Constrained Sequential Inference in Machine Learning Using Constraint Programming

Virasone Manibod[2] , **David Saikali**[1] , **Gilles Pesant**[1]

[1]Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada
[2]GIRO, Montreal, Canada
virasonemanibod@hotmail.com, {david.saikali, gilles.pesant}@polymtl.ca

Abstract

Sequence models in machine learning often struggle to exhibit long-term structure. We consider this problem at inference time in the context of enforcing constraints that are not necessarily featured in the dataset on which the generative model was trained. The difficulty lies in imposing previously-unseen structure while staying close to the training dataset. It is particularly hard for long-term structure, which requires balancing foresight over many yet-to-be generated tokens and the immediacy of next-token predictions from the sequence model. We address this problem by introducing our neurosymbolic framework GeAI-BLAnC. The learned probabilities of the sequence model are mixed in with the marginal probabilities computed from a constraint programming / belief propagation framework applied to a constraint programming model expressing the desired structure. The next predicted token is then selected from the resulting probability distribution. Experiments in the context of molecule and music generation show that we can achieve the structure imposed post-training without straying too much from the structure of the dataset learned during training.

1 Introduction

Sequence models in machine learning often struggle to exhibit long-term structure, stemming in part from the token-by-token nature of the prediction process used to generate a sequence. This problem can arise both during training, where such global structure must be learned, and during inference (generation), where the gradually-outputted sequence must be guided toward that structure, each with its own challenges. In this paper we consider the latter.

Imposing structure at inference time is sometimes useful in combination with a training phase that may have learned the structure only partially, in order to fine-tune the sequence being generated, especially if that structure is mandatory [Deutsch *et al.*, 2019; Lee *et al.*, 2019]. It is equally relevant when we wish to impose constraints that are not featured in the dataset on which the sequence model was trained, either to avoid the expense of retraining a large model or to offer a flexible interactive system (e.g. computer-aided music composition). The difficulty in such a context is to impose that structure while staying close to the dataset the model was trained on. Indeed, one important aspect of controllability is to keep a satisfying quality of the generated sequence while respecting the constraints [Young *et al.*, 2022]. It is particularly hard for long-term structure, which requires balancing foresight over many yet-to-be generated tokens and the immediacy of next-token predictions from the sequence model.

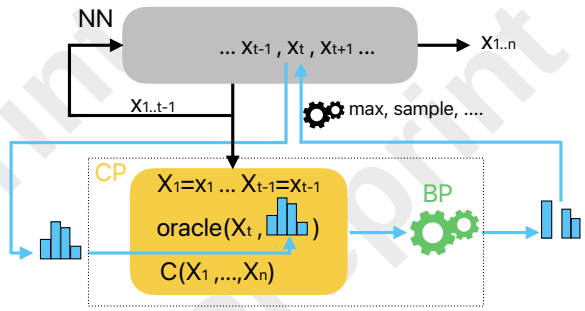

Figure 1: Our neurosymbolic architecture for constrained sequence generation. At each token generation step, the application of BP on the CP model modifies the trained neural model's output probability distribution. The next token is selected using that new distribution.

We address this problem in the following way (see Fig. 1). To enable the introduction of post-training combinatorial structure in sequences outputted by a trained machine learning model, we first express that structure as a constraint programming (CP) model. Indeed, it offers the advantage of being declarative and modular, both desirable features in an interactive setting where one tries adding and combining constraints. Stating such a model on a representation of the sequence to be generated ensures that the added constraints will be satisfied. However, the resulting sequence will not necessarily be representative of its training dataset. Hence at each token-generation step we feed the probability distribution from which a token should be selected to a CP solver equipped to handle such probabilities on domain values. We then perform belief propagation (BP) on the CP model in order to produce a distribution taking into account the desired

structure. Finally, the next token is selected from that new probability distribution. To achieve the balancing act sometimes necessary to smoothly handle long-term structure, we also provide the ability to progressively adjust the impact of the learned distribution on the belief propagation process as the sequence is generated.

The originality of our contribution, beyond the fact that few works have used CP for generative AI, lies in the CP-BP framework being used to guide a trained neural network during the inference phase. We introduce the `oracle` constraint that acts as a bridge between the two parts and propose the novel ability to apply a weight to messages from constraints in order to handle long-term constraints more smoothly.

The next section provides some background. Section 3 presents related work on adding structure to machine learning models. Our framework GeAI-BLAnC integrating constraint programming in a sequence generation model is detailed in Section 4. Sections 5 and 6 describe and discuss our experiments in the domain of molecule and music generation respectively. Finally Section 7 provides concluding remarks.

2 Background

2.1 Generative Neural Models

One can hardly overstate the surge in accomplishments and in public interest for generative AI, where huge neural models are trained with vast amounts of data in order to respond to prompts by generating realistic text, images, music, computer code, or other structured output. In this paper we consider sequential data such as text. We briefly present some of the most relevant machine learning architectures and mechanisms to learn from sequential data.

