Binary Relational Constraints in Non-Homogeneous Markov Models

Anonymized

Content areas: Constraint Satisfaction, Global Constraints, Markov Decision Processes, Constraint Applications, Probabilistic Machine Learning, Constraints and Data Mining / Machine Learning

Abstract

Enforcing structure while still affording autonomous expression is a challenge for many probabilistic sequential data models. We present a method for modeling and generating sequences with structural patterns that are defined as a function of binary relational constraints relating sequence elements. We present an deterministic finite automaton (DFA) to build an automaton which accepts sequences with the desired structural patterns. We also present an algorithm which combines an arbitrary DFA with a Markov model to create a non-homogeneous Markov model which samples sequences belonging to the language of the automaton with exact probabilities according to the Markov model.

1 Introduction

Structure is often observed or created in sequential data as a result of relationships between elements at potentially distant positions. Protein and RNA folding depend on bonds formed by the pairing of amino acids and ribonucleotide bases respectively at various intervals. In natural language subject-verb and pronoun-antecedent agreements depend on the correct pairing of potentially distant words. Other examples of sequencing problems with relational structures might include the rostering and car sequencing problems.

In this paper we are interested in the problem of probabilistically modeling sequences under *relational constraints*, that is sequences in which values at distant positions within the sequence are constrained to relate according to an arbitrary predefined relation (e.g., matching, bonding, agreement, etc.). Owing to their stochasticity, Markovian sequential data models struggle to account for structure beyond the Markov window which results in a limited ability to generate or classify sequences in problem domains characterized by such structures.

Much work has been done to impose structure in Markov processes by combining such processes with constraint programming (CP) and constrained probabilistic modeling techniques. Non-homogeneous Markov models (NHMMs) [Pachet *et al.*, 2011] generate finite-length sequences which adhere to a set of unary constraints with probabilities determined from a Markov process.

Recent work has focused on statistical models which obey *regular constraints*. A regular constraint on a sequence of finite-domain variables is a constraint that requires that the corresponding sequence of values taken by these variables belongs to a given regular language [Pesant, 2004]. These models are convenient because they can effect structure without imposing unary constraints at fixed positions. Papadopoulos *et al.* demonstrate how a regular constraint in the form of a deterministic finite automaton (DFA) can be combined with probabilistic Markov model to create a model that generates sequences belonging to the language of the automaton with probability approximately equal to the Markov probability distribution [2014]. Their follow-up work devises a belief propagation model in the form of a factor graph capable of exact sampling of sequences under arbitrary regular and Markov constraints [Papadopoulos *et al.*, 2015].

Relational constraints can often not be characterized using regular languages (e.g., $A = \{ww|w \in a, b*\}$ or $A = \{ww|w \in \Sigma\}$ where Σ is in infinite domain). However, under the assumptions of finite-domain variables and finite-length sequences, relational constraints can be modeled using regular constraints. This suggests that an approach similar to that of Papadopoulos *et al.* [2015] may be suitable for imposing relational constraints given a method for proper DFA construction.

In what follows we detail the formulation of a regular constraint (in the form of a DFA) from an arbitrary set of relational constraints. We also detail a method for exact probabilistic sampling of sequences from the language an arbitrary DFA which uses NHMMs and demonstrate its improved efficiency over factor graphs for sampling large batches of constrained sequences.

2 Related Work

Several works have been presented which address the problem of imposing relational constraints.

Barbieri *et al.* demonstrate examples of how unary constraints can be used to imitate relational rhyming constraints [2012]. This approach is not suitable for most relational constraint applications as it relies on separately constraining values at independent positions rather than constraining based on the comparison of values.

? demonstrate that arbitrary relational constraints can be effected in d-order NHMMs but only when matching positions are within the Markov window (i.e., at most d sequence positions apart) [**?**]. The method we present has no limitation on the distance between relationally constrained positions.

Roy *et al.* define an Allen constraint as a global constraint relating indices with temporal positions [2016]. This work is designed to primarily address contiguous temporal sequences in which temporal positions and sequence indices are not directly related. Their interest is primarily in sub-sequence *equality* constraints. The algorithm for how these constraints are effectuated is not presented.

Collins and Laney create long-term repetitive and phrasal structure in music using a "copy-paste" approach to enforce binary matching of full sequences at intervals [2017]. This approach is fundamentally non-Markovian at splice sites causing unnatural transitions at splice sites. Pachet *et al.* present a similar "copy-paste" method for creating structured music lead sheets, but use a controlled variation mechanism to ensure that copies are contextually situated to satisfy Markovian properties [2017].

