integer durations by multiplying all fractional durations with a fixed rescaling factor, equal to the least common multiple of the denominator of all possible fractional durations. This ensures that durations are integer, while maintaining proportions. We train an order 1 Markov model on the corpus represented as sequences of notes. We use METER to impose a total duration equal to that of the lead sheet to compose, multiplied by the rescaling factor, and to forbid notes to cross bar lines.

3.3 Enforcing Harmonic Synchronisation

To generate convincing lead sheets, our model also captures interactions between the note and chord models, in a way that is stylistically consistent with the chosen corpus. Such interactions can be represented in our two-voice model, by exploiting the structure of the factor graphs. We first build a *harmonic model* representing these relations, and then use it to bias sampling.

The Harmonic Model We define a *harmonic model*, which gives the probability $p_h(n|Ch)$ of placing note n under chord Ch , trained on the corpus. As a consequence, every chord defines a distribution on notes, and this distribution is fully parameterised by the chord label Ch . In order to decrease the amount of data needed to train this model, we adopt a more abstract representation and chords are reduced to their structure alone (for example Fm7 is represented by m7), and we ignore the octave and the duration of a note (for example, A5{2}, the A of the fifth octave, of duration 2, is represented only by A). Technically, for a given observation, we transpose the observed chord to a chord rooted in C with the same structure, and we transpose the observed note accordingly by the same amount of semitones. For example, an observation of a note A5{2} under a chord F m7 is abstracted as E under m7, since there are five semitones between C and F (the chord roots), and equally between E and A (the notes).

Enforcing Stochastic Temporal Constraints We showed in Section 2.3 how to sample METER. We now show that we can define constraints holding on elements specified by their temporal position, rather than by their index in the sequence, a novel result in this paper. We exploit the particular semantics of the METER automaton, i.e. states correspond to temporal positions. Temporal constraints are defined by generalising the METER predicate $\pi(o, e)$ to a *stochastic predicate* $p_\pi(e|o)$. The stochastic predicate defines the probability of placing event e at temporal position o . We then define $\pi(o, e) \equiv (p_\pi(e|o) > 0)$.

We enforce p_π by adding a unary factor in the p_{reg} model, for every Y_i . We recall that the variables Y_i of p_{reg} take values of the form (e, q) , with e a value in the alphabet of the Markov chain, and q a state of the automaton. By specifying a unary factor on all variables Y_i , we can bias the probability of an element e appearing with a state q , i.e. at a particular temporal position. The unary factor p_i applied to each Y_i is defined as follows: $p_i(e, q_d) \propto p_\pi(e|o) \cdot p(o)$, where q_d is the state of the Meter automaton corresponding to temporal position $d = o + d(e)$. The probability $p(o)$ gives the probability that o is the start time of an element. We assume it is uniform, but we can also learn this probability from the corpus.

Harmonic Constraints for Melody Given a chord sequence, we bias the generation of a melody to comply harmonically with the chords. We define the stochastic predicate $p_\pi(n|o) \propto p_h(n|Ch)$, where n is a note, and Ch is the chord occurring at temporal position o .

Harmonic Constraints for Chords Inferring chord labels from unlabelled melodies has been addressed previously, e.g. using Bayesian inference [14]. In our case, we need to bias the generation of a chord sequence to comply harmonically with the melody. We use a log-likelihood based approach assuming that notes are independent observations and follow the distribution given by p_h . Let us assume that we want to compute the probability $p_\pi(Ch|o)$ of placing chord Ch at temporal position o . Let n_1, \dots, n_p be the notes of the melody that occur between temporal positions o and $o+d(Ch)$. The average log-likelihood of chord Ch given notes n_1, \dots, n_p is:

$$\hat{l}(Ch; n_1, \dots, n_p) = \frac{1}{p} \sum_{i=1}^p \log p_h(n_i|Ch)$$

In order to introduce variety in the generated lead sheets, we do not choose the chord with the maximum \hat{l} , but rather use this value to define its probability, so that more likely chords are closer to the observed distribution of notes. In practice, we set $p_\pi(Ch|o) \propto \exp\{\hat{l}(Ch; n_1, \dots, n_p)\}$.