The *Recurrent Neural Network* (RNN) [Hochreiter and Schmidhuber, 1997] is a type of neural network for sequential data. It can be trained to predict the next element of a sequence based on the previous input's information. Thus, once trained, it can be used to generate new sequences, token by token. A RNN is able to predict what's following due to its *hidden state* which functions as a *memory* for the previous inputs. Therefore, at each index of the sequence, the current input and the *hidden state* are used to predict the next element. The *Long Short-Term Memory* (LSTM) [Hochreiter and Schmidhuber, 1997] was created to overcome the vanishing or exploding gradient problem from the RNN. The LSTM is able to learn better long-term dependencies with its *cell state* which can decide what information from the past to keep or forget. The *self-attention mechanism* [Parikh *et al.*, 2016; Vaswani *et al.*, 2017] is another way to add long-term dependencies in neural networks. It consists of analyzing the relationship between the elements within a same sequence. Therefore, during an autoregressive generation, the model can determine which previously generated tokens should the current token pay more attention to.

A *Transformer* [Vaswani *et al.*, 2017] is a deep learning architecture used in natural language processing (NLP). It is made up of a stack of encoders and a stack of decoders. The encoder layers feed their output to the next encoder in the stack, the final encoder then passes it to the decoders' multi-headed cross-attention. Each encoder layer contains a self-attention mechanism as well as a feed-forward layer, while the decoders contain a masked multi-headed attention layer, a feed-forward layer as well as the previously mentioned multi-headed cross-attention layer.

2.2 Constraint Programming

Constraint Programming (CP) [Rossi *et al.*, 2006] is a paradigm used to solve constraint satisfaction problems (CSP). A CSP is a tuple $(\mathcal{X}, D, C)$ where $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a finite set of variables, $D = \{d_1, \ldots, d_m\}$ is a finite set of values and $C = \{c_1, \ldots, c_\ell\}$ is a finite set of constraints, i.e. relations over a subset of variables from $\mathcal{X}$. To each variable X_i we associate a domain $D(X_i) \subseteq D$ from which it takes its value, which can be seen as a unary constraint. The goal of a CP solver is to find an assignment (or multiple ones) for each variable from their respective domain such that every constraint in the CSP is satisfied. It typically proceeds by backtracking search interleaving branching and *constraint propagation*, a form of message-passing between constraints where each filters out *inconsistent* domain values (i.e. which cannot satisfy that constraint). For example let $\mathcal{X} = \{X_1, X_2, X_3\}$, $D(X_1) = \{0, 1, 2\}$, $D(X_2) = D(X_3) = \{1, 2, 3\}$ and consider the following constraints: $C = \{\texttt{alldifferent}(X_1, X_2, X_3),\ X_2 + X_3 = 5,\ X_1 + X_3 > 2\}$. The second constraint filters out inconsistent value 1 from the domain of X_2 and X_3 and propagates that information to the first one which in turn filters out 2 for X_1. This CSP admits three solutions (X_1, X_2, X_3): (0, 2, 3), (1, 2, 3), (1, 3, 2).

The *CP-BP framework* [Pesant, 2019] augments CP with the ability to perform belief propagation (BP) on the model. BP is a message-passing algorithm to perform inference on graphical models [Pearl, 1982]. In the context of CP, it approximates the marginal probabilities of being part of a feasible solution for each variable-value pair. In the previous example, value 1 for variable X_1 is present in two out of three solutions. Thus, that variable-value assignment has a true marginal probability 2/3 of being part of a feasible solution. These marginals not only tell us whether a value for a variable is possible (e.g., that value 2 is inconsistent for X_1), they also tell us how likely they are to lead to feasible solutions. Computing marginals corresponds to *weighted counting*. Despite being intractable in general, for many constraints we can compute them efficiently and for the others we rely on approximations.

3 Related Work

We focus our discussion on constrained sequential inference involving generative NNs, particularly if they involve constraint programming as well. Broader presentations of neurosymbolic AI, *e.g.* about structure learning, may be found in recent surveys such as [Marra *et al.*, 2024; Yu *et al.*, 2023].

Some works carry out post-training adjustments to the NN according to constraints. [Lattner *et al.*, 2018] use a convolutional restricted Boltzmann machine as a generative model and constraints are enforced as differentiable cost functions that are minimized during the sampling process to resemble the structure of a reference musical piece. [Lee *et al.*,

2019] use gradient-based inference to continue adjusting the model's parameters toward the satisfaction of the constraints during inference. [Dragone *et al.*, 2021] introduce a *constrained structured predictor* expressed in a CP language that acts as a final layer to a NN and which is trainable to finetune the predictions but also enforces the constraints during inference. [Young *et al.*, 2022] use bias-tuning [Zaken *et al.*, 2021] to steer the output of the Music Transformer [Huang *et al.*, 2018], making it more likely to produce a sequence with the desired constraints. [Jaques *et al.*, 2017] use reinforcement learning to finetune an RNN to take into account domain-specific rules while staying close to the original RNN. Rules are manually encoded into the reward signal. [Lafleur *et al.*, 2022; Yin *et al.*, 2024] improve the latter by more generically expressing arbitrary constraints in the CP-BP framework and by automatically deriving a reward signal.