To our knowledge, the method described below is novel in its strict use of Markovian processes (of arbitrary order) for generation while simultaneously enforcing arbitrary relational constraints at arbitrary distances.

3 A DFA for Relational Constraints

A *binary relation* ρ on a set Σ is defined as a set of ordered pairs of elements of Σ. Examples include the set of pairs of amino acids (x, y) such that x forms a bond with y; or the set of ordered pairs of words (x, y) such that x is an antecedent and y is a pronoun that agrees with x.

In generating a sequence of random variables $X = (X_1, ..., X_n)$ we define a *relational constraint* (X_i, X_j, ρ) for an arbitrary binary relation ρ to mean that the values x_i and x_j assigned to the variables X_i and X_j are constrained so as to ensure that $(x_i, x_j) \in \rho$. A partially instantiated relational constraint of the form (x_i, X_j, ρ) or (X_i, x_j, ρ) represents a unary constraint in which X_i or X_j has been further constrained to the values x_i or x_j respectively.

A DFA, or simply an automaton, is defined as a quintuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ in which:

- Q is a finite set of states;
- Σ is a finite set of symbols termed the alphabet;
- $q_0 \in Q$ is the initial or start state of the automaton;
- δ is the transition function $Q \times \Sigma \to Q$, mapping a state to a successor state for a given symbol;
- $F \subseteq Q$ is the set of final or accept states.

A sequence $s = \{s_1, ..., s_n\}$ is *accepted* by $\mathcal{A}$ iff there exists a sequence $q_0, ..., q_n$ of states such that $\forall i, 1 \leq i \leq n$, $\delta(q_{i-1}, s_i) = q_i$ and $q_n \in F$. The language $\mathcal{L}(\mathcal{A})$ is the set of all sequences which $\mathcal{A}$ accepts.

A Markov model is a stochastic process where the probability of a sequence of random variables $X = (X_1, ..., X_n)$ is computed as $P(X) = P(X_1) \cdot P(X_2|X_1) \cdots P(X_n|X_{n-1})$. As such a Markov model consists of a set of initial probabilities $\mathcal{I}$ for each state and a set of transition probabilities $\mathcal{T}$ between states. Note that when combining a DFA and Markov model, the domain Σ for variables $X_1, ..., X_n$ is the same as the alphabet Σ for the DFA.

Given a set of binary relational constraints $\mathcal{M} = \{(X_i, X_j, \rho)\}$, a set of valid start symbols $I \subset \Sigma$, a set of valid transitions $T = \{s_i s_j | s_i \in \Sigma \text{ and } s_j \in \Sigma\}$, and a length n, Algorithm 1 creates a DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$ is the set of all sequences $s \in \Sigma^n$ such that $s_1 \in I$; $s_{i-1}s_i \in T$ for all $i, 1 < i \leq n$; and $(s_j, s_k) \in \rho$ for all relational constraints (X_j, X_k, ρ) in $\mathcal{M}$. Where I and T are derived from the sets $\mathcal{I}$ and $\mathcal{T}$ of a Markov model M, this means $p_M(s) > 0$.

The algorithm takes the general approach of building breadthwise a tree-like DFA where each layer (of edges) in the tree represents a position in the sequence to be gener-

Algorithm 1 RELATIONAL AUTOMATON

Data: n a sequence length
$\mathcal{M}$ a set of binary relational constraints
I a set of valid initial states
T a set of valid transitions
Result: $\mathcal{A}$ a DFA for $\mathcal{M}$

```
A ← ⟨Q, Σ, δ, q_0, F⟩
S ← {⟨∅, I, q_0⟩}
for i ← 1 to n − 1 do
    S' ← {}
    for ⟨C, V, q⟩ ∈ S do
        for v ∈ V do
            C' ← {(A, B, ρ) ∈ C | A ≠ X_i ∧ B ≠ X_i}
            for (X_j, X_k, ρ) ∈ M do
                if i = j and j < k then
                    C' ← C' ∪ (v, X_k, ρ)
                if i = k and k < j then
                    C' ← C' ∪ (X_j, v, ρ)
            V' ← ∅                          ▷ initialize possible next states
            for vv_2 ∈ T do                 ▷ filter by C'
                if ∀(v', X_{i+1}, ρ)∈C', (v', v_2) ∈ ρ then
                    if ∀(X_{i+1}, v', ρ)∈C', (v_2, v') ∈ ρ then
                        V' ← V' ∪ v_2
            if V' ≠ ∅ then                  ▷ if not a dead state
                if ∃q_i s.t. ⟨C', V', q_i⟩ ∈ S' then
                    q' ← q_i
                else
                    q' ← NewState(Q)
                    S' ← S' ∪ ⟨C', V', q'⟩
                δ(q, v) ← q'                ▷ extend path
    if S' = ∅ then
        return null                         ▷ unsatisfiable
    S ← S'
q_acc ← NewState(Q)                         ▷ define accepting state
F ← F ∪ q_acc
for ⟨C, V, q⟩ ∈ S do
    for v ∈ V do
        δ(q, v) ← q_acc
return A
```

