Expressing Musical Ideas with Constraint Programming Using a Model of Tonal Harmony

Damien Sprockeels, **Peter Van Roy**
UCLouvain, Louvain-la-Neuve, Belgium
{damien.sprockeels, peter.vanroy}@uclouvain.be

Abstract

The realm of music composition with artificial intelligence stands as a pertinent and evolving field, attracting increasing interest and exploration in contemporary research and practice. This paper presents a constraint-programming based approach to generating four-voice diatonic chord progressions according to established rules of tonal harmony. It uses the strength of constraint programming as a formal logic to rigorously model musical rules and to offer complete control over the set of rules that are enforced. This allows composers to iteratively interact with the model, adding and removing constraints, allowing them to shape the solutions according to their preferences. We define a constraint model of basic tonal harmony, called Diatony. We show that our implementation using the Gecode solver finds optimal solutions in reasonable time and we show how it can be used by a composer to aid in their composition process.

1 Introduction

Music composition and artificial intelligence have always been seen as a promising combination. Attempts at using artificial intelligence to compose music can be dated as far back as the 1970's [Smoliar, 1972; Meehan, 1979]. With the development of increasingly more powerful AI techniques, musical applications have sprouted in many different areas, including music generation [Dong *et al.*, 2018; Mao *et al.*, 2018], signal treatment [Purwins *et al.*, 2019], performance assisting tools [Wu *et al.*, 2021] and many more [Nam *et al.*, 2018; Zhang, 2021]. The application of deep learning to music generation has been studied [Briot *et al.*, 2020] and its strengths and weaknesses have been assessed [Briot and Pachet, 2017; Casini *et al.*, 2018].

Constraint programming applied to music generation has also been studied [Pachet and Roy, 2001; Anders and Miranda, 2011; Pachet and Roy, 2014], though literature is less abundant on the subject. Machine learning is often preferred because of its ability to absorb large amounts of data and to produce musical material exhibiting common properties. Constraint programming, on the other hand, has the advantage of precisely respecting a set of given musical rules, and the model can easily be modified without requiring a new training phase. It does require more computation at run time, but this is not prohibitive. It is therefore an appealing concept for computer-aided composition as it gives full control to composers. Recently, [Sprockeels *et al.*, 2023] have completely formalized two-voice counterpoint in the style of Johann Fux. They also developed a simple interface giving composers control over the rules applied, which gives an indication of the potential that a constraint formalization can provide to composers.

This paper explores the use of constraint programming for a general computer-aided composition tool for composers. To alleviate the time issue, music generation is divided into four steps, starting from the overall structure, to general harmony, voice movement, and finally to ornaments. This paper focuses on the generation of a four-voice diatonic harmonic chord progression, taking as input chord names that can be generated using [Pachet and Roy, 2014], machine learning or coming directly from the composer, and develops them into four voices respecting the general rules of tonal music. In this way, it aims to provide an efficient working tool to generate tonal harmonic progressions, that can serve as a basis to create complete musical pieces.

1.1 Musical (De)composition

When composing music, it is natural to decompose the process in steps. Composers do not write a full musical piece from left to right, they work in steps with each a finer grain. It is hence natural that we follow a similar approach when generating music with rules. This paper identifies four steps:

- Defining the global structure of the piece (Sonata, verse-chorus, counterpoint, etc.). It defines the "shape" of the musical piece.
- Defining the harmonic evolution of the piece. On a large scale, this defines the tonality(ies) of the piece, and on a smaller scale, the harmonic development and rhythm [1] and modulations.
- Defining the voicings of the chords as well as the movement of the different voices. Once the harmonic development is known, it is important to consider how each chord is voiced and how voices move over time.

[1]The harmonic rhythm is different from the melodic rhythm, and defines how the harmony evolves over time.

- Adding ornaments. Once the harmonic development and general voicing is known, composers add ornaments, variations, and other stylistic elements to make the piece complete.

This paper focuses on the third step, namely the voicing of the chords and the evolution of voices over time. It thus assumes that the structure and harmonic evolution of the piece are known. Models exist to generate the harmonic evolution of the piece, such as [Pachet and Roy, 2014] that generates leadsheets and [Shukla and Banka, 2018] that generates midifiles. Of course, composers can also use their original ideas.

1.2 Contributions and Structure of the Paper

This paper has two main contributions. First, a formalization of diatonic harmonic rules based on [Duha, 2016] and [Gauldin, 2004], that defines how chords should be voiced in the case of a four-voice composition. Second, a constraint-based model implementing this formalization called *Diatony*, allowing composers to generate chord progressions that respect the rules of tonal music. It is designed to give as much freedom to composers as possible. They can interact with Diatony by controlling the costs and adding constraints. This tool is intended to be one part of a creation process, where the tool support for the whole process is still work in progress.

The paper is organized as follows:

- Section 2 compares constraint programming to deep learning and justifies our use of constraint programming.
- Section 3 defines the model of basic tonal harmony.
- Section 4 explains the strategy used to search for the optimal solution.
- Section 5 gives concrete examples of how a composer can use the model to write music.
- Section 6 summarizes our contributions and gives insight on future work.

This paper assumes some knowledge of music theory. To aid comprehension, we have added footnotes to explain the most important musical concepts.

