Introduction

Machines that make music: the prophecy has been running throughout history since the mathematical nature of music was put forward by the Pythagoricians. Athanasius Kircher's *Arca Musurgica*, described in his 1650 treatise *Musurgia Universalis*, displays wooden inserts that can be drawn to arrange pre-defined musical fragments into a coherent contrapuntal composition. Not exactly an automaton, the *arca* could be seen as an early algorithmic music setup. It constrained, through a clever classification of musical bits into columns and rows, a huge combinatorial space to a subset of eligible arrangements that made sense with regards to the underlying music theory.

In 1821, Winkel built the "self-composing" Componium, yet another instance of orchestrion, a mechanical organ bringing into life the colors of the whole orchestra thanks to the mechanism of pinned cylinders and levers acting to open sound pipes or to strike drums. However, Winkel's invention had something more to it: a combination of Kircher's formal algorithmics and of a sound-producing device. Each barrel was formed by a juxtaposition of cylindric "slices", each storing two measures of music. A total of 40 pairs of measures, and, for each of these, seven additional variations were available. While a barrel was playing its two measures, a complex mechanism worked to shift the second cylinder to one of eight variations for the next pair of measures, so the music would seamlessly stream through one of 840 sequential combinations.

Chapter written by Charlotte TRUCHET and Gérard ASSAYAG.

These inventions, among many others of the like, drew an incredible amount of fascination. The dreams, however, spread faster than the technologies evolved. French writer Raymond Roussel (1877–1933), himself a musician, unleashes his imagination in two of his novels, *Locus Solus* and *Impressions d'Afrique*. One of his poetic machines is a pocket Componium, where the source of variations is the unpredictable wriggling of insect feets. These motions activate a complicated system of wheels and rods that end up plucking tuned metal strips. The purpose of this setup is to benefit from the endless changes induced by the random source while controlling the polyphonic output so that it still conforms to defined musical rules:

> *En outre, édifié avec le concours d'un harmoniste éclairé, un prodigieux système frénateur de rouages inextricables, régentant les huit zones séparément et dans leur ensemble, s'opposait à la production de toute cacophonie sans exclure aucune combinaison rationnelle et analysable.*
>
> (In addition, built with the help of an enlightened harmonist a wonderful reduction system of inextricable wheels, dictating the eight voices separately and altogether, prevented the production of any cacophony without excluding any rational and analyzable combination)
>
> Raymond Roussel, *Locus Solus*, 1914

In his *Impressions d'Afrique* (1910), Roussel describes a bigger machine, enclosed in a glass container, as a full-scale Componium extended with bowed strings, plucked harps, brass ensembles, and even a mechanical piano. However, the purely generative fabric is mixed here with the actual voices of opera singers played by an automated phonograph juke-box. Quite magically, the whole mechanism ensures synchronization and harmonic coherence between the audio material and the generative one, as if the machine was to "understand" that, as soon as the acoustical signal comes into play, the mechanical algorithmics should be constrained to an accompaniment behavior and should play by the rules.

These examples show that long before the advent of the digital era and musical informatics, visionary characters paved the way either by actually conceptualizing and engineering machines or by imagining them in the fictional and poetic realm. As is sometimes the case with science fiction, the imaginary inventions were not that far in essence from what happened eventually when science and technology were ready to fulfill the poetic vision. One of the finer technical achievements in modern music computing, called "score following", results from different research streams that took place for more than 25 years in such places as Carnegie Mellon University or Ircam research center in Paris. Just like Roussel's literary invention, the machine, now a digital one driven by complex software, tries to continuously synchronize with a live performer (or a recorded signal), sharing the same reference score, acting on the one hand to understand the live input (Is it slowing down or accelerating? Has the performer skipped or permuted a note in a rapid movement?) and on the other hand to perform its own part in harmony with the musical context. The machine part can be anything from just reading a (possibly time-scaled) sound file to executing generative algorithms that produce rich synthetical or sampled musical textures in coherence with the live part.

An interesting variant of this technology addresses the case when there is no common score to start with. This situation typically arises with improvisation. Computer music researchers have come up with software (such as the OMax software at Ircam) that incrementally build and update a model of the music played by the musician. Providing an interesting improvisation cannot be totally chaotic – signal processing, statistical learning, and formal sequence modeling engines work in cooperation to describe whatever structures of recurrence or variation they find in the input, and to set predictive hypothesis about what is coming next. Referring to this speculative model as a kind of open score, the digital agent can again unroll its musical contribution and add synthetic layers, by controlling virtual synthesizers or playing realistic instrumental samples.