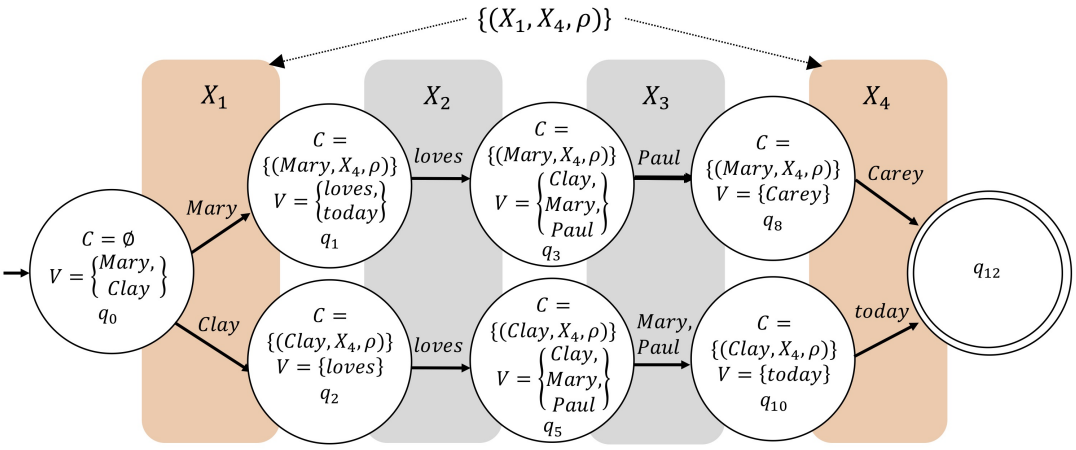

Figure 1: *A* RELATIONAL *automaton*. The result of Algorithm 1 on inputs $n = 4$; $\mathcal{M} = \{(X_1, X_4, \rho\}$ (where ρ represents the set of rhyming word pairs); $I = \{$*Mary*, *Clay*$\}$; and T derived from the non-zero transitions represented in the Markov model shown in Figure 2.

ated. Each state q_i is defined by a triple $< C, V, q_i >$ where $C = \{(X_i, X_j, \rho)\}$ is the set of relational constraints that are guaranteed from the state; $V = \{x | x \in \Sigma\}$ is the set of valid labels *allowed* by T for edges originating from the state (in practice dead states and paths may be pruned); and q_i is a reference to the state itself.

When expanding the path via a state q_i associated with sequence position i and triple $< C, V, q_i >$, a new path is considered for every label $v \in V$. In order to build the path, the algorithm first computes what would be the new triple $< C', V', q' >$ for the new state reached via v in order to check if such a state already exists. The new set C' inherits all constraints from C except those already satisfied at sequence position i and adds new constraints applying to position i in their partially instantiated form using v. The new set V' is populated with the set of all labels v_2 that are 1) valid transitions from v according to T and 2) satisfying values for any constraints in C' which apply to sequence position $i + 1$. In practice, V' can be further constrained by unary constraints or negated relational constraints (i.e., constraining to ensure $(x_i, x_j) \notin \rho$) to further prune undesirable sequences. Assuming V' is not empty (a sign of a dead state), we check for an existing state associated with sequence position $i + 1$ and triple $< C', V', q' >$ or create a new one which is then used to extend the path. In the last state layer there is a single accepting state to which all states in the previous state layer connect via all valid labels in their respective V sets.