Releasing Harmonic Pressure The approach we described is often too strict in practice, and, sometimes, we want to relax harmonic pressure. To this end, we propose two strategies. First, we introduce a parameter called *harmonic conformance* that specifies how biased or uniform the harmonic model $p_h(n|Ch)$ should be. The harmonic conformance is a factor α ranging between 0 and 1, and we define a new, relaxed, harmonic model as follow:

$$p'_h(n|Ch) \propto \alpha \cdot p_h(n|Ch) + (1 - \alpha) \cdot p_{uniform}$$

A value of 1 implies strict conformance, a value of 0 results in a uniform distribution, i.e. no harmonic bias at all. This value can be set by the user in the GUI in the form of a slider.

Second, we choose to impose harmony on beats only, as a way of approximating the detection of passing notes, i.e. notes on which harmony is typically less important. In practice, the stochastic predicates $p_\pi(Ch|o)$ and $p_\pi(n|o)$, for chords and notes, are uniform if o is not the start of a beat.

4 Applications

In this section, we describe FlowComposer, a web service for the composition of lead sheets based on the algorithms described in Section 3. The web service was implemented using Java. The models for chords, notes and harmony, and the belief propagation procedure to sample those models, have been implemented as an in-house solver. FlowComposer is both an autonomous generator and an interactive music composition application integrated in an ecosystem of online tools for the creation and manipulation of lead sheets. Lead sheets are represented as JSON objects stored in a MongoDB database with more than 12,000 songs in

various styles [10]. A graphical lead sheet viewer and editor is implemented as a JavaScript library [7] running in the client web browser. Other services are provided, such as MIDI and audio rendering tools, harmonic and pattern analysis. User sessions and persistence is managed on the server side by a PHP module.

The database covers several genres of popular music: jazz, pop, rock, and Brazilian music, by hundreds of famous composers. A style is defined as a corpus, i.e. a selection of songs from the database, for instance, all the songs by a given composer, in a given genre, or any manual selection. There are 157 types of chords used at least in one song of the database. Among them, 37 chord types have more than 1000 occurrences, e.g., major chords, diminished seventh; 43 occur between 100 and 1000 times, e.g., m69 chords; 61 are used between 20 and 100 times, e.g., m7sus4; and 126 occur fewer than 20 times, e.g., M7#9.

We show three typical scenarios of the general resolution procedure explained in the preceding section. Section 4.1, describes how FlowComposer can be used as an autonomous lead sheet generator. Section 4.2, describes how FlowComposer can harmonise imposed melodies. Section 4.3, describes how FlowComposer may be used as an online interactive lead sheet composition application.

4.1 Autonomous Generation

The lead sheet generation algorithm may be used to generate lead sheets from scratch, in the style of a given corpus. In this scenario, we parse each song in the training corpus, defining the style of the generated lead sheet. Then, we build and train the Markov+Meter models for the chord and melody generations and the harmonic model. The generation of the lead sheet follows the procedure specified in the preceding section. Note that in the first step, the chord sequence is generated with no harmonic constraints since there is not melody yet. Figure 4 shows an 8-bar lead sheet generated by this procedure in the style of Bill Evans.

Fig. 4. An 8-bar lead sheet generated in the style of Bill Evans

The length of the lead sheet to generate and the training corpus are two input parameters of the generation algorithm. The generation is also influenced by other parameters, such as the number of chord changes, the number of notes, or the harmonic conformance. The ‘Number of chord changes’ and ‘Number of notes’ are set to match the average number of notes and chord changes in the corpus. The padding strategy, see Section 2.3, allow the system to generate sequences with *approximately* the specified numbers of notes and chord changes. The harmonic conformance α is set by default to its maximum value.