Other works use the trained NN as is. For a very constrained text generation task related to vision screening, [Bonlarron *et al.*, 2023] encode all valid texts in a multi-valued decision diagram and then use an LLM to evaluate their quality *a posteriori*. [Bonlarron and Régin, 2024] alternatively defines a new constraint on n-grams for CP-driven sequence generation that filters based on n-gram perplexity scores computed *a priori* by an LLM. In a CP-driven generation, [Régin *et al.*, 2024] builds a CSP incrementally by adding sequence variables on the fly and limiting their domain to an LLM's short list of candidate tokens, queried at each step. [Deutsch *et al.*, 2019] express commonly-used constraints in NLP as automata. At each step, the automata filter out the inconsistent token values and renormalize the NN's output probability distribution accordingly.

Our method is closest in spirit to the latter. We too modify the output probability distribution by removing inconsistent values but, more importantly, we potentially change these probabilities relative to each other to also reflect the marginal probabilities computed from the constraints of our CP model, and thus guide the generation toward values more likely to satisfy the constraints. Using CP also offers a variety of constraints, including automata [Pesant, 2004], and inconsistent values are removed at each step through constraint propagation instead of checking the full completion of the partial sequence with the automata.

4 Our Framework

We describe our framework GeAI-BLAnC (Generative AI using BeLief-Augmented Constraints) to add given combinatorial structure to the output sequence of a generative NN [Manibod, 2022]. We use the MiniCPBP solver [Pesant, 2019] implementing the CP-BP framework which, in addition to standard constraint programming support, offers marginal probabilities for the model variables, which is critical to guide our sequence generation.

The sequence of tokens to be generated can be represented in a CP model as a sequence of variables $X_1, \ldots, X_n$, each having a domain corresponding to the possible token values. As shown in Figure 1, at each step during the generation phase, CP is used to modify the distribution for the current x_t token obtained from the NN's softmax layer. That probability distribution only takes into account the structure learned from the dataset. Through BP the CP model will modify that distribution to obtain a new one that will also take into consideration the satisfaction of the constraints. The next token x_t is then selected using that new distribution, for example by sampling from it or by choosing one of its modes. Standard constraint propagation on the CP model would filter the domain of the corresponding variable X_t according to the constraints, possibly removing (token) values that cannot satisfy them. Belief propagation subsumes this: it will assign zero probability to such inconsistent values but more generally computes marginal probabilities for each value in the domain of X_t.

We now detail the components of the CP model. The tokens generated so far, $x_1, \ldots, x_{t-1}$, are used to set the corresponding variables $X_1, \ldots, X_{t-1}$ in order to take them into account in the model. The NN's learned probabilities for each candidate token value at current step t are also given to the CP model through a unary constraint that is added to the model: we introduce the $\texttt{oracle}(X, p)$ constraint with X a finite-domain variable and p a fixed probability mass function over the domain of X. In our case the variable is X_t, representing the token at step t, and p is the NN's probabilities for the current token x_t. Uncharacteristically, this constraint does not enforce a relation but only associates a probability to each domain value. Its sole purpose is to contribute messages to variable X_t during BP in the same way as the other constraints in the CP model. Finally the arbitrary structure we wish to impose on the output sequence is represented as constraint $C(X_1, \ldots, X_n)$. Without the `oracle` constraint, the resulting marginals would only take into account the satisfaction of $C(X_1, \ldots, X_n)$ and not what was learned from the dataset. Therefore, the `oracle` constraint is our way of integrating the NN's knowledge into the process of CP-BP.

In order to balance such an integration, we also introduce the ability to associate a weight to each constraint. This weight affects the marginals sent by constraints during BP (and not the filtering nor the hardness of the constraint). Given a positive weight w (the default value being 1) each marginal $p_X(v)$ for a value v in the domain of variable X is raised to the power of that weight and normalized, yielding new marginal $(p_X(v))^w / \sum_{d \in D(X)} (p_X(d))^w$. As a result, a weight $w > 1$ accentuates the disparities between marginals while $0 < w < 1$ lessens them and makes them more uniform. We will use such a weight on the `oracle` constraint in order to control its importance on the resulting distribution.

5 Application to Molecule Generation

We first demonstrate and evaluate our framework in the domain of chemistry. We aim to show through our experiments that our approach more consistently generates sequences exhibiting the desired structure while still reflecting what the NN has learned from the training corpus. GPT2-Zinc480M-87M is a transformer model trained to generate molecules in the standard string representation SMILES [Weininger, 1988] (Fig. 2 gives an example). Valid strings can be described by a context-free grammar. The syntax is quite restrictive and difficult to learn by a NN: using this transformer to gener-

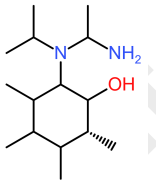

Figure 2: "C[C@H]1C(O)C(N(C(N)C)C(C)C)C(C)C(C)C1(C)", a molecule of weight 256.3 Da and PPL=1.8487 generated by GeAI-BLAnC.

ate 40-token molecules, which is the length we report on in our experiments, only yielded a 73% validity rate. In contrast we achieve 100% by adding to this transformer a CP model through our framework.

For the CP part of our architecture, we represent the generated molecules as a sequence of $n = 40$ variables, each having a domain corresponding to the SMILES alphabet (45 values). Note that such a sequence length is long enough to represent most molecules in our dataset, including many in pharmaceutical use.