Figure 1 shows the DFA built using Algorithm 1 with $n = 4$; $\mathcal{M} = \{(X_1, X_4, \rho_{rhyme}\}$ (where ρ_{rhyme} represents the set of rhyming word pairs); $I = \{$*Mary*, *Clay*$\}$; and T derived from the non-zero transitions represented in the Markov model shown in Figure 2.

The time and space requirements for the algorithm vary significantly depending on the inputs. Each constraint in $\mathcal{M}$ at a new position splits the path into a number of paths dependent on the number and distribution of transitions in T and the restrictiveness of the constraint's relation ρ. Divergent paths will reconverge once all overlapping constraints have been resolved. The number of paths and states can also be reduced by the addition of unary constraints, negated relational constraints, increasing the number of symbols per label (i.e., increase the Markov order), and reducing the size of the transition set T.

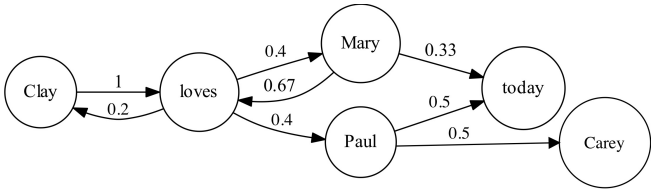

Figure 2: *A Markov model.*

One way to optimize the construction of a RELATIONAL automaton is to take advantage of binary relational constraints for which ρ is an *equivalence relation* (i.e., reflexive, symmetric, and transitive). In such cases many partially instantiated constraints become equivalent. For example, under the equivalence relation of rhyming (allowing for a word to rhyme with itself), the following partially instantiated binary relational constraints are equivalent: $(Mary, X_4, \rho), (Fairy, X_4, \rho), (Carry, X_4, \rho)$. These constraints can all be represented using a generalized constraint that uses the equivalence class $\mathcal{E}_{Mary}$ representing the set of all rhyming word pairs that rhyme with "Mary": $(\mathcal{E}_{Mary}, X_4, \rho)$. This facilitates the combining of states in the DFA.

4 Exact Sampling of Constrained Sequences

Given a RELATIONAL automaton $\mathcal{A}$, a Markov model M, a set of unary constraints C, and a length n we now turn attention to sampling sequences from the target distribution p^* defined as:

$$p^*(X_1, ..., X_n) \propto \begin{cases} p_M(X_1, ..., X_n) \cdot \prod_{i=1}^{n} p_C(X_i) & \text{if } X_1, ..., X_n \in \mathcal{L}(\mathcal{A}) \\ 0 & \text{otherwise} \end{cases}$$

Papadopoulos *et al.* demonstrate one mechanism for computing p^* that uses factor graphs with belief propagation for sampling sequences with exact probabilities according to the original distribution [2015]. In this latter solution a backward phase computes the impact on each sequential position i in the factor graph of the sub-factor graph representing all positions "to the right" of i. A forward phase computes the marginal distribution over values for each sequence position i given the partial instantiation for variables representing positions "to the left" of i and simultaneously samples a sequence. A potential drawback to this solution is that though the backward phase is completed once, the forward phase is repeated each time a sequence is sampled.

We present a novel algorithm (Algorithm 2) for sampling sequences under regular and Markov constraints with exact probability that uses a NHMM [Pachet *et al.*, 2011]. A NHMM $\mathcal{N}$ is a constrained probabilistic model constructed from a Markov model M, a set of unary constraints C, and a length n such that

$$p_{\mathcal{N}}(X_1, ..., X_n) \propto p_M(X_1, ..., X_n) \cdot \prod_{i=1}^{n} p_C(X_i)$$

Constructing a NHMM computes all marginal distributions ahead of time. This speeds up sampling time at the cost of an increased build time, making it a better choice for sampling large sets of sequences or online applications (see Figures 4 and 5). A detailed explanation of NHMMs and the method of their construction (referenced as "constructNHMM()" in Algorithm 2) is available from Pachet *et al.* [2011].

Given an automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, a Markov model $M = \{\mathcal{I}, \mathcal{T}\}$, a set of unary constraints $C = \{c_{X_i}\}$ (where c_{X_i} represents a unary constraint c applied to the random sequence variable X_i), and a length n, Algorithm 2 builds a new "state-sensitive" Markov model M' which incorporates the regular constraint represented by $\mathcal{A}$. Each Markov label in M' represents a label-state pair $< x, q >$ where $x \in \Sigma$ is a label of the original Markov model M and $q \in Q$ is a state of the automaton $\mathcal{A}$. Whereas with M we can directly sample values $x_1, ..., x_n$ for a sequence of random variables $X = X_1, ..., X_n$, in the "state-sensitive" model M' we first sample values $x'_1, ..., x'_n$ for a sequence of random label-state variables $X' = X'_1, ..., X'_n$. We obtain X through the assignment $X = x'_1.label, ..., x'_n.label$ where $x'_i.label$ is the value of the label in the label-state pair x'_i.