Performance is reported on Table 1. The **Parsing** time is linear in the number of notes and chords in the corpus. In practice, this is a linear function of the total number of bars in the corpus. The **Training** time indicates the time needed to build and train the Markov+Meter models and the harmonic models and to initialise the belief propagation algorithms. The time reported in column **Next sol.** is less than the whole training time as the chord model is not retrained. The

Corpus	Size	Parsing	Length	Training	Next sol.	Model size
ASW	429 songs 630 chords 356 notes	3"480	4 bars	4"	1"7	731,468
			8 bars	9"	5"3	2,922,436
			12 bars	18"	12"5	6,465,202
			16 bars	30"	22"	11,509,685
Beatles	45 songs 134 chords 199 notes	803 ms	4 bars	600 ms	600 ms	245,948
			8 bars	1"75	1"4	984,099
			12 bars	3"9	3"	2,179,359
			16 bars	6"6	5"	3,939,752
			32 bars	27"	24"	15,668,776

Table 1. Performance of the algorithm for the generation of lead sheet for various lengths and two corpora; ASW stands for American Songwriters, a corpus with 429 songs by composers such as Richard Rodgers, Lorenz Hart, or Irving Berlin

generation times increase quadratically with the length of the generated lead sheet. This experimental observation is consistent with the expected complexity of the algorithm (see Section 2.3) and is reflected in the size of the models.

The reported performance shows that generation becomes slow for lead sheets longer than 16 bars, especially with a large corpus, such as American Songwriters. Parsing the corpus uses non-optimised code, and its performance will be reduced by a substantial amount in the next version. The time needed to train the model and to find solutions may also be reduced by discarding very small probabilities in the model.

In general, music composed using statistical-based approaches is locally consistent but lacks a *sense of direction*, or *global structure* [6]. Consequently, purely autonomous generation is usually used to produce short musical sequences to be used as fragments in a longer piece. We observed that composers using the system rarely generate sequences exceeding 8 bars (see 4.3). The time taken to generate long lead sheets is therefore not a strong limitation of the system. The automatic generation of interesting long sequences requires models of large-scale musical structure including repetition of patterns, variations, and sections.

4.2 Harmonisation

FlowComposer may be used to *harmonise*, i.e. infer chord labels for a given melody in an imposed style. We illustrate such style-based harmonization by using FlowComposer to reharmonize “Yesterday”, by the Beatles, in four styles: the Beatles themselves, Cole Porter, Michel Legrand, and Bill Evans.

The image shows a musical score for the first 15 bars of "Yesterday" in 4/4 time. The melody is written on a single staff in treble clef. Below the melody, there are three lines of chord labels. The first line contains: F, Em7, A7, Dm7, Dm7, Bb. The second line contains: F, C, Dm7, G7, Bb, F, Em7, A7. The third line contains: Dm, C, Bb, Dm, Gm, C7, F, Em7, A7. The fourth line contains: Dm, C, Bb, Dm, Gm, C7, F.

Fig. 5. The lead sheet of the first 15 bars of “Yesterday” with the original chord labels

Table 2 shows the number of songs in the corpus corresponding to each of these styles. Note that in the case of the Beatles, we did not include “Yesterday” in the corpus. The first 15 bars of the original harmonisation is shown on Figure 5.

The image shows a musical score for the first 15 bars of "Yesterday" in 4/4 time, featuring an alternative harmonisation. The melody is on a single staff in treble clef. Below the melody, there are three lines of chord labels. The first line contains: Bb, Dm, A7, Dm, Gm. The second line contains: Dm, A7, Dm, A7. The third line contains: Dm, Gm, A7, Dm, A7. The fourth line contains: Dm, C, Bb, F, C, Bb.