If we follow the same little game, trying to find out to what extent the poet's prophecies have met with modern achievements, we would assimilate the "insect machine" to algorithmic music generators, hundreds of which have been experimented since the early experiments by Hiller and Isaacson in the mid-1950s. Their famous *Illiac suite* for string quartet (premiered at University of Illinois Urbana-Champaign on August 9, 1956) respects in part the pocket componium paradigm. The source note material is generated stochastically using Monte-Carlo methods. Then it is filtered out using formal rules inspired by 16th Century modal counterpoint as well as 20th Century serialism. Non conforming notes were excluded from combinations, in a sense preventing "the production of any cacophony without excluding any rational and analyzable combination". This seminal experiment has left a classical iterative scheme for music generation: produce material with greater diversity, combine it into vertical and horizontal arrangement, check against rules pertaining to the logics of some music theory, detect violations, exclude bad candidates, and look again for better ones. This applies to purely generative contexts such as algorithmic music and to interactional contexts as well, where part of the material is imposed by a live musician and part is generated in conformance with both the imposed reality and with general rules.

However, the generate and filter approach, in its basic definition, is clearly far from optimal considering the huge combinatorial spaces generated by music problems involving a great deal of variables. In a polyphonic music sequence, basically each note is a problem variable, connected to many others by constraints, that is, logical predicates restricting their possible values. Constraint programming has brought another way to look at it, by shifting the rules (now modeled as predicates connecting the variables) from the filtering stage to the exploration strategy itself. In effect, each choice for a variable, for example a note in a chord, restricts the remaining exploration domain for the others, for example the remaining notes in that chord. How does one dynamically define a combinatorial space and how does one navigate it in the search for solutions that have become important. There is no single magical solution, as will be shown in this book. This is a reason why many of the major software environments for assistance to music composition, analysis, or performance may contain not only one, but several constraint programming engines with different underlying models. This is, for example, the case for the OpenMusic environment developed at Ircam, or PWGL at the Sibelius Academy in Helsinki.

Techniques of (or inspired by) constraint programming are percolating into many regions of musical informatics such as algorithmic music, assistance to composition and analysis, interaction design, real-time control, sound synthesis and processing. In effect, recent research has shown that musical problems are intrinsically multi-scale: from sound physics to signal modeling to symbolic categorization to complex structures linking discrete units together. An ancient conception, where artificial intelligence techniques, including constraint programming, were reserved to symbolic descriptions (e.g. notes, chords, and discrete music parameters) while sound and interaction was in the realm of signal processing, is now challenged. Research in machine audition and music information retrieval produces more and more high-level signal-based descriptors that get closer to perception and cognition, thus to symbolic categorization. Composition software tends to mix formal symbolic representations with parameters for the continuous control of sound synthesis and processing. In the real-time interaction domain, specialists no longer believe that a purely signal processing approach will be powerful enough to model complex musical behaviors. Hybrid architectures, combining multi-scale representations such as time-frequency data, signal descriptors and symbolic alphabets, as well as various techniques such as signal processing, formal languages, and logical modeling, seem to appear as the next step in research and development. In this new landscape, constraint programming takes its place as one of the major high-level musical problem resolution techniques. It may appear in one of two ways: explicitly as a user-level language for specifying problems, or implicitly as a background processing library, transparent to the user, to which a part of the computations are delegated, in the frame of a complex composition or interaction system.

The seminal works on constraint and logic programming in music are Kemal Ebcioglu's. His goal was to compute music that would respect the complex rules of harmony and counterpoint. For the non-musician reader: classical music relies on numerous rules that aim at describing good music. Several treaties describe an impressive number of such rules, for instance: the extreme voices, bass and soprano, should not have similar moves (ascending or descending); different voices must not cross; there must not be successive parallel fifths or octaves between two voices; some dissonant intervals between two successive notes of a voice are forbidden; a phrase must end with a cadence; and so on. From a computer science point of view, these rules are the constraints of classical music. Thus, in this formalism, writing good music is a matter of finding notes, within a given *range* (each voice has an interval of possible notes), respecting the rules. This naturally has a flavor of constraint programming.

In an article entitled "Computer counterpoint" in the *Proceedings of ICMC'80* (International Computer Music Conference), Ebcioglu details the rules of florid counterpoint, and proposes an enumeration technique to produce correct music. Heuristics could also be defined to guide the search toward preferred solutions. He then developed CHORAL, an expert system based on logic programming, to produce four-part harmonies in the style of J. S. Bach. Although this is not *stricto sensu* a constraint program, this work stated the bases of the automatic harmonization problem that was developed later by other authors (Tsang and Aitken in 1991, Ballesta in 1994, Pachet and Roy in 1998, to name but a few). Ebcioglu not only formalized the musical rules in a logical framework but also noticed that these rules "were grossly insufficient for producing a nice output". He thus added his own rules to improve the musical result by constraining part of the melodic profile. Later on, in 1998, Dellacherie and Chemillier showed that the harmonization problem had many solutions, which is not surprising.

What are the rules to produce good music? Does constraint programming have a role to play in contemporary music, and not only in classical music? On which criteria can the output of a musical system be validated? Are these questions even well defined? We do not have the answers. Actually, we even chose not to ask the questions. Instead, we invited composers and musicians to be part of this book, along with computer scientists. This book is thus a meeting between producers of constraints, that is, computer scientists who develop constraint system hopefully well suited to music, and users of constraints, that is, musicians who explain why and how they use this particular technique. Sometimes, it may not be the usual way.