The set of initial probabilities $\mathcal{I}'$ of M' is defined as

$$\mathcal{I}'(< x, q >) \propto \begin{cases} \mathcal{I}(x) & \text{if } q = \delta(q_0, x) \\ 0 & \text{otherwise} \end{cases}$$

and the transition probability function $\mathcal{T}'$ of M' is defined as

$$\mathcal{T}'(< x', q' > | < x, q >) \propto \begin{cases} \mathcal{T}(x'|x) & \text{if } q' = \delta(q, x') \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 2 REGULAR NHMM

Data: $\mathcal{A} \leftarrow \langle Q, \Sigma, \delta, q_0, F \rangle$ an automaton
$M \leftarrow \{\mathcal{I}, \mathcal{T}\}$ a Markov model
$C \leftarrow \{c_{X_i}\}$ a set of unary constraints
n the finite length of the resulting model
Result: $\mathcal{N}$ a NHMM which samples sequences from $\mathcal{L}(\mathcal{A})$ with exact probability according to M

$M' \leftarrow \{\mathcal{I}', \mathcal{T}'\}$ ▷ Create new Markov model
for $(q, a) \in \delta$ s.t. $q = q_0$ **do**
  $\mathcal{I}'(< a, \delta((q, a)) >) \leftarrow \mathcal{I}(a)$ ▷ Add initial states
for $(q, a) \in \delta$ **do**
  for $a_2 \in \Sigma$ **do** ▷ Add transition probabilities
    $\mathcal{T}'(< a, q >, < a', \delta((q, a)) >) \leftarrow \mathcal{T}(a_1, a_2)$
$C' \leftarrow \{\}$ ▷ Create new set of control constraints
for $c_{X_i} \in C$ **do**
  $C' \leftarrow C' \cup c_{X'_i.label}$
$C' \leftarrow C' \cup c_{X'_n.state \in F}$
return constructNHMM(M', C', n)

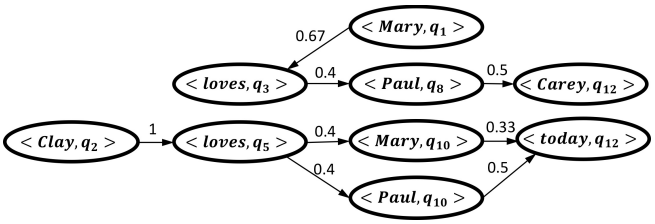

Figure 3: *A "state-sensitive" pseudo-Markov model.* This is the model M' built in Algorithm 2 given as inputs the automaton in Figure 1, the Markov model in Figure 2, an empty unary constraint set, and a length $n = 4$. This is a "pseudo"-Markov model because, given this approach, probabilities must remain unnormalized for proper construction of the NHMM.

A new set of unary "state-sensitive" constraints C' is also created such that for each unary constraint $c_{X_i} \in C$, an equivalent constraint $c_{X'_i.label}$ (meaning the same general constraint c applied to the *label* attribute of the random label-state variable X'_i) is added to C' which applies only to the label x in the new label-state pair $x' = (x, q)$. An ACCEPTING constraint $c_{X'_n.state \in F}$ applying to X'_n is also added to C' to ensure that all valid Markov sequences from M' end with a label-state pair $x' = (x, q)$ where $q \in F$.

The final step of Algorithm 2 constructs a NHMM from the unnormalized M', C', and n using the construction set forth by Pachet *et al.* [2011] which normalizes probabilities to create the target distribution p^*.

