Luc Döbereiner
Institute of Electronic Music and Acoustics
University of Music and Performing Arts Graz
Inffeldgasse 10/3, 8010 Graz, Austria
luc.doebereiner@gmail.com

Between the Abstract and the Concrete: A Constraint-Based Approach to Navigating Instrumental Space

Abstract: This article deals with a way that algorithmic composition systems can be informed by material realities of musical performance. After a general discussion of the relation of abstract algorithms to concrete materiality, the article focuses on the idea of an instrument's space of possibilities. It briefly discusses a number of compositional approaches that seek to derive musical structure from bodily movements and from the physical properties of instruments. The last part describes a new open-source JavaScript library called OboeJS and a Web application based on this library. The system is an experimental exploration of the idea of instrumental space and an attempt to bring together abstract algorithmic processing and the concrete possibilities of a musical instrument. The system implements a flexible constraint-based search algorithm for the generation of oboe fingering sequences. This tool is presented as part of a wider approach to algorithmic composition that aims not to map data output of generative procedures to "sound generators" (e.g., performers, instruments, sound synthesis processes). Instead, I propose to derive structure from the space of possibilities of the instrument itself, which in this case is the oboe.

From its inception in the late 1950s until today, publications in the field of algorithmic composition have been characterized by a striking absence of the role of acoustic instruments and the musical performer. Pioneering projects such as Hiller and Isaacson's *Illiac Suite*, Gottfried Michael Koenig's *Project 1* and *Project 2*, and Iannis Xenakis's early endeavors in computer-generated music are almost exclusively concerned with the properties of abstract structures, which are made audible in a secondary stage. The algorithms themselves are removed from the material properties and conditions involved in their sonic realization. On the reasons for their choice of instrumentation, for example, Hiller and Isaacson (1959, p. 153) write:

> The first decision was the choice of a musical medium through which the results could ultimately be heard. . . . [T]ranscription for a keyboard instrument such as the piano was . . . eliminated because this would introduce the special restriction of having to have the music fit under the hands at the keyboard. Therefore, since, even in Experiment One, the objective was to produce a four-voiced polyphonic texture, the choice of a string quartet medium appeared logical and convenient; the problems in transcription would be minimized.

Computer Music Journal, 43:1, pp. 8–20, Spring 2019
doi:10.1162/COMJ_a_00497

The specific properties and restrictions of the sounding instruments did not inform the design of the algorithms in any way other than the pitch ranges of the four instruments. Even if instrumental techniques are a parameter in some sections of the *Illiac Suite*, the physical and performative properties of the instruments do not influence the space of musical possibilities and structures. This is also true for Koenig's composition program *Project 1*, in which "INSTRUMENT" is an open parameter that can be organized without knowledge of the underlying physical realities to which it refers. Although Koenig (1983, pp. 27–28) states that "players should not be treated like generators, nor generators like players," in his conception of compositional processes both take on roles of purely executing performance instructions. For Koenig, algorithms mediate between an initial conception and a musical realization; they form abstract objectifications of structural compositional decision-making processes, whose results require interpretation by the composer and by performing musicians to be turned into aesthetic objects. As Koenig writes, "the result of algorithmic composition can be converted into performance categories that are foreign to it: voice leading, phrasing, and playing technique" (Koenig 1983, p. 28). This shows that for Koenig the categories of material performance and those of computation and algorithmic abstraction are located in entirely separate spheres.

In many ways this separation of symbolic operation and material performance is inherent to traditional understandings of an algorithm. Algorithms are commonly defined as abstract methods, recipes, or rules that are composed of a finite series of instructions taking input data and producing output data. These instructions, which "must be rigorously and unambiguously specified for each case" (Knuth 1997, p. 5) can be carried out automatically, either manually or by a programmable computer. Algorithms constitute formalized generalizations that are not tied to any particular situation, material, space, or object but that can be applied to a variety of different domains and to diverse forms of data. Their effectiveness is based on their transferability, which in turn rests on the representational discretization and symbolization of opaque and contingent material realities.

The mathematician Giuseppe Longo (2009, p. 46) has criticized computational reason as a biased and dualistic view that asserts that "intelligibility is produced by the decomposition of the universe into atoms and the discourse on the universe into simple and elementary links." This way of thinking is, for example, reflected in the assumption that genetic information is "alphabetically" encoded in the genotype, which, much like a computer program, deterministically produces the phenotype—i.e., the life form in its physical expression. He argues further that "many other factors contribute to ontogenesis; particularly, a multitude of irreproducible and irreversible dynamics, which are at the center of the variability of living phenomena" (Longo 2009, p. 60). He continues that, although highly effective, computational reason is biased and leads us to disregard the complexity of the "elementary." Discrete symbolic representation reduces the manifold contingent dynamics of material processes, yet it is able to model real spaces of possibility.

The transferability of such abstract models has fueled much research in the field of algorithmic composition. Many encyclopedic surveys of the field (e.g., Nierhaus 2009; Miranda 2001) consist, to a large extent, of brief accounts of computational procedures of prediction, control, and analysis from various technological and scientific disciplines, such as biology, physics, chaos theory, linguistics, probability theory, artificial intelligence, and music cognition. A common assumption at the basis of the musical application of such procedures is that the generated data can be rendered audible with an instrument or sound synthesis process that is independent of the abstract model itself. In doing so, the mapping of data to established musical parameters regards the abstract realm of algorithmic processes, on the one hand, and the material realm of performance and sound production, on the other, as fundamentally different. One produces shaping forms, while the other passively mediates form.