5.1 Experimental Set-Up

The GPT model, GPT2-ZINC480M-87M[1] (henceforth referred to simply as GPT), has 87M parameters and was trained on 480M molecules from the ZINC database[2]. During generation we kept the model's default configuration with the following exceptions: we limited the generation to one new token at a time as per Fig. 1, we set the model's temperature to 1.5 which gives more varied results as reported by the model's authors, we decreased the maximum length of the molecule to fit our target length, and we disabled the early stopping parameter. For the CP part we use MiniCPBP[3] with the default parameter values. The initial input is the start-of-sequence token. We attempted to generate 100 molecules. Our experiments were run using an 8-core processor with a core speed of 4.20 GHz and 64GB of RAM. All our code and data are available[4].

5.2 Evaluation Metric

To evaluate our molecule's quality, we use perplexity [Jelinek *et al.*, 2005], a common metric in NLP:

$$\mathrm{PPL}(x_1, \ldots, x_n) = \exp\left(-\frac{1}{n}\sum_{t=1}^{n} \log p(x_t | x_1, \ldots, x_{t-1})\right)$$

The higher the perplexity, the less likely it would be generated by the neural model and the more surprising it is with respect to the training set. Note that given the imposed constraints, the CP model may disagree with the probability distribution it receives from GPT, modify it, and therefore receive a higher perplexity value for the chosen token. We also consider perplexity for individual tokens, $1/p(x_t|x_1, \ldots, x_{t-1})$, in order to track its behaviour across the whole sequence.

[1]https://huggingface.co/entropy/gpt2_zinc_87m
[2]https://zinc.docking.org/
[3]https://github.com/PesantGilles/MiniCPBP
[4]https://github.com/cravethedave/MiniCPBP/tree/ijcai-2025

method	success(%)↑	PPL↓	time(s)↓
GPT	7	5.30	1.2
CPBP (backtrack)	100	1236.71	125.4
CPBP (no backtrack)	59	503.48	62.4
GPT+CP	18	13.50	25.2
GPT+CPBP ($w = 1$)	92	8.35	67.2
GPT+CPBP $w = 0.5$	82	17.30	70.8
GPT+CPBP $w = 1.5$	72	8.14	63.6

Table 1: Success rate, average perplexity, and average runtime over 100 attempts to generate weight-constrained 40-token molecules.

5.3 Molecular-Weight Constraint

While obtaining valid SMILES molecules is interesting in itself, generation usually targets some properties, e.g. molecular weight. To constrain our sequence generation, we choose to impose a weight between 200 and 275 Da, only achieved by a small fraction of possible 40-token molecules.

Formally we define a CSP $(\mathcal{X}, D, C)$ with $\mathcal{X} = \{X_1, \ldots, X_{40}\}$, $D(X_i) = \{C, N, O, @, (, \ldots\}$, the set of variables representing each of the 40 tokens to generate. Constraints are

$$200 \leq \textstyle\sum_{i=1}^{40} \mathbf{w}[X_i] \leq 275$$
$$\texttt{grammar}(\langle X_1, X_2, \ldots, X_{40}\rangle, \mathcal{G}_{\mathrm{SMILES}})$$

where $\mathbf{w}$ is a table of weights indexed by token values and $\mathcal{G}_{\mathrm{SMILES}}$ is a context-free grammar for SMILES strings. The first is our molecular-weight constraint, and we add the second to help enforce the validity of the generated SMILES sequences: indeed as mentioned before, GPT on its own had a 73% validity rate when generating 40-token molecules.

To this CP model we add at step t an $\texttt{oracle}(X_t, p)$ constraint and we set $X_1, \ldots, X_{t-1}$ to their generated value.

Table 1 reports the results of our experiments, comparing several approaches for constrained sequential inference: the transformer on its own, generating individual tokens in sequence (GPT); MiniCPBP on its own, generating individual tokens in sequence, sampling its value from the marginal distribution, both with and without backtracking (CPBP); our architecture without belief propagation (GPT+CP); our full architecture GeAI-BLAnC with different weights applied to the `oracle` constraint (GPT+CPBP).

Success rate. As expected, the molecules generated by GPT mostly do not satisfy the constraint (or are simply invalid) and CPBP with backtrack search ultimately generates a satisfying molecule every time. Greedy CPBP (i.e. without backtracking) achieves a much better success rate than GPT but may still fail to complete a sequence. GPT+CP shows that using CP simply to filter out inconsistent values (i.e. what [Deutsch *et al.*, 2019] would achieve) performs barely better than GPT. Among all non-backtracking methods, GPT+CPBP performs better by far for all three values of w with the default value achieving the best results (92%).

Perplexity. Despite its moderate to high success rate CPBP on its own tends to generate unrealistic molecules, as indicated by very high perplexity scores. GPT+CP and GPT+CPBP achieve fairly low PPL (see GPT's as reference),

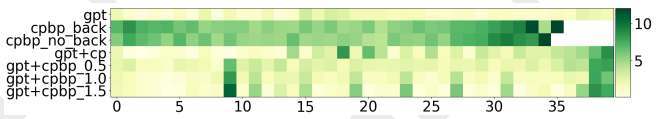

Figure 3: Average perplexity indexed by token (darker is higher).

with GPT+CPBP (aka GeAI-BLAnC) offering an excellent combination of high success rate and low perplexity score. The impact of weight w on perplexity is consistent with its effect on the distribution from the NN: lessening its importance makes the PPL rise and inversely. Figure 3 tracks PPL of individual tokens across the sequence: it confirms PPL rankings between methods without showing any clear imbalance with respect to token position, which in contrast we shall witness in Section 6.