5 Structured Parallel Sequence Generation

RELATIONAL automata are effective tools for being able to impose both Markovian transitional constraints (*horizontal* structure) and relational constraints (*vertical* structure). This is well-demonstrated in using binary relational constraints to constrain parallel interrelated sequences, as we show in the following example of generating a musical sequence with

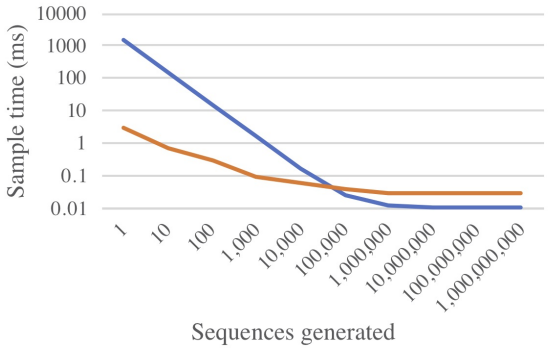

Figure 4: *Amortized Sample Time By Sequences Generated*. Shown are average amortized sample times per sequence (belonging to the set $\{aa + b+\}$ of fixed length 100) when sampling using a NHMM (blue) and factor graph (orange). The NHMM has a longer build time but shorter sample time per sequence meaning that as the number of sequences increases, the NHMM has a lower amortized sample time than the factor graph.

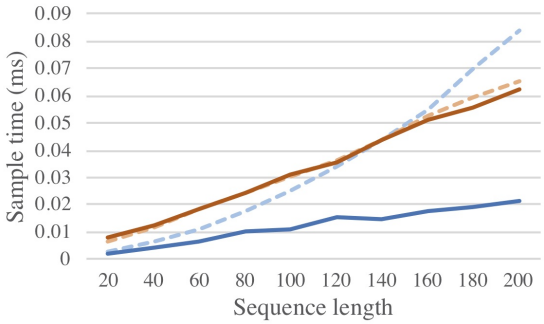

Figure 5: *Sample Time By Length*. Shown are average sample times for the NHMM (blue) and factor graph (orange) from sampling 100,000 sequences belonging to the set $\{aa + b+\}$. Both sample times increase linearly with the sequence length. Though the sample time per sequence is always lower for the NHMM, the NHMM build time also increases with sequence length resulting in a lower *amortized* sample time (dotted lines) for factor graphs as the sequence length increases.

global verse-chorus structure across multiple viewpoints.

Verse-chorus structure is effected in lyrical sectional-form music when subsequences of different types or *viewpoints* (e.g., chords, pitches, rhythms, and lyrics) are repeated at intervals throughout the musical sequence. For example, a *verse* is generally regarded as a segment where the chords, pitches, and rhythms are repeated, but the lyrics are different and a *chorus* is generally regarded as a segment where all viewpoints are repeated.

Repetition of subsequences requires the use of a series of binary relational constraints using the equality binary relation. Rather than hand-craft a set of relational constraints $\mathcal{M}_v$ for each viewpoint v, we learn existing patterns of repetition from normalized data and then translate these patterns into sets of binary relational constraints. For each of four viewpoints (chords (H), rhythm (R), lyrics (L), and pitch (P)) we perform multi-Smith-Waterman (mSW) alignment on the discretized musical sequence using parameterizations as found by **?** [**?**] on the song *Twinkle, Twinkle, Little Star*. This song is characterized by a chorus-verse-chorus structure with each segment lasting 4 measures. Graphic representations of the structural patterns learned from this song are shown in Figure 6. For each viewpoint $v \in \{H, R, L, P\}$ the alignment identifies matching regions (signified by dark red diagonals) of the training song according to v. Each alignment essentially defines for each position i a set of matching positions $S_i = \{j | j > i\}$ such that $\mathcal{M}_v = \{< i, j, \rho > | \forall i, j$ such that $j \in S_i\}$ where ρ is the equality relation.

In addition to the vertical structure imposed by these binary relational constraints, we added the following constraints:

Chord. Because binary relational constraints are designed to enforce structural repetition, it is not uncommon for generated sequences (particularly those from models trained on small data sets) to become *overly* repetitive, repeating structure even when it is not enforced. To counteract this effect in the chord sequence, we added a second set of binary relational constraints which constrained a subset of positions within regions *not* aligned via the mSW alignment to *not* match according to the equality relation. The model is constrained to generate sequences starting and ending with a C major chord.

Rhythm. We add unary constraints to make the generated rhythmic sequence end with a half-note rhythm to evoke a sense of finality.

Lyric. The length (in syllables) of the lyric sequence n_L is derived from the number of sampled rhythm tokens representing non-rest rhythmic onsets. Prosody (i.e., aligning stressed syllables with stressed beats) is achieved by constraining syllables for offbeat rhythms to be unstressed. Rhyming is effectuated using a second binary relational constraint set $\mathcal{M}'_L = \{(X_i, X_j, \rho)\}$ with ρ representing the binary relation for which a pair of syllables $(x_i, x_j) \in \rho$ iff x_i and x_j have identical phonemic nuclei and codas. $\mathcal{M}'_L$ is constructed such that the last syllables in measures 2 and 4 rhyme and the last syllables in measures 6 and 8 rhyme.