2 CP Versus ML

Much of the recent work on musical generation using artificial intelligence has used a learning based approach. This section compares constraint programming (CP) to machine learning (ML) and explains the differences (and complementarity) between the two approaches.

CP is a powerful approach to solve complex combinatorial problems that is based on formal logic. Most CP systems use first-order logic, which defines logical sentences using predicates, quantifiers, and variables that range over a domain of discourse, supported by a proof theory and a model theory. Proof theory shows how to do formal inferencing and model theory gives a mathematical structure in which the axioms and their inferred consequences are true. For this paper, the mathematical structure is a space of musical solutions that satisfy musical rules. Constraint programming is complementary to current approaches for ML such as LLMs. CP has been under intense development for more than 40 years and is widely used in practice [Van Hentenryck, 1989; Nethercote *et al.*, 2007; Verhaeghe *et al.*, 2020; Kizilay *et al.*, 2020; Lam *et al.*, 2022; Delecluse *et al.*, 2022]. CP is based on a search for solutions in a space defined by logical relations (called 'constraints'). This search is highly optimized with sophisticated inferencing algorithms and heuristics. An important property of the search is that it is *complete*: it searches the whole space for solutions. With respect to music composition, completeness gives CP a form of creativity, i.e., it can find nonobvious solutions that are implied by the musical ideas. It is not limited by a training dataset and can generate all valid musical solutions to a problem. The price paid for this power is that search can be computationally expensive.

ML differs from CP in that it relies on the mathematical theory of statistics. The model is trained with a dataset, and uses the gathered knowledge to solve new problems. ML is widely used for text, music, and image generation and is highly flexible in the outputs it can generate. The generated content heavily depends on the training data.

ML and CP each in its own way requires a base of knowledge to be practically useful. ML creates its base during a training phase using prepared data. CP uses a base of knowledge encoded in logical form. Music theorists have long defined musical knowledge as rules that can be incorporated into a CP solver [Tagg, 1982; Gauldin, 2004; Duha, 2016], and examples of constraint formalizations of musical rules are numerous [Truchet and Codognet, 2004; Anders and Miranda, 2011; Sandred, 2021; Sprockeels *et al.*, 2023]. There is also ongoing work to add logical rules and inferencing to ML, combining the flexibility of ML with the reasoning ability of CP [Hadjeres and Nielsen, 2020; Cappart *et al.*, 2021; Kotary *et al.*, 2021; Ignatiev *et al.*, 2022; Tsouros *et al.*, 2023]. This is an important direction for ML research, but it is out of scope for the present paper.

3 Constraint Model of Tonal Harmony

This section defines a basic model of tonal harmony that we use for composing tonal music. Our model, called *Diatony*, defines a four-voice texture with diatonic triads and dominant seventh chords including their inversions[2]. The model contains harmonic and melodic rules as well as preferences to create diatonic chord progressions. It contains rules for note occurrence in chords, as well as rules for how voices move between chords. The model is based on a selection of rules from two treatises on music theory. Together they give a complete and coherent model of basic harmony that holds for the majority of tonal music. Future work will extend this model to support modulations as well as non-diatonic chords. From [Duha, 2016] we take the chapters on 3- and 4-note chords and inversions (*Accords de 3 et 4 sons*). From [Gauldin, 2004], we take the chapters on four-voice texture, diatonic harmony, dominant seventh, and inversions (chapters 5-11, 16, and 17).

[2]Four-voice texture is a common technique used to represent harmony. There are four distinct voices, namely the bass, tenor, alto and soprano, from lowest to highest, that play chords. Diatonic chords are chords formed with notes from the given tonality.

3.1 Rules and Preferences of the Diatony Model

The Diatony model consists of two parts: strict rules, that have to be followed at all times, and preferences, that may or may not be followed. This section gives a complete list of the musical rules of the Diatony model. Mathematical definitions for a subset of the strict rules are given in section 3.3.[3]

Strict Rules

- Voices should be in their assigned range and chords should have the right root, quality and state (**H1**, **H2** and **H3**).
- In fundamental state chords, the bass should be doubled (**H4**).
- From a chord in fundamental state to another, common notes should be kept in the same voice and other voices should move to the closest note (**P4**, **P5**).
- If there are no common notes between two fundamental state chords of successive degrees, i.e. fourth degree and fifth degree, upper voices should move in contrary motion to the bass (**M1**).
- It is forbidden for the harmonic interval between two voices to be a perfect fifth or a perfect octave in two successive chords, unless the notes are the same (**M2**).
- When the tritone[4] of the tonality is present in a chord, it must be resolved in the next chord. The voice playing the fourth degree of the scale should move down by step to the third degree of the scale, and the voice playing the leading tone should move up by step to the tonic (**M3**).
- In the case of an interrupted cadence, if the mode is minor or if the leading tone is in the soprano, the leading tone must move up by step to the tonic and the other voices should move down. The third of the chord must be doubled for the sixth degree chord instead of the fundamental (**M4**, **H6**).
- For chords in first inversion, each note should be present once and any tonal note[5] can be doubled. If the bass is a tonal note, it can also be doubled (**H7**).
- If the bass and the soprano move by step in contrary motion from a chord in fundamental state to a first inversion chord to a fundamental state chord, the bass can be doubled in the second chord even if it is not tonal (**H9**).
- For the seventh degree diminished chord in first inversion, the fundamental should be doubled (**H7**).
- For chords in second inversion, the bass should be doubled (**H8**).
- If a chord in second inversion is the appogiatura of the fifth degree chord, the tonic should be approached by contrary or oblique motion. The third degree and tonic should move down by step (**M5**).
- If a diminished chord is in second inversion, the third of the chord should be doubled (**H10**).
- In perfect cadences, one of the chords must be incomplete[6]. If the fifth degree chord is incomplete, its bass should be doubled. If the first degree chord is incomplete, the bass should be tripled (**H5**).
- If a dominant seventh chord is in second inversion, the tritone resolution can be altered so that the fourth of the scale moves up by step instead of down by step (**M3**).