Computation time. Using CP and BP comes at a computational price: whereas GPT alone may generate a molecule in about one second, adding CP slows it down to about half a minute and adding BP as well pushes it to about one minute.

6 Application to Melody Generation

We next demonstrate and evaluate our framework in the domain of music. Our experiments again measure how close to the dataset the generated sequences are but focus on the smooth enforcement of long-term structure. The Chord-conditioned Melody Transformer (CMT) [Choi *et al.*, 2021] is a neural model that generates a melody based on a given chord progression $c_1, \ldots, c_n$. It is made up of a chord encoder (CE), a rhythm decoder (RD) and a pitch decoder (PD). CMT generates a rhythm sequence $r_1, \ldots, r_n$ and then a pitch sequence $p_1, \ldots, p_n$ in two separate steps, each token being sampled from the distribution (for pitch, sampling is restricted to the top five values). Note that all sequences are time-indexed, the step unit being a 16th note, and they span 8 bars so $n = 8 \times 16 = 128$.

For the CP part of our architecture, we represent the rhythm and the pitch sequences as two sets of variables, each having a domain corresponding to the possible token values: *onset*, *rest*, and *hold* for rhythm, and 48 MIDI pitches plus *rest* and *hold* for pitch. Note that if the rhythm token at time step t is *hold* then the pitch sequence token at the same time step must also be *hold* (likewise for the *rest* token). We build one CP model for the rhythm and one for the pitch to be able to constrain both aspects of the melody. The overall system thus has two components in sequence, each following the architecture of Fig. 1, the first one (rhythm) taking the chord sequence as additional input to its NN and the second one (pitch) taking both the chord and rhythm sequences as additional input.

6.1 Experimental Set-Up

We used the publicly-available code of CMT[5] with the same preprocessing and hyperparameters as [Choi *et al.*, 2021] and of MiniCPBP with the default parameter values. We ran all our experiments using 8 CPU cores, 48,000 M memory and a GPU type v100 with 32 GiB memory. The dataset used for the experiments is the EWLD [Simonetta *et al.*, 2018] containing more than 5,000 lead sheets in many musical styles, but mainly Jazz, Pop and Rock. All songs were transposed to C major or A minor. Because CMT is limited to 8-bar melodies, the lead sheets were cut up into 8-bar segments. After preprocessing, instances were split into training and test datasets of respective size 23,800 and 3,150. We generated 3,150 melodies, each primed with the first four ground-truth tokens of a test set melody and the corresponding chord sequence. All our code and data are available[6].

6.2 Evaluation Metrics

Even though evaluating music remains a challenge because it can be highly subjective, some objective metrics have been proposed in the literature to evaluate automated music generation. The MGEval framework [Yang and Lerch, 2020] compares several pitch and rhythm features between the test and generated datasets and expresses their similarity as a value between 0 and 1, the latter corresponding to perfect similarity. We report on a few that are representative of those most affected here: note count (NC), pitch range (PR), and pitch interval (PI). In order to be consistent with the other metrics, we use 1 minus the original metric so that closer to 0 is better.

We also introduce two metrics specific to rhythm. For the first one we compute the distribution of rhythm patterns of a bar, separately in the test and generated datasets. The Jensen-Shannon divergence (JSD) [Lin, 1991], ranging between 0 and 1, is then calculated between these distributions and we denote it RPD for Rhythm Pattern Divergence. For the second one, considering together bars containing the same number of notes, the normalized placement distribution of onset tokens in a bar is computed for the test and generated datasets. The average JSD is then calculated between every pair of distributions containing the same number of notes, denoted OTPD for Onset Tokens Placement Divergence. This metric differs from the previous one as it focuses on the placement of notes instead of the specific rhythm patterns.

One last metric specific to pitch, the chord tone ratio (CTR) [Choi *et al.*, 2021] is also used. Their intuition behind this metric is that a harmonious melody would contain notes that appear in the triad of the chords (e.g. $\{C, E, G\}$ for a C major chord). Therefore the CTR computes the ratio of notes found in the triad of the chord that is playing when the note of the melody is playing. In other words, it tells us how harmonious a melody is given its chord progression. We report the difference between the average CTR in the test and generated datasets.

6.3 Distinct-Number-of-Notes-per-Bar Constraint

Our first experiment imposes on melodies to have a different number of notes played in each bar. It is a restriction that only affects the rhythm. To study the effects of long-term structure on a variable number of bars, it is applied to the first span of four to eight bars. Note that only 19% of our test dataset satisfy such a constraint when applied on the first group of four bars and less than 0.2% on eight bars, which is indicative of how little success CMT would have on its own.

[5]https://github.com/ckycky3/CMT-pytorch

[6]https://github.com/Manibod/CMT_CPBP

nb bars spanned	4	5	6	7	8
RPD↓	0.203	0.254	0.297	0.334	0.367
OTPD↓	0.019	0.022	0.026	0.029	0.032
NC↓	0.081	0.108	0.136	0.194	0.267
RPD↓	0.213	0.271	0.321	0.369	0.403
OTPD↓	0.018	0.019	0.022	0.025	0.028
NC↓	0.091	0.126	0.182	0.248	0.323

Table 2: Rhythm metrics of CMT for the Distinct-Number-of-Notes-per-Bar Constraint (top: unweighted `oracle`, bottom: geometrically decaying weight). The smaller the value the better.