Pitch. We constrain the last pitch in the sequence to be the root of the final chord. For all positions i we constrain the pitch at i to equal one of the pitches defining the chord at position i. We constrain the last pitch in the sequence to be the root of the final chord. For all positions i we constrain the pitch at i to equal one of the pitches defining the chord at position i as per the previously sampled harmonic sequence.

The Markov models M_H and M_P for the chord and pitch sequences were trained on the chord and pitch sequence data in *Somewhere Over the Rainbow*. The rhythmic Markov model M_R was trained on rhythms from 5 different songs to give the model a rich choice of rhythmic variation. The lyric Markov model M_L was trained on the lyrics from the Beatle's *Hey Jude* and John Lennon's *Imagine* parsed using word pronunciations from the CMU dictionary[1] and the CMUSphinx grapheme-to-phoneme converter [Walker *et al.*, 2004].

Given the relational constraints $\mathcal{M}_v$, the Markov model

[1] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

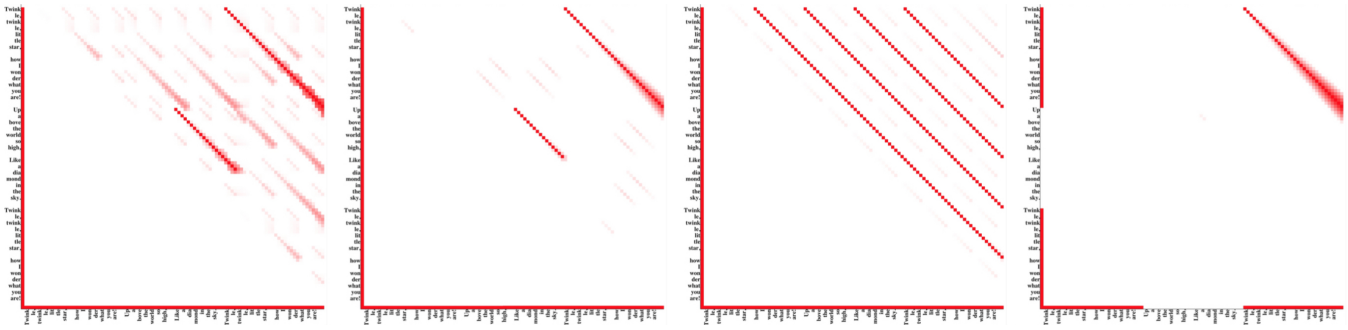

Figure 6: *Inferring Relational Constraints*. Relational constraint sets are inferred from real data using multiple-Smith-Waterman sequence alignment. Shown are the structural patterns inferred for the chord, pitch, rhythm, and lyric sequences in *Twinkle, Twinkle, Little Star*.