Preferences

We define the following preferences, in decreasing order of importance. An explanation of their use is given in section 3.4.

- Chords should be complete (**P1**).
- Diminished chords in fundamental state should be used with three voices instead of four (**P2**).
- Chords should have four different note values[7] (**P3**).
- Melodic intervals should be small (**P4**).
- Common notes should be kept in the same voice (**P5**).

3.2 Variables

The main array of variables contains the notes for each voice in each chord. It has the following definition, where $i \in [0, n-1]$ and $v \in [0, 3]$. The domain is the range of MIDI values[8]. i is the index of a chord, v the index of a voice and n the number of chords.

$$N[i][v] \in [0, 127] \tag{1}$$

Additionally, arrays have been defined for the melodic intervals in each voice and for harmonic intervals between each pair of voices. For melodic intervals, $i' \in [0, n-1[$ as melodic intervals are defined with respect to the first of the two successive chords. Harmonic intervals are defined for each pair of different voices: $v_1, v_2 \in [0, 3]$ where $v_1 < v_2$. They do not have a direction.

$$M[i'][j] \in [-12, 12] \tag{2}$$
$$H[i][0][v_2] \in [0, 12 \times v_2 + 7] \tag{3}$$
$$v_1 > 0 : H[i][v_1][v_2] \in [0, 12 \times (v_2 - v_1)] \tag{4}$$

Additional variables have been defined to model preferences. For brevity reasons, they will not be listed here but there is a variable for each cost detailed in section 3.4.

[3]The complete mathematical formalization of the rules can be found here: http://hdl.handle.net/2078.1/287476.

[4]The tritone in a tonality is perhaps the most important notion in tonal harmony. It is the interval between the fourth degree and the leading tone, which is a half step below the tonic. It is highly dissonant and demands to be resolved.

[5]Tonal notes are the first, fourth and fifth degree of the scale.

[6]A chord is complete if all the pitch classes that make the chord are present at least once. If a note is not present, it should be the fifth of the chord.

[7]Not to be confused with having four different notes. This preference states that one note can be present more than once, but it should not be in the same octave.

[8]MIDI notation maps an integer to each note of a keyboard. Middle C (C4) has the value 60, and a semitone corresponds to a difference of 1.

3.3 Constraint Formalization

This section shows the constraint formalization of four of the rules defined in section 3.1. We consider two main types of constraints: harmonic constraints, that regard the notes in a given chord; and melodic constraints, that regard transitions between consecutive chords as well as voice leading. Examples of each type of constraint are presented below. Our first example is a harmonic rule chosen for its simplicity, to introduce our mathematical notation. The second example formalizes a more complex harmonic rule that is an exception to the first rule. The third example shows tritone resolution, which is one of the most important melodic rules. The fourth example is another important melodic rule that forbids parallel fifths and octaves.

Harmonic Constraints

Harmonic constraints are constraints on simultaneous notes. They typically dictate how many times each note of the chord should be present. For example, for perfect chords in fundamental state, the general rule states that the fundamental of the chord, i.e. the note it is based on, should be doubled. The third should be present once, and the fifth should be present at most once. This can be expressed mathematically as follows:

$$\forall i \in [0, n-1], \forall v \in [0, 3]$$
$$|\{v \mid N[i][v] \bmod 12 = C_{\mathit{fund}}\}| \geq 1 \tag{5}$$
$$|\{v \mid N[i][v] \bmod 12 = C_{\mathit{third}}\}| = 1 \tag{6}$$
$$|\{v \mid N[i][v] \bmod 12 = C_{\mathit{fifth}}\}| \leq 1 \tag{7}$$

Here i represents a chord's position, $N[i]$ represents the notes of a chord, 12 represents an octave, C_{fund}, C_{third}, and C_{fifth} represent the fundamental, third and fifth of the chord respectively. For example, if the chord is C major, then the note C should be present at least once, the note E (a third above) should be present once, and the note G (a fifth above C) should be present at most once.

