**Constraint Programming: Applications for Modelling Music Theories**

**Harry Schultz**

**Abstract**

Constraint programming and the modelling of music theories go hand in hand. Traditionally musical theories are expressed by a set of rules which describe the intended musical result, which makes constraint programming’s declarative and modular approach perfect for writing music theory and composition systems. Various musical theories have been modelled using constraint programming practices, including harmony, rhythm, counterpoint, form and instrumentation. Since using constraint programming to model music theories from the ground up is a time consuming and difficult task, some generic music constraint systems have been introduced. The goal of this paper will be to introduce the basic concepts of constraint programming, introduce basic music theory, show how certain elements of music theory have already been modelled and the limitations of the current models, as well as give an overview of a generic music constraint system and its benefits and limitations.

**Introduction**

The concepts of computer programming and musical theory have been intertwined almost since the birth of programming with the first musical modeling system being developed in 1958 by Lejaren Hiller and Leonard Isaacson, who created the *Illiac Suite* which is regarded as the first score composed by an electronic computer. Part of the reason composers and programmers alike are intrigued by modelling music theories using computer programming is that programming and musical theory have much in common and certain programming practices are particularly complimentary to model musical theories. This begins with musical composition’s ability to be expressed by sets of rules, rules which describe the musical style of a piece or when teaching composition to others.

Compositional rules are perfect for describing music for a couple of reasons:

1. The rules are declarative (in that they do not specify procedurally how to create a specific musical result, but rather describe important features the result should have), and they describe the music in a modular way
2. The rules can help to express the multi-dimensionality of music in much easier terms. Things like rhythm, harmony, voice or instrumentation throughout a composition can be much easier to explain, instead of looking at a composition as a whole, and seeing it as small parts working together.

An example of this concept would be if you want to describe rules for harmony, the composer would say that a harmony consists of chords which in turn consist of, usually, three pitches that are within a certain range of each other. So by taking pitches as our base, we can give those pitches some rules to follow(there must be three and they must be within a given range of each other) and suddenly we have a chord, once we define how different chords interact, such as which chords should follow which others we have begun to define what harmony is.

Because compositional rules are such a well-established concept, programming approaches that support rule-like programming constructs have been invaluable to people interested in modelling music theories using computers. These approaches translate the advantages of compositional rules into the world of computer programming: implementing music theory models becomes declarative and modular. It is very difficult to computationally model music theories by a procedural programming approach where the programmer details *how* to obtain a certain result. In addition, changing or adding a single rule can require redesigning the entire procedural program. These difficulties are shared by object-oriented programming well. By contrast, rule-based approaches free programmers to concentrate on what they want to do in a musical sense; they do not need to define how to achieve this outcome.

Another advantage of using rule-based approaches lies in the fact that music theory models are defined in a modular fashion. Multiple rules can even affect the same parameter value. For instance, a music theory model may restrict the note pitches of a score by melodic rules on the one hand and by harmonic rules on the other. Each of these separate rules affect the same parameter values, namely the pitches. However, no rule necessarily determines the parameter values fully. Search finds one or more solution that fulfils all rules.

Constraint programming has proven to be a particularly successful programming system for realizing ruled-based systems (another option is logic programming). The attraction of constraint programming is easily explained. Constraint programming allows users to model complex problems in a simple way. A problem is stated by a set of *variables* (unknowns) and *constraints* (relations) between these variables.

Applied to music composition, a compositional task is stated by:

1. A music representation in which some musical aspects are unknown – and therefore represented by variables
2. Compositional rules that impose constraints on these variables. For instance, a chord can be expressed by a set of notes, and the note pitches can be variables. Some harmonic constraints may specify how the chord pitches are related to each other, other constraints define the relation to the pitches of other chords and so forth. A musical constraint problem does not necessarily result in only a single solution. Instead, the restrictions and dependencies expressed by a set of constraints reduce the set of solution candidates.

Existing constraint programming systems can efficiently solve a constraint problem – a fact that has greatly contributed to the popularity of constraint programming.

**What is constraint programming?**

Constraint programming is a programming paradigm, which introduces techniques to solve constraint satisfaction problems. A *constraint satisfaction problem* (CSP) consists of a set of *variables* and mathematical relations between these variables which are referred to as *constraints*. Usually, a CSP presents a combinatorial problem. A *constraint solver* finds one or more solutions for the problem. A *solution* of a CSP shows for each variable of the problem a determined value, which is consistent with all constraints. A *constraint system* (e.g., a programming language supporting constraint programming) allows its user to define and solve CSPs.