Formally we have CSP $(\mathcal{X} \cup \mathcal{O}, D, C)$ with $\mathcal{X} = \{X_1, \ldots, X_{128}\}, D(X_i) = \{rest, hold, onset\}$, the set of variables representing each of the 128 rhythm tokens to generate, $\mathcal{O} = \{O_1, \ldots, O_b\}, D(O_i) = \{0, \ldots, 16\}$, the number of occurrences of the *onset* token in each of the b constrained bars. Constraints C are

$$\texttt{among}(\{X_{((i-1)\cdot 16)+1}, \ldots, X_{i\cdot 16}\}, onset, O_i) \quad 1 \leq i \leq b$$
$$\texttt{alldifferent}(O_1, \ldots, O_b)$$

To this we add at step t an $\texttt{oracle}(X_t, p)$ constraint and we set $X_1, \ldots, X_{t-1}$ to their generated value.

The melodies we generate with GeAI-BLAnC satisfy the constraint by design, but do they resemble the training corpus? The top half of Table 2 shows the rhythm metrics about the effect of adding the Distinct-Number-of-Notes-per-Bar Constraint to the generation of the melodies. Overall the generated rhythms are less and less similar to those in the test dataset as the constraint's span increases and the melody thus becomes more restricted. This is expected since, as previously noted, sequences satisfying that constraint are scarce in the music corpus. But the generated sequences are not overly dissimilar. To put things into perspective, we also experimented with generating them solely with CP: it yielded a RPD of 0.99. This highlights the advantage of our proposed neurosymbolic architecture.

Procrastination Problem. An interesting result is shown in Figure 4. The heat maps show, among the top 10 bar rhythm patterns being the i^{th} bar when the constraint is applied to groups of b bars, how often the *onset* token was found at a given time step in a bar. A noticeable issue, compared to what one observes in the test dataset, is that a lot of the *onset* tokens are found toward the end of a bar, especially in the last few bars of the constraint's span. This is indicated by the darker region at the bottom right of each heat map. This problem is more prominent as the constraint's span increases. It seems the overall model generates the melodies based mostly on CMT's probabilities until the last moment when it has no other choice but to select an *onset* token to satisfy the constraint. It is as if the constraints are not pressing enough except at the end. We call this the *procrastination problem*. A more realistic or uniform *onset* token placement is desired: indeed in the test dataset only 4% of bars end with an *onset* token whereas in the generated dataset, it ranges from 27% to 43% when the constraint is applied to 4 up to 8 bars. We show next how we address this problem.

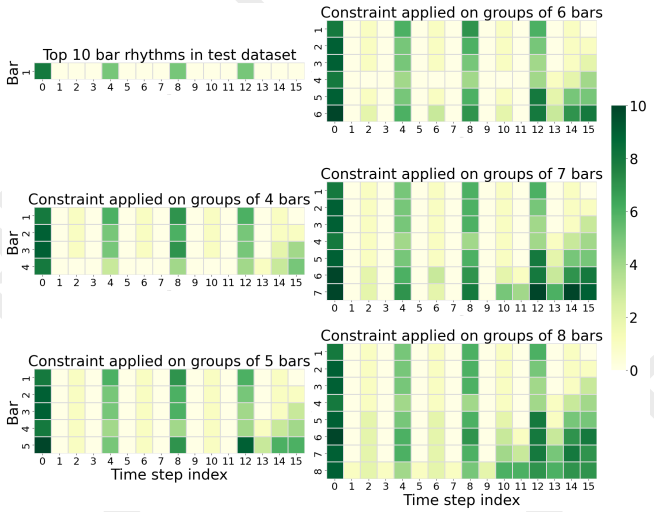

Figure 4: Top 10 bar rhythm patterns being the i^{th} bar when the Distinct-Number-of-Notes-per-Bar Constraint is applied to groups of b bars presented as heat maps of *onset* tokens at each time step of a bar (unweighted `oracle`).

Decreasing Weight of the `oracle` Constraint. Our solution to mitigate the *procrastination problem* is to give more impact to the satisfaction of the constraints relative to CMT's marginals by decaying the weight of the `oracle` constraint throughout the generation (initial attempts to use a *fixed* weight did not solve our problem). The intuition to decay the `oracle` weight is to gradually accentuate the pressure of the constraints. When lowering the `oracle` weight, it becomes more difficult for the CMT model to encourage a certain token value because their marginals will be brought closer together. After some initial experiments, we concluded that a geometric decay r^k, with ratio r chosen so that the weight reaches about 0.7 once half the sequence has been generated ($r = 0.9943$) and k representing the index of the token in the span, could lead to making the constraints more pressing and having a more uniform onset token placement.