Another exception to this rule is that in the case of a perfect cadence, if the fifth degree chord is a dominant seventh chord, it is impossible to have all the different notes in both chords. This is due to the constraint forbidding parallel fifths and the constraint for tritone resolution. Hence, one of the chords has to be incomplete. In that case, if the fifth degree chord is incomplete, the fundamental should be present twice. On the other hand, if the first degree chord is incomplete, the fundamental should be present three times and the third once. The following constraints are added, where $N[i']$ is the dominant seventh and $N[i'+1]$ is the tonic chord (first degree).

$$\forall i', \forall v$$
$$|\{v \mid N[i'][v] \bmod 12 = C_{\mathit{fund}}\}| \geq 1 \tag{8}$$
$$|\{v \mid N[i'][v] \bmod 12 = C_{\mathit{third}}\}| = 1 \tag{9}$$
$$|\{v \mid N[i'][v] \bmod 12 = C_{\mathit{fifth}}\}| \leq 1 \tag{10}$$
$$|\{v \mid N[i'][v] \bmod 12 = C_{\mathit{seventh}}\}| \leq 1 \tag{11}$$
$$|\{v \mid N[i'+1][v] \bmod 12 = C_{\mathit{fund}}\}| \leq 1 \tag{12}$$
$$|\{v \mid N[i'+1][v] \bmod 12 = C_{\mathit{third}}\}| = 1 \tag{13}$$
$$|\{v \mid N[i'+1][v] \bmod 12 = C_{\mathit{fifth}}\}| \leq 1 \tag{14}$$

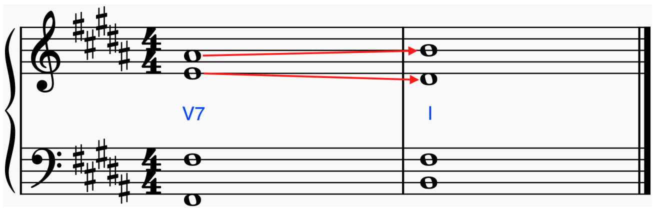

Figure 1: Tritone resolution in a perfect cadence in B major.

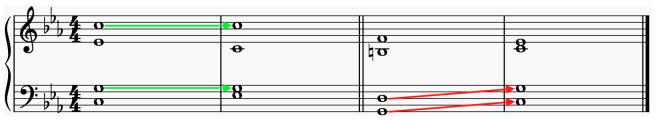

Figure 2: Examples of parallel fifths: allowed (left) and forbidden (right) in C minor.

Melodic Constraints

Melodic constraints are constraints on successive notes. One good example of melodic rules is that when the tritone of the tonality is present in one chord, it should resolve in the next chord. That means that the voice playing the fourth degree of the scale should move down by step to the third, while the voice playing the leading tone should move up by step to the tonic. This situation typically happens when one chord is the dominant seventh version of the fifth degree and the next chord is the first degree. Figure 1 shows the tritone resolution in a perfect cadence in B major. This rule can be expressed mathematically as follows:

$$\forall i' \in [0, n-1[, \forall v \in [0, 3]$$
$$N[i'][v] \bmod 12 = T_{\mathit{fourth}} \implies M[i'][v] \in \{-1, -2\} \tag{15}$$
$$N[i'][v] \bmod 12 = T_{\mathit{leading\text{-}tone}} \implies M[i'][v] = 1 \tag{16}$$

Here *i* represents a chord's position, *v* represents a voice's index, T_{fourth} is the fourth of the tonality, -1 and -2 represent a descending minor and major second respectively, $T_{\mathit{leading\text{-}tone}}$ represents the leading tone, and 1 represents an ascending minor second. Melodic constraints are applied with respect to the first chord.

Another example of a melodic constraint is that parallel fifths and octaves are forbidden. That is, if the harmonic interval between two voices in a chord is a perfect fifth or a perfect octave, then the harmonic interval between these voices cannot be the same in the next chord. In other words, the melodic intervals of the concerned voices between this chord and the next cannot be equal. This is however allowed if both voices play the same note in both chords. Figure 2 gives an example of valid and invalid parallel fifths. This rule can be mathematically expressed as follows:

$$\begin{aligned} \forall v_1, v_2 \in [0, 3], v_1 < v_2 \quad \forall i' \in [0, n-1[\\ (H[i'][v_1][v_2] \quad \bmod 12 \in \{7, 12\}) \wedge \\ (N[i'][v_1] \neq N[i'+1][v_1] \vee N[i'][v_2] \neq N[i'+1][v_2]) \\ \implies H[i'+1][v_1][v_2] \neq H[i'][v_1][v_2] \end{aligned} \tag{17}$$

3.4 Preferences

Preferences are guidelines that should generally be followed unless another rule forbids it. Preferences can be seen as levers that a composer can use to customize the model according to their own ideas. This also means that if they want to enforce a behaviour that goes against the default preferences, solutions can still be found. In Diatony, the preferences have a default behaviour that corresponds to the rules of tonal harmony, but they can easily be modified according to personal choice. We represent preferences as costs that are minimized in a lexicographic order. This means we have to rank them in order of importance. Some costs have a harmonic nature, and some have a melodic nature. The list of preferences in decreasing order of importance is given in section 3.1.

The first three preferences are harmonic preferences. The number of incomplete chords is minimized first as it is determined by the input and does not depend on the values that variables take. The second most important preference is that chords should contain four different note values (i.e. they can contain the same note twice, but not in the same octave). There is an exception to this rule when the chord is diminished; that is why this preference is put first.