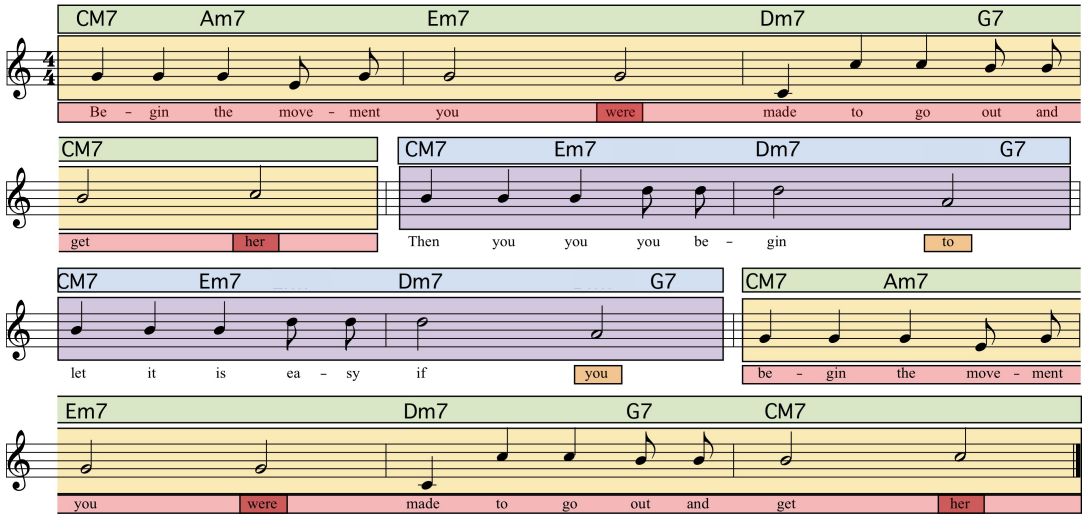

Figure 7: *Horizontal and Vertical Structure*. Shown are four parallel sequences (chords, pitches, rhythms, and lyrics) that exhibit both *horizontal* structure—each fully satisfies Markovian constraints—and *vertical structure*—each fully satisfies binary relational constraints, frequently in the same sequence positions as with relational constraints in other sequences. Boxes of the same color are used to illustrate subsequences which position-by-position are constrained via binary relational constraints to be equivalent. Dark red boxes reflect binary relational rhyming constraints. Not labeled is the pattern of rhythmic repetition every 2 measures.

M_v, and these additional binary and unary constraints, we generate a RELATIONAL automaton $\mathcal{A}_v$ and a REGULAR NHMM $\mathcal{N}_v$ for each viewpoint $v \in \{H, R, L, P\}$. The chord, rhythm, lyrics, and pitch sequences shown in Figure 7 were sampled from their respective NHMMs with probabilities 4.9e-5, 7.7e-6, 0.032, and 2.0e-3 respectively.

6 Discussion and Conclusion

Whereas arbitrary relational structure is possible using the approach taken by **?** [**?**], the approach we have presented is not restricted to using higher-order Markov transitions. This enables a very expressive model, even when trained on very small datasets.

Although for simplification we have focused exclusively on binary relations, the algorithms and logic in this approach could be adapted to allow for n-ary relations ρ and relational constraints $(X_{i_1}, ..., X_{i_n}, \rho)$. Such adaptations are unlikely to affect runtime or space (except to the extent that they span longer distances) because they potentially constrain the model more heavily, thus reducing the search space more significantly. These impacts and possible applications of n-ary relational constraints in Markov processes are a possible target of future research.

We have demonstrated the modeling of relational constraints in Markov processes using a RELATIONAL automaton. We have also provided a method for exact sampling of sequences from an arbitrary automaton according to Markovian probabilities using a REGULAR NHMM. These solutions enable probabilistic sampling of sequences that exhibit arbitrarily complex relational structure while also maintaining natural, fully-Markovian transitions and allow via low Markov orders a broad range of expression.

References

[Barbieri *et al.*, 2012] Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. Markov constraints

for generating lyrics with style. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 115–120. IOS Press, 2012.

[Collins and Laney, 2017] Tom Collins and Robin Laney. Computer–generated stylistic compositions with long–term repetitive and phrasal structure. *Journal of Creative Music Systems*, 1(2), 2017.

[Pachet *et al.*, 2011] François Pachet, Pierre Roy, Gabriele Barbieri, and Sony CSL Paris. Finite-length markov processes with constraints. *transition*, 6(1/3), 2011.

[Pachet *et al.*, 2017] François Pachet, Sony CSL Paris, Alexandre Papadopoulos, and Pierre Roy. Sampling variations of sequences for structured music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR'2017), Suzhou, China*, pages 167–173, 2017.

[Papadopoulos *et al.*, 2014] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Avoiding plagiarism in markov sequence generation. In *AAAI*, pages 2731–2737, 2014.

[Papadopoulos *et al.*, 2015] Alexandre Papadopoulos, François Pachet, Pierre Roy, and Jason Sakellariou. Exact sampling for regular and markov constraints with belief propagation. In *International Conference on Principles and Practice of Constraint Programming*, pages 341–350. Springer, 2015.

[Pesant, 2004] Gilles Pesant. A regular language membership constraint for finite sequences of variables. *CP*, 3258:482–495, 2004.

[Roy *et al.*, 2016] Pierre Roy, Guillaume Perez, Jean-Charles Régin, Alexandre Papadopoulos, François Pachet, and Marco Marchini. Enforcing structure on temporal sequences: The allen constraint. In *International Conference on Principles and Practice of Constraint Programming*, pages 786–801. Springer, 2016.

[Walker *et al.*, 2004] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical report, Sun Microsystems, Inc., 2004.