Fig. 6. The lead sheet of the first 15 bars of Yesterday with an alternative harmonisation generated by FlowComposer in the style of the Beatles

The re-harmonisation in the style of the Beatles (Figure 6) uses many chords appearing in the original lead sheet, e.g., Dm, Gm, Bb. Some chords are simple harmonic substitutions, such as the Gm in place of Bb on bar 4. The A7 on bar 7, which is not a substitution of the original harmony Bb-F, is quite surprising given the F in the melody. However such an augmented fifth is not unusual in the Beatles, and occurs for example in songs “I Want to Tell You” and “I’m Only Sleeping”, in this latter case with the same resolution to a Dm chord. The progression in bars 13 to 14 is equal to the original. Overall, this re-harmonisation is probably less interesting than the original one, but is new, valid, and can be considered as in the style of the Beatles.

The harmonisation in Cole Porter’s style (Figure 7) uses an E \flat 7 (half diminished seventh) and an F \sharp o7 (diminished seventh) chords. Cole Porter is the composer who uses these chord types the most in the database. The progression DmM7-Dm-G7 appears identically in Cole Porter song “Do I Love You”.

The use of a suspended 7th chord, E7sus in bar 10 in the example in Figure 8 is typical of Michel Legrand: he is, in our database, the composer using them the most. The progression B half-diminished 7, B \flat 7, followed by E7sus actually occurs in “The Easy Way”, or, transposed, in “Papa Can You Hear Me?”. The E \flat 7 chord in bar 12 is a tritone substitution of the original A7 chord.

Fig. 7. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Cole Porter

Fig. 8. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Michel Legrand

Fig. 9. The lead sheet of the first 15 bars of Yesterday with a harmonisation generated in the style of the Bill Evans

In the re-harmonisation in the style of Bill Evans (Figure 9), the opening transition from Eb69 to A+7 (augmented 7th chord) appears in “Yet Neer Broken”. Bill Evans is also the composer using 69 and 7b9sus chords the most.

We encourage the reader to listen to these examples³ to get a feel of the stylistic differences of the various harmonisations of the system.

Execution times are reported on Table 2. Most of the time is spent training the models, and is done only once, thanks to the persistence of user sessions. The third column reports the total time to get the first solution. Once the model is trained, it may be sampled virtually instantaneously to produce many representative solutions.

³ All examples are available online at: <http://www.flow-machines.com/generation-of-lead-sheets-with-flowcomposer>

Style	Corpus size	Parsing time	Training time	Time next sol.	Total time
The Beatles	45 songs	85 ms	830 ms	15 ms	1"030
Cole Porter	70 songs	203 ms	1"568	15 ms	1"786
Michel Legrand	62 songs	181 ms	2"339	15 ms	2"535
Bill Evans	87 songs	132 ms	2"700	15 ms	2"847

Table 2. Corpus size for several styles and execution times to harmonise the first 15 bars of “Yesterday”. The total time is the time needed to parse the corpus and to train the model. Solutions are obtained virtually instantaneously by sampling the model

4.3 Interactive Composition

FlowComposer is not only an automatic generator but also a fully-fledged interactive composition tool for professional composers who use it as an active, create software collaborator. FlowComposer is integrated with an online lead sheet editor so that the system never gets in the way of the user’s intentions and composition habits.

Benoit Carre (with FC)

Am7 D7 Gm7 C7

Fm7 Bb7 Eb6 Em7 Am7 D7 Bm7 E7

Fig. 10. Selection of two bars in the course of the composition of the song of Figure 11; the two selected bars will be replaced by new musical material generated by FlowComposer

The general idea is that the user is responsible for creating the structure of the lead sheet and FlowComposer is used to generate music for user selected parts. The general structure of a lead sheet consists of a sequence of sections, which are played in a sequence with optional repetitions. A typical structure is AABA followed by a Coda, such as in “Yesterday”. The editor allows the user to select contiguous fragments of the lead sheet and FlowComposer generates new music for this fragment; the fragments may contain chord labels, notes, or both (see Figure 10).