The last two preferences are melodic, and they depend on the harmonic preferences. Since the harmonic preferences are more important, they are minimized before the melodic preferences. The first melodic preference states that smaller melodic intervals should be preferred. This preference does not take into account the quality of the intervals, i.e. a minor second is as desirable as a major second. As a result, we do not just sum the absolute value of intervals as that would imply that minor intervals are preferred to major intervals. Each interval is assigned a weight. By default, unisons have a cost of 0; seconds have a cost of 1; thirds have a cost of 3; fourths, tritones and fifths have a cost of 6; sixths have a cost of 12; sevenths have a cost of 18; and octaves have a cost of 6. These values are somewhat arbitrary, but are designed so that bigger intervals have heavier weights. The only exception is the octave. The final preference states that common notes should be kept in the same voice when possible. It is after the previous preference because it causes a great increase in efficiency, and because it does not affect solution quality.

3.5 Implementation

Diatony is implemented as a constraint model using the Gecode solver [Gecode Team, 2019] [9]. Gecode was chosen for its efficiency and extensive functionality that gives a lot of versatility for modeling the problem. The constraint problem takes as input the set of chords with their quality and state, and can produce output in various ways. It can print to a file or command line the solutions it finds during search, but it is more practical to use it to generate MIDI files containing the solution. This allows for an easy integration to a composers' work environment, such as DAWs like LogicPro [Apple Inc., 1993] or Ableton [1999] as well as sheet music editing software like Sibelius [Avid Technology, Inc., 1993] or MuseScore [2002].

[9]The full code can be found at https://github.com/sprockeelsd/Diatony/tree/IJCAI2024

4 Search for Solutions

The search space grows rapidly in function of the length of the musical piece because the Diatony model gives only the piece's basic musical structure. The intention is that the composer adds musical ideas to this structure to restrict the search space and converge quickly to a desired musical result. This both gives freedom to the composer and makes the search efficient. Formally, musical ideas are added to the model by managing preferences and adding custom constraints. For the search algorithm, we have chosen to use a restart-based approach over a branch-and-bound solver when exploring the search tree. The branch-and-bound solver allows to find optimal solutions by adding a constraint every time a solution is found, and the restart-based approach allows to reduce the space of the search at every restart. The efficiency of our approach is sufficient to make the model usable for composers.

4.1 Branching Heuristics

Since all the rules are small scale, their variables tend to be similarly constrained. This means that approaches based on accumulated failure counts, domain size, or degree do not work well for this problem. The approach we have chosen is similar to how a composer would approach the problem: start at the end and work on each chord successively. The solver has an advantage that composers do not have, namely propagation, which means that as the notes of a chord are decided, the possible values for the yet unbound notes are updated. We have chosen to work from the end, because starting from the beginning is not as efficient. Starting from the end allows the solver to know where it has to go, whereas starting from the beginning may lead to a situation where resolution is not possible and the solver has to backtrack significantly. For value selection, the approach we have chosen is to randomly select a value from the domain. Smarter value selection would be possible, such as branching on the melodic intervals and choosing the smallest in absolute value, or choosing values based on how many times a note should occur in a chord, but these are computationally expensive and random value selection is efficient enough for problems of reasonable size (typically a musical phrase).

4.2 Restart-Based Search

Exploring the whole search tree to ensure optimality tends to be very computationally expensive for problems of moderate size. To improve performance, we decided to use a restart-based approach. In particular, the solver uses a cutoff generator to restart the search after a number of failures. For restart to be useful, the solver deduces no-goods, that is combinations of variable assignations that lead to no solutions, before restarting. This allows the solver to learn more about the specific instance of the problem over time. For the cutoff generation, we have decided to use a hybrid of two approaches: a linear generator with scale factor $2n$, and a geometric generator with base 2 and scale factor $16n^2$, where n is the number of chords. These values have been shown empirically to give good results. For the two cutoff sequences $l_0, l_1, l_2, ...$ and $g_0, g_1, g_2, ...$, the combined cutoff sequence used by the solver is $l_0, g_0, l_1, g_1, ...$ This approach gives significantly

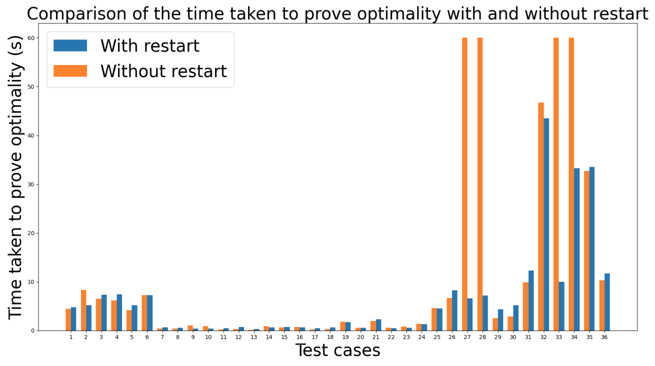

Figure 3: Comparison of the time taken for the search to prove optimality with and without restart. Test cases exceeding 60 seconds have been cropped for better readability, but test cases 33 and 34 reached the time out of 450s while test case 27 took 315s and test case 28 took 206s.

better results than each of them separately. For the no-good generation, we have to specify a maximal depth of the tree at which the no-goods can be generated. The bigger the depth, the more no-goods can be deducted but they will also likely be less generic, and it will require more memory. We have empirically shown that the value $4n$ is a good compromise for our problem. Smaller values resulted in worse performance, while higher values did not provide significant improvement.