Figure 5 shows that the procrastination problem has been greatly mitigated judging from the disappearance of the dark regions. The number of bars ending with an onset token in the generated dataset is reduced, now ranging from 23% to 32%. The bottom half of Table 2 shows some metrics decreasing (OTPD) and others increasing (RPD, NC) but no major change. Therefore *this offers a reasonable balance between overall similarity with the test dataset and uniformity in how that similarity is spread over the whole sequence, all this while satisfying the constraint.*

6.4 Occurrence-of-Notes-in-Key Constraint

We next consider a constraint affecting pitch in order to experiment in the context of a larger set of token values. This time the restriction consists of having each note in the scale (i.e. C,D,E,F,G,A,B) occur at least once in the melody (regardless of the octave). We apply it to the initial span of six to eight bars. Note that only 13% of our test dataset satisfy such a constraint. To make sure there are enough notes, we

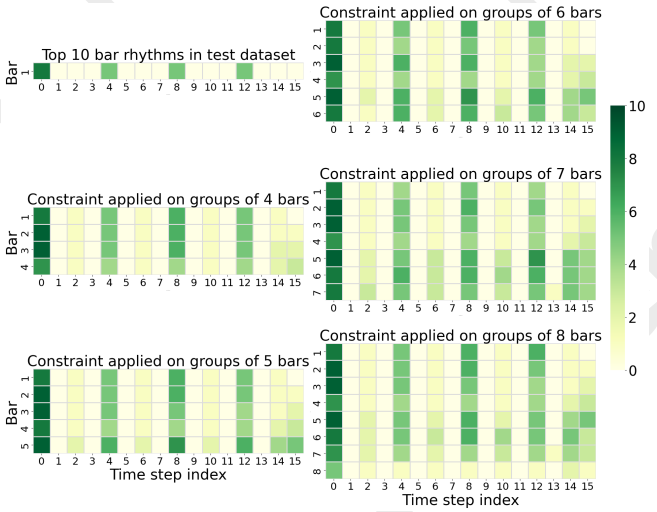

Figure 5: The same as Fig.4 but with a geometrically decaying weight applied to the `oracle` constraint.

nb bars spanned	6	7	8
PR↓	0.142	0.149	0.143
PI↓	0.077	0.082	0.080
CTR↓	0.062	0.059	0.056
PR↓	0.091	0.086	0.094
PI↓	0.054	0.058	0.061
CTR↓	0.089	0.098	0.108

Table 3: Pitch metrics of CMT with the Occurrence-of-Notes-in-Key Constraint (top: unweighted `oracle`; bottom: geometrically decaying weight). The smaller the value the better.

also add a constraint on rhythm to have at least eight *onset* tokens in the constrained span of bars. This rhythm constraint should not heavily affect the generation of the melodies because CMT is already very likely to generate more than eight notes in a span of six bars or more.

We define a CSP as before but with $D(X_i) = \{0, \ldots, 49\}$, $\mathcal{O} = \{O_1, \ldots, O_{12}\}$ for each of the twelve possible pitch classes, and $D(O_i) = \{1, \ldots, n\}$ if pitch class i is in the scale else $D(O_i) = \{0, \ldots, n\}$, where n is the number of *onset* tokens in the span of b bars in the rhythm sequence. Constraints C are

$$\texttt{among}(\{X_1, \ldots, X_{b \cdot 16}\}, \text{Class}(i), O_i) \quad 1 \leq i \leq 12$$
$$\textstyle\sum_{i=1}^{12} O_i = n$$

where Class(i) is the set of tokens corresponding to pitch class i over different octaves.

The top half of Table 3 reports fairly low metrics, this time little impacted by the span of the constraint. Figure 6 (left) shows heat maps counting the number of first occurrences of the notes of the key at given locations. Once again there are very dark regions at the end. These are other manifestations of the *procrastination problem*. Indeed, it can be interpreted as the model having, at the end, no other choice but to generate the pitches that are missing in order to satisfy the

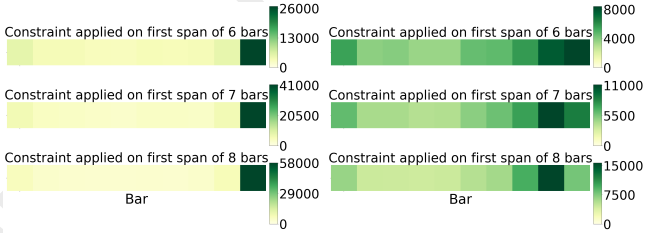

Figure 6: First occurrence of the notes of the scale within the span of constrained bars with the Occurrence-of-Notes-in-Key Constraint: unweighted `oracle` (left) and geometric decay (right).

constraint of generating each note of the key. The result can be very unpleasing to the ear. Again, the constraints are not pressing enough, and something more realistic or uniform is desired.

As before we apply a geometric decay of the `oracle` constraint's weight, choosing ratio $r = 0.985$ so that the weight reaches around 0.4 once half the sequence has been generated. Figure 6 (right) shows that once again the *procrastination problem* has been considerably mitigated. Indeed the distribution of the first occurrence of the notes in the scale is much more spread across the melodies as found in the test dataset. The bottom half of Table 3 reports some metrics decreasing (PR, PI) and others increasing (CTR), but all values remain quite low. It appears *we reach a good balance between the smooth satisfaction of long-term constraints and the similarity with what CMT has learned.*

Computation time. Using the original CMT (no constraints imposed), the total runtime was about 22 minutes. Imposing a rhythm constraint (3 token values), the runtime was about 30 minutes whereas with both rhythm and pitch (50 token values) constraints, it took about 90 minutes.