The chapters are independent and they can be read in any order. They are sorted by lexicographical order according to the name of the first author. Several paths can be followed through the book, and three of them are proposed below, addressed to non-musician computer scientists, constraint programmers, and musicians.

For the non-musician reader, a good starting point is the chapter by Georges Bloch and Charlotte Truchet. Georges Bloch, a French composer and Professor at the *Conservatoire de Paris* (France), explains how he sees the role of constraints in the compositional process. The authors use a local search algorithm to solve a harmonization problem, and the approached solutions, depending on their costs, are integrated into the score by unfolding the timeline.

Musical time is also an important concern in the chapter by Antoine Allombert, now a researcher at the University of Paris 13 (France), with his co-authors. This chapter describes the problem of interactive score, where performers can improvise some parts. Constraints maintain the temporal coherence of the score. The basic constraint system can be solved by Petri nets, but the use of a timed calculus model enables a finer interaction. This can be followed by the chapter by Carlos Olarte, computer science researcher at the University Javierana Cali (Colombia), with his co-authors, which presents in more detail a temporal concurrent constraint calculus. In music, this process calculus models several problems such as dynamic interactivity in a score.

Time in music is of course related to the rhythmical structures. Rhythms are difficult to formalize, probably because, from a computer science point of view, they are contextual and have a complex hierarchical structure. Örjan Sandred, a Swedish composer at the University of Manitoba (Canada), has designed a sophisticated system, in particular for rhythm representation. This underlying structure can be accessed by the constraint system, with a clever masking system for choosing the constrained parameters.

The question of an appropriate music representation for musical constraints is detailed in Torsten Anders' chapter. Torsten Anders, a German composer now at the University of Bedfordshire (United Kingdom), designed Strasheela, a constraint-based music system for composition.

Serge Lemouton, a computer scientist and musician from Ircam in Paris (France), details in his chapter several constraint problems, again on different musical objects: melodies, harmonic profiles, and so on. These problems are solved in Gecode. His point of view is precious as he has worked in collaboration with several composers and details the musical aspects of these works.

Modeling musical constraint problems is also at the basis of the chapter by Charlotte Truchet, computer science researcher at the University of Nantes (France), which describes several musical problems in contemporary music. The problems are solved with a local search method, enabling us to deal with meaningful approached solutions for overconstrained problems. She then developed OMClouds, a library dedicated to musical constraints.

Finally, the chapter by Grégoire Carpentier, researcher at Ircam and *Haute Ecole de Musique* in Geneva (Switzerland), opens an entire new field for musical constraints, with the appearance of sound features. It formalizes the problem of automatic orchestration, which consists of combining sounds from different sources or instruments to meet a given sound profile. To tackle this problem, a constraint system must deal with musical compound objects and integrate multi-criteria optimization. Grégoire Carpentier finally describes the use of his system by the British composer Jonathan Harvey.

There are other ways to read this book. A constraint programmer might prefer to be guided by the solving techniques, starting by the well-known Gecode solver used by Serge Lemouton, or the Oz (Mozart) programming language underlying the Strasheela system by Torsten Anders. Still on complete methods, Örjan Sandred uses PWMC, an *ad hoc* solver for musical constraints on rhythms. Incomplete methods, here local search algorithms, have also been introduced in musical constraints, and the use of a particular metaheuristic is exposed in the chapters by Charlotte Truchet and Georges Bloch. A similar algorithm is adapted by Grégoire Carpentier to solve the orchestration problem. Finally, the chapters by Antoine Allombert *et al.* and Carlos Olarte *et al.* introduce interaction and dynamicity, which are research domains within constraint programming.

Musicians should probably start with the two chapters providing a catalog of detailed musical constraint problems, for example, Serge Lemouton and Charlotte Truchet, to get the flavor of constraint programming through musical examples. The chapter by Torsten Anders details the example of florid counterpoint as a constraint problem, which should sound familiar. This chapter also discusses music representation issues, and can naturally be followed by the chapter by Örjan Sandred that also details a music representation, focusing on rhythms. The chapter by Georges Bloch and Charlotte Truchet then shows how musical questions and constraint programming issues interact in the compositional process. The question of time raised by Georges Bloch is also at the core of the chapters by Antoine Allombert *et al.* and Carlos Olarte *et al.*, which shift the music representation from the usual, pre-defined score to the problem of interactive score where music is partly written, and partly improvised. Grégoire Carpentier's chapter finally extends the constrained musical objects from the symbolic score to sound attributes with the problem of orchestration.

Constraint programming has now earned a full place in the toolbox offered by computer-assisted composition, and more generally computer music. The variety of solving techniques, languages, and music representations used in the field shows that musical applications of constraint programming should be designed in a true computer-aided decision process, interleaving the constraint technology and the musical knowledge expressed in the music representations.

Finally, it is an awful thing to write about music. Music is meant to be heard, not read. Accompanying this book, the reader will find online[1] some musical excerpts associated with several chapters. After reading this book, please, forget about the constraints, and listen to the music.

1 http://contraintesmusique.lina.univ-nantes.fr/ContraintesMusique/.

Acknowledgment

The editors would like to thank Carlos Agon for his help on reading and formatting this book.