Figure 3 compares the time taken to prove optimality for a set of test cases[10] between the approach using restarts and the approach exploring the search tree without restarting for each test case. The test cases have been chosen to have variable sizes, contain different chords and different tonalities. As we can see, using restarts with no-goods significantly reduces the amount of time necessary to prove optimality for harder instances. This is due to the fact that the search space becomes smaller each time the solver is restarted. On easier problems, this approach tends to take slightly more time than exploring the search tree without restarting, but the difference is minimal.

4.3 Convergence of Solution Quality

Figure 4 shows the evolution of the cost for melodic intervals over time for a variety of test cases. This cost is shown instead of others as it is the one that varies the most and is the bottleneck of the search. The jumps indicate that a lower value for one of the more important costs has been found (see section 3.1 for a list of the preferences in order). It is clear that this cost tends towards optimum very fast for most cases, and optimality is proven for all test cases. Once additional constraints are added, the time required to prove optimality drastically decreases. Three test cases behave very differently from the others. They occur with the chord progression I5-V6-VI5-V5-IV5-I6-II5-V5-I5-V6-VI5-V5-IV5-V7+-I5[11] in B♭, C and C♯ minor. This does not seem to be linked to the fact that the tonalities are minor, as it does not happen for

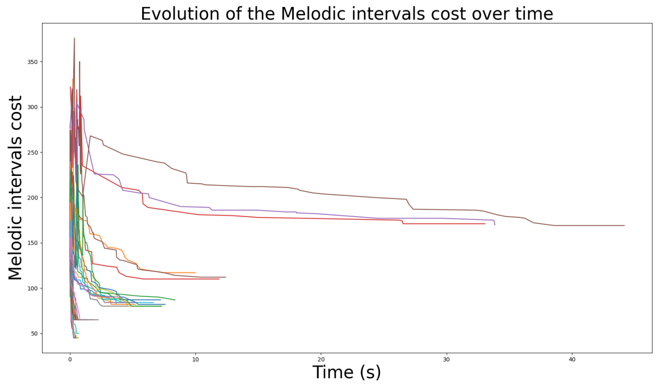

Figure 4: Evolution of the melodic cost of solutions over time.

other chord progressions. It does not seem to be linked to the size of the problem either, as there are bigger problems in the same tonalities and the search is completed long before. It is still unclear why this happens on these particular instances.

5 How a Composer Can Use Diatony

This section illustrates how composers can use Diatony in their composition process. Though Diatony is not intended to be a standalone tool for composers but rather a part of a more complete tool, it can still be useful to get an idea of the harmonic development of a musical piece. Composers can interact with the initial solution by identifying parts they want to keep or change, and run the solver in an iterative process to shape the original solution into their musical idea. They can also add or remove constraints dynamically. The first example is chosen to illustrate how a composer would start with a musical idea, run the solver to get a first suggestion, and then iteratively add constraints until the result is satisfactory. The second example is a more ambitious example of composition, where the composer wants to create an accompaniment for a pop song, for a singer to then improvise on.

5.1 First Example: Descending Bass Line

The first example is based on a descending bass line, like in the iconic "Piano Man" by Billy Joel. One of the possible chord progressions that can fit onto that bass line is the following: I5-V6-VI5-V5-IV5-I6-II5-V5-I5-V6-VI5-V5-IV5-V7+-I5. If we run the solver for this chord progression, and choose the tonality of the piece to be E major, the four-voice solution generated by the solver is depicted in figure 5. This is (one of) the optimal solutions for the problem.

In this example, the bass has the most interesting melodic line. It was our intention, and we designed the chord progression to allow that to happen. Nonetheless, the solver came up with that bass line on its own, the only input provided was the chords and their states. The soprano's melodic line is not ideal, as it lacks intent. This is because it is the composer's job to add constraints, Diatony just ensures that the rules of tonal music are followed. To remedy that, we add the constraints that the soprano's melody should start and end on a G♯. Indeed, it is fairly common that the first degree chord in

[10]The test cases can be found in the technical appendix.

[11]The Roman numeral gives the degree of the scale the chord is based on, and the decimal numeral refers to the state of the chord. *5* means it is in fundamental state, *6* means it is in first inversion and *7+* means it is a dominant seventh chord in fundamental state.

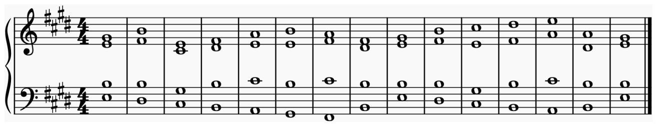
Figure 5: First solution generated by the solver for the chord progression I5-V6-VI5-V5-IV5-I6-II5-V5-I5-V6-VI5-V5-IV5-V7+-I5 in E major. It can be listened to here: https://on.soundcloud.com/4tpXL

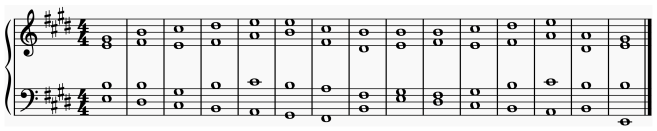
Figure 6: Solution generated by the solver after additional constraints have been posted (starting and ending the soprano melody on a G♯). It can be listened to here: https://on.soundcloud.com/4uTFF

fundamental state has the third of this chord in the soprano, and this will also allow for the tritone resolution in the perfect cadence at the end of the chord progression to be heard more clearly. The solution generated by the solver with those additional constraints is shown in figure 6.