7 Conclusion

We presented our neurosymbolic framework GeAI-BLAnC to add long-term combinatorial structure to machine learning sequence models at inference time. A new sampling distribution was obtained by combining the learned probability distribution (through an `oracle` constraint) and the marginal probabilities of a CP model expressing the desired structure. We achieved a much better combination of dataset similarity and constraint satisfaction than NN or CP alone. The single parameter, weight w on the `oracle` constraint, can be used to balance the respective influence of the NN and CPBP probability distributions, or to promote a more uniform satisfaction of the constraints across the sequence, by decaying it throughout the generation.

As we saw in Section 5, if the structure is very restrictive, it can happen that a partial sequence cannot be completed despite the foresight provided by the computed (approximate) marginals. Adding *beam search* to our framework could help in such a situation.

Very recently [Zhang *et al.*, 2024] proposed an approach similar to ours that samples from a distribution modified to take into account constraints expressed as automata. A comparative analysis of the two frameworks should be of interest.

Acknowledgements

This research was made possible through funding from IVADO Fundamental Research grant PRF-2019-5178609901 and NSERC Discovery grant RGPIN-2023-05705.

References

[Bonlarron and Régin, 2024] Alexandre Bonlarron and Jean-Charles Régin. Markov constraint as large language model surrogate. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 1844–1852. ijcai.org, 2024.

[Bonlarron *et al.*, 2023] Alexandre Bonlarron, Aurélie Calabrèse, Pierre Kornprobst, and Jean-Charles Régin. Constraints first: A new mdd-based model to generate sentences under constraints. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 1893–1901. ijcai.org, 2023.

[Choi *et al.*, 2021] Kyoyun Choi, Jonggwon Park, Wan Heo, Sungwook Jeon, and Jonghun Park. Chord conditioned melody generation with transformer based decoders. *IEEE Access*, 9:42071–42080, 2021.

[Deutsch *et al.*, 2019] Daniel Deutsch, Shyam Upadhyay, and Dan Roth. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 482–492, 2019.

[Dragone *et al.*, 2021] Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In Artur S. d'Avila Garcez and Ernesto Jiménez-Ruiz, editors, *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021*, volume 2986 of *CEUR Workshop Proceedings*, pages 6–14. CEUR-WS.org, 2021.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Huang *et al.*, 2018] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[Jaques *et al.*, 2017] Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, José Miguel Hernández-Lobato, Richard E. Turner, and Douglas Eck. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1645–1654. PMLR, 2017.

[Jelinek *et al.*, 2005] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 08 2005.

[Lafleur *et al.*, 2022] Daphné Lafleur, Sarath Chandar, and Gilles Pesant. Combining reinforcement learning and constraint programming for sequence-generation tasks with hard constraints. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Lattner *et al.*, 2018] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints. *Journal of Creative Music Systems*, 2:1–31, 2018.

[Lee *et al.*, 2019] Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime Carbonell. Gradient-based inference for networks with output constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4147–4154, 2019.

[Lin, 1991] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Inf. Theory*, 37(1):145–151, 1991.

[Manibod, 2022] Virasone Manibod. Ajout de structure aux modèles génératifs de séquences avec la programmation par contraintes. Master's thesis, Polytechnique Montréal, August 2022. https://publications.polymtl.ca/10495/.

[Marra *et al.*, 2024] Giuseppe Marra, Sebastijan Dumancic, Robin Manhaeve, and Luc De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artif. Intell.*, 328:104062, 2024.

[Parikh *et al.*, 2016] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

[Pearl, 1982] Judea Pearl. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In David L. Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982.*, pages 133–136. AAAI Press, 1982.

[Pesant, 2004] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

[Pesant, 2019] Gilles Pesant. From support propagation to belief propagation in constraint programming. *Journal of Artificial Intelligence Research*, 66:123–150, 2019.

[Régin *et al.*, 2024] Florian Régin, Elisabetta De Maria, and Alexandre Bonlarron. Combining constraint programming reasoning with large language model predictions.

In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Rossi *et al.*, 2006] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[Simonetta *et al.*, 2018] Federico Simonetta, Filippo Carnovalini, Nicola Orio, and Antonio Rodà. Symbolic music similarity through a graph-based representation. In Stuart Cunningham and Richard Picking, editors, *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion, Wrexham, United Kingdom, September 12-14, 2018*, pages 26:1–26:7. ACM, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Weininger, 1988] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[Yang and Lerch, 2020] Li-Chia Yang and Alexander Lerch. On the evaluation of generative models in music. *Neural Computing and Applications*, 32(9):4773–4784, 2020.

[Yin *et al.*, 2024] Chao Yin, Quentin Cappart, and Gilles Pesant. An improved neuro-symbolic architecture to finetune generative AI systems. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II*, volume 14743 of *Lecture Notes in Computer Science*, pages 279–288. Springer, 2024.

[Young *et al.*, 2022] Halley Young, Vincent Dumoulin, Pablo S Castro, Jesse Engel, and Cheng-Zhi Anna Huang. Compositional steering of music transformers. *Intelligent User Interfaces*, 2022.

[Yu *et al.*, 2023] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. A survey on neural-symbolic learning systems. *Neural Networks*, 166:105–126, 2023.

[Zaken *et al.*, 2021] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[Zhang *et al.*, 2024] Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.