A simple numeric example may illustrate these concepts. The example introduces the two variables *X* and *Y* and restricts their value by two basic arithmetical operations, connected by a conjunction. One possible solution for this problem is *X* = 4, *Y* = 5, another solution is *X* = 3, *Y* = 6:

*X* + *Y* = 9 ∧ *X < Y*

Note that the term *variable* has a clearly different meaning in mainstream programming paradigms and languages (e.g., C or Java) on the one hand, and in the field of constraint programming on the other hand. In mainstream programming languages, a variable denotes a *stateful* computational entity: such a variable always has a specific value and a program can alter the value of a variable with an assignment statement at any time. By contrast, in constraint programming the notion of a variable is more similar to the notion of a variable or *unknown* in mathematics. The value of a variable may be unknown or partially known. For example, it may only be known that *X* ∈ {1*...*10}. However, a variable never changes its value, it is *stateless*.

Sometimes, the term *constrained variable* will be used to explicitly denote a variable in the context of constraint programming. A constrained variable is a variable that has a domain that is a set of values it may take in a solution. Some constraint systems support variables with an infinite domain (e.g., the domain of all real numbers in some interval). The values in the domain are often all of the same type (e.g., Boolean domain, integer domain or domain of finite sets of integers), some systems support domains with mixed types.

In principle, constraints can be arbitrary mathematical relations. Examples include numerical relations, set relations, logic relations, and tree or graph relations. Constraint systems predefine a set of constraints and often allow the user to extend this set.

Solving a CSP requires searching. Because the search space – that is the set of (partial) solution candidates – of a CSP is often huge, an efficient constraint solver has great impact on the usability of a constraint system.

**Musical Constraint Satisfaction Problems**

A musical CSP implements a music theory model as a computer program: when the program is executed, it generates music that complies with the modelled theory. A music theory model implemented by a musical CSP must be fully formalized (fully expressible in mathematical notation). However, in the context of music constraint programming a music theory model does not necessarily need to be consistent with any existing musical style. For instance, a composer may develop some musical CSP (and implicitly define a theory model) in an ad-hoc manner in order to generate some subpart of a composition in a novel way.

Most of the present musical CSPs primarily constrain note pitches in some way. Constraint programming is highly suited for this specific task. Also, the rule-based approach of conventional music theories focusses on pitches. By contrast, it is often difficult to adequately address the complexity required to model pitch structures with other algorithmic composition strategies. Nevertheless, these strategies are well applicable for compositional tasks that are not pitch-specific, for example, to generate complex trajectories for parameters which control arbitrary sound synthesis details or the specialization of sounds. It therefore makes sense to complement these different strategies and to create different aspects of the music by different strategies.

Many music theories have been modelled and implemented by constraint programming. These will be further explored in the following subsections.

**Counterpoint**

Polyphonic music consists of multiple voices, which accompany each other. The practical training to write polyphonic music is traditionally called counterpoint. Over the centuries, many counterpoint textbooks were written. Different textbooks often cover different musical styles. Today, two style families are taught most frequently. One approach is oriented in Renaissance music and the composer Palestrina in particular. JohannJosephFux wrote on this approach in 1725 titled *Gradus ad Parnassum*, a treatise on counterpoint. Whereas in the first approach harmonic considerations are at best secondary, the other – and historically younger – approach teaches how to compose polyphonic music that expresses a harmonic progression. Baroque music, for example, usually follows this approach.

The 20th and 21st century saw further developments of polyphonic music. These are rarely covered by counterpoint textbooks, but nevertheless have been addressed in musical CSPs. There exist several systems creating polyphonic music by means of constraint programming. Scholastic counterpoint features a particular strict set of rules when compared with other music theory subdisciplines, for example, rhythm or form. A strict rule set makes formal modelling easier, which explains why counterpoint has been of great interest for designers of rule-based systems. Nevertheless, harmonic counterpoint has rarely been addressed, because it is more complex, as it implies a theory of harmony model.

KemalEbcioglu, in 1980, proposed a system for creating two-part florid counterpoint: to a given cantus firmus (“fixed song”) the system composes a matching voice, which is rhythmically independent. The author lists almost 50 implemented counterpoint constraints, which include complex high-level constraints such as ‘the pitches of different local maxima (i.e., melodic peaks) within three measures of the voice are unique’. Sources for compositional constraints were Joseph Marx and Charles Koechlin. Because their rules were insufficient for automatic composition, Ebcioglu added rules of his own. The search strategy embeds heuristics that prefer steps to skips and note pitches that have not occurred before.