5.2 Second Example: Composing a Pop Song

For the second example, we will start from the basics again, i.e., the constraints we added for the first example are removed. We now want to write a pop song, so we will first write an accompaniment for the verses and then for the chorus. We will not write a melody here, as this composition is intended to be improvised on by a singer.

For the verses, we will use the following chord progression: I5-V5-VI5-I5-III5-VI5-II5-I5-V5 in A♭ major. We will change the harmonic rhythm, so some chords will play for longer than others. The last chord of the progression is the fifth degree to create tension leading into the chorus. The first solution generated by the solver for this chord progression is shown in figure 7.

We are happy with this solution, so we will keep it and use it for the first verse. We will also create a variation for the second verse by making the rhythm more complex. We will add an alternating rhythm between the two inner voices to create a sense of motion, and passing notes linking the different sections of the verses. The result for the second verse is shown in figure 8.

Now that we are satisfied with the accompaniment for our verses, we will work on the chorus. The chord progression we will use is a variation of the famous I-V-VI-IV chord progression as a loop, starting on the fourth degree and ending on the sixth degree for a darker feel. We want the notes to be fairly low to emphasize the darker feel as well. To make the loop feel more natural, we will add a passing V6 chord between the VI and the IV chords. We want the melody at the soprano to go down by step between the VI5, V6 and IV5 chords so we add a constraint accordingly. With all these additional constraints, the solution returned by the solver is shown in figure 9. The combination of the verses and chorus can be listened to here: https://on.soundcloud.com/gQ6v9.

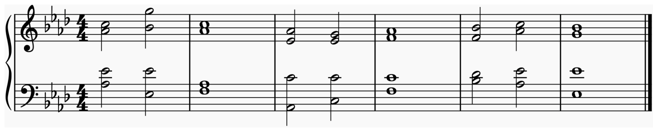
Figure 7: First solution returned by the solver for the verses' accompaniment. The rhythm is different because the harmonic rhythm we gave to our problem is different, it is not generated by the solver. It can be listened to here: https://on.soundcloud.com/Ga1VP

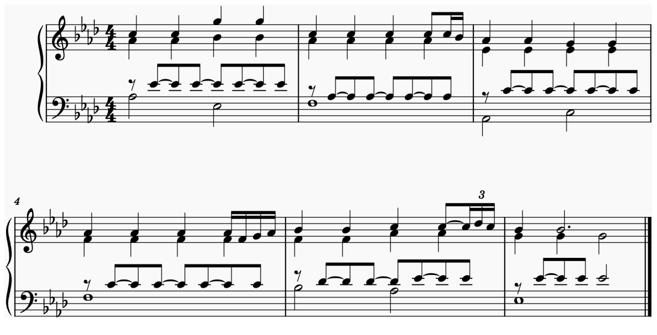
Figure 8: Variation of the first verse with a more intricate rhythmic pattern. It can be listened to here: https://on.soundcloud.com/aPZir

6 Conclusion

This work provides two contributions. On the one hand, it provides a formalization of diatonic tonal harmonic rules that defines how chords should be voiced in four-voice composition. On the other hand, it provides a constraint model implementing this formalization that can be used by composers to generate chord progressions. It has the advantages of being highly controllable and highly permissive, allowing composers to build their ideas iteratively.

The model and implementation can be enlarged to include four-note chords other than dominant seventh chords, modulations, and the use of non-diatonic notes and chords as well as rhythmical aspects of composition. The implementation has been designed in a way that makes it easy to add new constraints. Additionally, as presented in section 1.1, we have decomposed the composition problem in different steps in the way a composer could do it. This paper only tackles one of these steps, and we believe constraint models for the other steps could greatly benefit composers. Finally, having a user-friendly interface allowing composers to interact with the model from a high level of abstraction would also be beneficial.

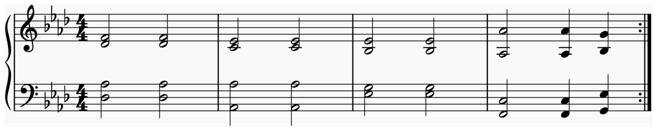
Figure 9: Chord loop generated by the solver for the chorus, with a passing chord at the end. It can be listened to here: https://on.soundcloud.com/QJEZz

References

[Ableton AG, 1999] Ableton AG. Ableton Live, 1999.

[Anders and Miranda, 2011] Torsten Anders and Eduardo R Miranda. Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR)*, 43(4):1–38, 2011.

[Apple Inc., 1993] Apple Inc. Logic Pro, 1993.

[Avid Technology, Inc., 1993] Avid Technology, Inc. Sibelius, 1993.