Benoit Carre (with FC)

Am7 D7 Gm7 C7 Fm7 Bb7 Eb6 Em7 Am7 D7 Bm7 E7

Fig. 11. A song composed by French pop song writer Benoît Carré with FlowComposer

The selection is not considered as an isolated musical fragment, but rather as belonging to the context of the current lead sheet. The non-selected parts are fixed and imposed as constraints to the system. Technically, the model covers the user selection extended to the music immediately before and after, to ensure that transitions between the selection and the surrounding musical context are in the chosen style.

The front end of the application provides the user with control on the generation parameters, which can be changed at any time: the training corpus, the number of notes or chord changes, and the harmonic conformance α .

The system updates the model with the music entered in the lead sheet being composed. An additional control, called ‘inspiration’, is used to control the relative weight in the model of the training corpus and of the current composition. This control is typically used to put the emphasis on the current composition when the lead sheet contains already several bars of music to increase the probability that the music generated by the system will repeat some fragments present in the non-selected parts of the score.

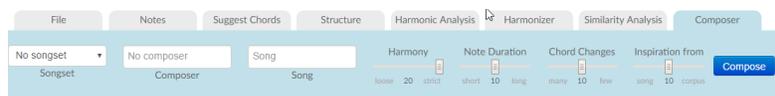


Fig. 12. The control panel with fields to select composition style and sliders to set harmonisation conformance, *inspiration*, and average *note duration* and *chord changes*

French songwriter Benoît Carré is using FlowComposer for a forthcoming pop music album (see excerpts in Figure 11). FlowComposer was also used by professional composers Nathan Taylor and Benjamin Till to compose three songs for the “Beyond the Fence” musical. For example, the chorus and some chord progression of “Scratch That Itch” come from FlowComposer. The final score has then been reworked by the composers. “Beyond the Fence” was the first musical ever created by software (not only songs, but also lyrics and the story line itself), and was performed at the Arts Theatre in London in February 2016 (see [5] for discussions about this pioneering experiment). Figure 13 shows the beginning of one of the songs in its final version.

5 Conclusion

We presented FlowComposer, an online application for assisted music composition. The application enables users to compose musical lead sheets from partial information, and uses a generation algorithm to fill in the missing parts in the style of a chosen composer. The generation algorithm exploits a representation of meter as a REGULAR constraint which enables efficient sampling. Lead sheets are represented as a couple of meter-constrained Markov chains, synchronised through a likelihood function learnt from the selected corpus of lead sheets.

Benjamin Till / Nathan Taylor /
Computational Composer
(CERIDWEN:)

12: Scratch That Itch From Beyond the Fence

The image shows a musical score for a song. The top staff is a vocal line in 8/8 time, starting with a treble clef and a key signature of one flat. The lyrics are: "(freely) Now there's one thing I want to make clear, there's wo-men of all sorts by here. All". The vocal line has dynamics markings of *mp* and *f*. The bottom staff is a piano accompaniment in 8/8 time, starting with a bass clef and a key signature of one flat. It features a steady bass line and chords in the right hand, with dynamics markings of *f* and *mp*.

Fig. 13. Final sheet of a song composed by Nathan Taylor and Benjamin Till using FlowComposer, as part of the “Beyond the Fence” musical

The image shows a lead sheet for a jazz piece in 4/4 time. The melody is written in the treble clef with a key signature of one flat. The chords are: F7 (1), Bbm7 (2), Eb7 (3), Eo7 (4), Bb7 (5), EbM7 (6), D7alt (7), Eb7 #11 (8), E7 #9 (9), and Am9 (10). The melody consists of eighth and quarter notes, with some triplet markings.

Fig. 14. A lead sheet composed by the authors of this article with FlowComposer, in the style of Miles Davis, and evaluated (informally) by fellow jazz musicians as particularly good

We have shown several typical uses of FlowComposer to generate lead sheets from scratch, from partial information, or to reharmonize existing melodies¹. More work is being done to improve the user experience, notably by storing statistical models to avoid on-the-fly training, which calls for an incremental updating of these models.