A system for creating species counterpoint was introduced by William Schottstaedt in 1989, who aimed to follow the rule set of Fux as closely as possible. The system implements all five species for up to six voices. However, the author modified the original Fuxian rule set (more than 40 rules are quoted in the article) to get closer to Fux’ actual examples. In accordance with music theorists (including Fux) that state that rules are merely guidelines and not absolutes, the system assigns each constraint a numeric penalty value to denote its relative importance: the system searches for a solution with a small accumulated penalty. Compared with other counterpoint studies, Schottstaedt achieved relatively advanced examples (e.g., fifth-species for five voices). Still, the shown musical results reveal some limitations of the system’s rule set: in particular, the rhythmic structure is atypical for Palestrina style (almost march-like), and the melodies contain many large skips – in contrast to the Fuxian examples.

**Harmony**

As with counterpoint, there exists a huge amount of literature on the subject of harmony. Authors differ less in their focus on a certain style – the history of harmony shows a more continuous development when compared with counterpoint. However, there exist different approaches to explain harmonic phenomena. Most authors describe harmonic progressions as progressions of chord roots and analyze all chords in terms of their relationship with the tonic. One approach is based on the assumption that chord roots indicate one of the seven (major or minor) scale degrees, which are conventionally notated by Roman numerals. Another approach (often called functional harmony) only accepts three different main harmonic functions, namely the tonic, dominant and subdominant; notated usually with their initials. This approach explains all chords as variants of one of these main functions. Scale degrees highlight the diatonic interval between chords, whereas functional harmony denotes, which chords can substitute each other (e.g., *S* vs. *Sp*).

Like counterpoint, the development of harmony and its study is still ongoing and these developments are of particular interest for composers using constraint systems. For example, the harmonic language of ‘atonal music’, music in 12-tone equal temperament without a tonal center and often consisting of highly complex chords, is described in terms of pitch class sets. Microtonal music, and in particular music in just intonation is another important example for the ongoing development.

**Melody and Form**

Melody-writing is highly style-dependent, and this subject is traditionally less established in music theory than counterpoint and harmony. Nevertheless, the subject is covered, for example, by some general textbooks on composition. For instance, Schoenberg explains how in classical music a melody expresses the underlying harmony and how a melody is composed from motifs and their variations. Whereas Schoenberg teaches melody composition in a more systematic way, FrédéricMotte studies various aspects of melodies from different musical styles.

Research on modelling melody and form is still at an early stage. Motifs and their variation are very important for classical melody composition and form. It has been argued that the foundation for the successful modelling of harmony was the modelling of harmonic concepts such as pitches, intervals, chords and scales. As of now there are no examples of CSPs which have been able to show a complex solution to modelling melodic forms, current CSPs rely on harmonization of existing melodies.

**Rhythm**

Rhythm can be more easily defined compared to melody as the elements of rhythm such as pulse, meter, accent, stress, tie, syncopation, and suspension have well defined rules within music composition so it becomes much easier to express these concepts using constraint programming.

A system completely devoted to rhythmical CSPs is OMRC. For example, OMCR proposes a constraint-based quantification of the rhythms of every-day gestures (e.g., extracted from the sound of a passing train) and forces these gestures into readable music notation. Constraints may control what time signatures are allowed and how often the time signature may change. Additionally, the composer may apply further constraints. For example, the composer may demand that the quantified result will be built from pre-composed motifs.

**Instrumentation**

The art of instrumentation involves writing for different instruments taking particular playing techniques and limitations of instruments into account. The discipline orchestration additionally addresses how to sonically balance instruments. Instrumentation and orchestration are still developing today.

Mikael Laurson and Mika Kuuskankare present musical CSPs in which melodic, harmonic and voice leading constraints are complemented by instrumentation constraints, and they are presenting solutions showing idiomatic instrumental writing. The authors discuss guitar and brass instrument fingering in two case studies. For example, writing music for the guitar in a way that is well playable requires instrument-typical considerations. Guitar music is performed on six strings – tuned in a particular way – on which only four left-hand fingers are placed. Furthermore, the fingers can only be stretched up to a certain amount and moving the fingers requires a certain amount of time.

**Conclusions**

Constraint programming is a highly suitable paradigm for computationally modelling music theories and composition. The constraint programming paradigm has been used for several decades in this field, and many music theory subdisciplines have been addressed including counterpoint, harmony, rhythm and instrumentation.

Nevertheless, many complex aspects of music theory and composition still await constraint-based modelling. Examples include a rich theory of harmony focusing on convincing chord progressions (i.e. *not* harmonization of an existing melody, nor figured bass), harmonic counterpoint, or the modelling of melody and musical form. Also, there has been very few work on modelling instrumentation and orchestration so far. Moving forward in the field, these will be key theories to model if a more complete musical theory and composition program system will ever be realized.