[Briot and Pachet, 2017] Jean-Pierre Briot and François Pachet. Music generation by deep learning-challenges and directions. *arXiv preprint arXiv:1712.04371*, 2017.

[Briot *et al.*, 2020] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep learning techniques for music generation*, volume 1. Springer, 2020.

[Cappart *et al.*, 2021] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.

[Casini *et al.*, 2018] Luca Casini, Gustavo Marfia, and Marco Roccetti. Some reflections on the potential and limitations of deep learning for automated music generation. In *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 27–31, 2018.

[Delecluse *et al.*, 2022] Augustin Delecluse, Pierre Schaus, and Pascal Van Hentenryck. Sequence variables for routing problems. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[Dong *et al.*, 2018] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Duha, 2016] Isabelle Duha. *L'harmonie en liberté: de la mémoire à l'improvisation*. Gérard Billaudot, Armiane Imp., 2016.

[Gauldin, 2004] Robert Gauldin. *Harmonic Practice in Tonal Music. Second Edition*. W. W. Norton and Company, Inc, 2004.

[Gecode Team, 2019] Gecode Team. *Gecode: Generic Constraint Development Environment*, 2019. Last accessed on 19/02/2023.

[Hadjeres and Nielsen, 2020] Gaëtan Hadjeres and Frank Nielsen. Anticipation-rnn: Enforcing unary constraints in sequence generation, with application to interactive music generation. *Neural Computing and Applications*, 32(4):995–1005, 2020.

[Ignatiev *et al.*, 2022] Alexey Ignatiev, Yacine Izza, Peter J Stuckey, and Joao Marques-Silva. Using maxsat for efficient explanations of tree ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3776–3785, 2022.

[Kizilay *et al.*, 2020] Damla Kizilay, Pascal Van Hentenryck, and Deniz T Eliiyi. Constraint programming models for integrated container terminal operations. *European Journal of Operational Research*, 286(3):945–962, 2020.

[Kotary *et al.*, 2021] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021.

[Lam *et al.*, 2022] Edward Lam, Guy Desaulniers, and Peter J Stuckey. Branch-and-cut-and-price for the electric vehicle routing problem with time windows, piecewise-linear recharging and capacitated recharging stations. *Computers & Operations Research*, 145:105870, 2022.

[Mao *et al.*, 2018] Huanru Henry Mao, Taylor Shin, and Garrison Cottrell. Deepj: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 377–382. IEEE, 2018.

[Meehan, 1979] James R Meehan. An artificial intelligence approach to tonal music theory. In *Proceedings of the 1979 annual conference*, pages 116–120, 1979.

[MuseScore BVBA, 2002] MuseScore BVBA. MuseScore, 2002.

[Nam *et al.*, 2018] Juhan Nam, Keunwoo Choi, Jongpil Lee, Szu-Yu Chou, and Yi-Hsuan Yang. Deep learning for audio-based music classification and tagging: Teaching computers to distinguish Rock from Bach. *IEEE signal processing magazine*, 36(1):41–51, 2018.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[Pachet and Roy, 2001] François Pachet and Pierre Roy. Musical harmonization with constraints: A survey. *Constraints*, 6:7–19, 2001.

[Pachet and Roy, 2014] François Pachet and Pierre Roy. Imitative leadsheet generation with user constraints. In *ECAI 2014*, pages 1077–1078. IOS Press, 2014.

[Purwins *et al.*, 2019] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[Sandred, 2021] Örjan Sandred. Constraint-solving systems in music creation. *Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity*, pages 327–344, 2021.

[Shukla and Banka, 2018] Shipra Shukla and Haider Banka. An automatic chord progression generator based on reinforcement learning. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 55–59. IEEE, 2018.

[Smoliar, 1972] Stephen W Smoliar. Music theory—a programming linguistic approach. In *Proceedings of the ACM annual conference-Volume 2*, pages 1001–1014, 1972.

[Sprockeels *et al.*, 2023] Damien Sprockeels, Thibault Wafflard, Peter Van Roy, and Karim Haddad. A constraint formalization of Fux's counterpoint. *Actes des Journées d'Informatique Musicale 2023*, pages 68–77, 2023.

[Tagg, 1982] Philip Tagg. Analysing popular music: theory, method and practice. *Popular music*, 2:37–67, 1982.

[Truchet and Codognet, 2004] Charlotte Truchet and Philippe Codognet. Musical constraint satisfaction problems solved with adaptive search. *Soft Computing*, 8(9):633–640, 2004.

[Tsouros *et al.*, 2023] Dimos Tsouros, Senne Berden, and Tias Guns. Learning to learn in interactive constraint acquisition. *arXiv preprint arXiv:2312.10795*, 2023.

[Van Hentenryck, 1989] Pascal Van Hentenryck. *Constraint satisfaction in logic programming*. MIT press, 1989.

[Verhaeghe *et al.*, 2020] Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25:226–250, 2020.

[Wu *et al.*, 2021] Yusong Wu, Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, and Jesse Engel. Mididdsp: Detailed control of musical performance via hierarchical modeling. *arXiv preprint arXiv:2112.09312*, 2021.

[Zhang, 2021] Jingwen Zhang. Music feature extraction and classification algorithm based on deep learning. *Scientific Programming*, 2021:1–9, 2021.