FlowComposer can be seen as a successful example of an online application mixing statistical models with hard constraints. FlowComposer integrates a number of recent results in constraint programming and sampling, and has been used successfully in several professional music projects. We believe such a tool offers unprecedented value to users wanting meaningful assistance in composition, thanks to the powerful underlying style modeling approach. We also believe that online applications mixing statistical models and hard constraints will be a very active thread of development for CP in the near future.

Acknowledgment

This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Union’s Sev-

¹ All the examples presented in this article are available online at <http://www.flow-machines.com/generation-of-lead-sheets-with-flowcomposer>

enth Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 291156. We thank Benoit Carré and the team of the musical *Beyond the Fence* for their insightful comments in using the system. We thank Fiammetta Ghedini for creating the associated website with audio and video examples.

References

1. Anders, T., Miranda, E.R.: Constraint programming systems for modeling music theories and composition. *ACM Comput. Surv.* 43(4), 30:1–30:38 (Oct 2011)
2. Begleiter, R., El-Yaniv, R., Yona, G.: On prediction using variable order markov models. *J. Artif. Intell. Res. (JAIR)* 22, 385–421 (2004)
3. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. pp. 1159–1166 (2012)
4. Brooks, F.P., Hopkins, A., Neumann, P.G., Wright, W.: An Experiment in Musical Composition. *Electronic Computers, IRE Transactions on* (3), 175–182 (1957)
5. Colton, S., Llano, M.T., Hepworth, R., Charnley, J., Gale, C.V., Baron, A., Pachet, F., Roy, P., Gervás, P., Collins, N., Sturm, B., Weyde, T., Wolff, D., Lloyd, J.: The Beyond The Fence Musical and Computer Says Show Documentary. In: 7th International Conference on Computational Creativity (ICCC 2016). Paris (France) (June 2016)
6. Eck, D., Schmidhuber, J.: Learning the long-term structure of the blues. *Proceedings of Int. Conf. on Artificial Neural Networks—ICANN 2002* pp. 284–289 (2002)
7. Martín, D., Neullas, T., Pachet, F.: LeadsheetJS: A Javascript Library for Online Lead Sheet Editing. In: 1st International Conference on Technologies for Music Notation and Representation (TENOR 2015). Paris (France) (May 2015)
8. Nierhaus, G.: *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media (2009)
9. Pachet, F., Roy, P., Barbieri, G.: Finite-length Markov processes with constraints. In: *IJCAI*. pp. 635–642 (2011)
10. Pachet, F., Suzda, J., Martinez, D.: A comprehensive online database of machine-readable lead-sheets for jazz standards. In: de Souza Britto Jr., A., Gouyon, F., Dixon, S. (eds.) *ISMIR*. pp. 275–280 (2013)
11. Pachet, F.: Flow-Machines: CP techniques to model style in music and text. *ACP (Association for Constraint Programming)* (2015), <http://www.a4cp.org/node/1066>
12. Papadopoulos, A., Pachet, F., Roy, P., Sakellariou, J.: Exact sampling for regular and markov constraints with belief propagation. In: *CP. Lecture Notes in Computer Science*, vol. 9255, pp. 341–350. Springer (2015)
13. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) *CP. Lecture Notes in Computer Science*, vol. 3258, pp. 482–495. Springer (2004)
14. Rhodes, C., Lewis, D., Müllensiefen, D.: Bayesian model selection for harmonic labelling. In: Klouche, T., Noll, T. (eds.) *Mathematics and Computation in Music*, pp. 107–116. Springer-Verlag, Berlin and Heidelberg (2009)
15. Roy, P., Pachet, F.: Enforcing Meter in Finite-Length Markov Sequences. In: desJardins, M., Littman, M.L. (eds.) *AAAI*. AAAI Press (2013)