This shows how algorithmic composition, and generative art more generally, are characterized by an opposition of abstract structures versus realized perceivable events, sounds, forms, or objects. The composer Karlheinz Essl, for example, describes algorithmic composition as based on the idea of "an abstract model behind the sensual surface" (Essl 2007, p. 107). The separation of a musical work into its sensual appearance, on the one hand, and its abstract organization, on the other, is also reflected in certain conceptions of musical listening. Roger Scruton's conservative notion of musical experience, for example, builds on Pierre Schaeffer's idea of acousmatic listening and argues that the identification of the abstract order "behind" the sonic phenomenon is what distinguishes musical listening from other forms of nonmusical listening. According to Scruton (1997, p. 221), a person listening to sounds as music "is hearing the sounds apart from the material world." But elevating the abstract and ostensibly immaterial order of musical tones (i.e., the organization of discrete elementary entities that are congruent with the parameters of traditional musical notation) not only limits the understanding of compositional practice, it discounts the material entanglement of musical works with perception, performance, acoustics, and discursive signification. This raises the question of how computational processes can be related to material and phenomenal properties.

One way in which artists have sought to go beyond the purity of symbolic computational processes is to embrace forms of indeterminacy that result from folding these processes back on themselves. By means of "feedback through which the genotype is modified by the phenotype" (McCormack et al. 2009, p. 357), the stark opposition of determining form and determined material is dismissed in favor of an idea of emergence. Cornelius Cardew's "Paragraph 7" from *The Great Learning*, (1968) and Agostino Di Scipio's *Audible Ecosystemics* (2002–2005) may serve as two different examples of how formal processes can give rise to unpredictable forms while interacting with concrete material situations.

In Di Scipio's case, feedback systems consisting of interacting components, including digital signal-processing networks, acoustic spaces, and musical instruments, react to their own sonic output and thus evade any clear distinction of an abstract model behind a sensory surface.

In Cardew's work, each member of a chorus initially chooses a pitch freely, and as the piece progresses the singers select pitches sung by their colleagues. If there is no pitch available that is different from the pitch they have just been singing, the singers once more choose a new pitch freely. Based on multiple actors performing the same basic rule, the piece thus creates emergent harmonic developments that oscillate in complexity, converge, and bifurcate. Cardew's composition is an example of a simple algorithmic procedure that is inseparable from its performance, perception, and musical interaction. In both examples algorithms are used to engage with concrete realities and generate unpredictable musical situations.

Although compositional algorithms are often developed to automate certain procedures, model particular existing musical styles, or to deduce structures from top–down formalisms, they can also be seen as extensions of human thought that allow composers to experiment with material realities in ways that exceed human imagination. As Kostas Terzidis writes, "algorithmic processes become a vehicle for exploration that extends beyond the limits of perception" (Terzidis 2003, p. 72). The examples by Di Scipio and Cardew demonstrate that algorithmic processes are not merely forms of mechanized reason that are removed from physical and sensual properties, but that, as Rutz and Pirrò (2018, p. 177) write, "computational processes . . . can also be understood to occupy abstract spaces which are interwoven with physiological spaces." Instead of viewing algorithms either as instruments of simulation or as purely formal and deterministic means of mechanical deduction, we can see them as ways of unlocking potentials and expressing virtual spaces of possibility that are entangled with material forces, objects, and situations.

Material bodies, human performers, acoustic spaces, and musical instruments, but also other algorithms and data, can condition and interact with compositional algorithms that create computational access to otherwise inaccessible, unchartered, or unimaginable abstract spaces. The distinction between abstract formal algorithms and the mapping or application thereof to a concrete instrument fails to recognize the immanent virtuality of the concrete itself. Musical instruments, in conjunction with their human performers, are bodies that have certain capacities that are structured according to abstract but real spaces of possibility, and algorithmic models can render regions of these spaces accessible. As Manuel DeLanda writes, a "model can capture the behavior of a material process because the space of possible solutions overlaps the possibility space associated with the material process" (DeLanda 2011, p. 19). Instead of understanding musical forms as the incarnation of abstract relations—that is, instead of viewing algorithmic composition as the mapping of models to concrete sounding entities—I seek to explore the formalization of the virtual structure immanent to concrete instruments. Although the concrete or material has traditionally been understood as shaped by abstract essences or ideas, "the true philosophical question is, how can concrete fact exhibit entities abstract from itself and yet participated in by its own nature?" as writes the philosopher Alfred North Whitehead (1978, p. 20).

Instrumental Space

The fundamental question motivating this article is how to bring together abstract algorithmic processing and the concrete contingencies of a musical instrument. The software discussed in the next section is an attempt to reconcile formalized generative approaches and the specificities of a given material body, which is hence not only a means of realizing an otherwise abstract idea. It rests on the central notion of an instrumental *possibility space*, that is, an abstract space that is immanent to a concrete body, which is the acoustic instrument in conjunction with the player's body. The central question is thus, how can the abstractness of an actual body be explored and traversed, how can we create a cartography of an instrument? In this section, I sketch a brief aesthetic lineage that has motivated me to develop the software, OboeJS, described subsequently, although it has a broader practical use and may be used for diverse artistic, technological, or pedagogic purposes.

Instrumental properties, playing techniques, and physical restrictions have certainly informed compositional practices in virtually any style throughout the history of instrumental music. The subject of a fugue of a sonata for violin by Johann Sebastian Bach, for example, is bound by structural and physical constraints. It is designed in view of the polyphonic and transformational possibilities that the violin's space of possibilities affords. The instrumental topography is thus inscribed into the musical material. In the late 20th and early 21st centuries, musicians and composers have not only extended instrumental techniques, but some have also developed approaches that explore the structural repercussions and potentials of instrumental restrictions and possibilities. These potentials and constraints have rarely been a central concern in algorithmic composition systems, however.

Although modernist serial and aleatoric music of the mid-20th century largely prioritized symbolic structures and generative mechanisms over instrumental actions, and although spectral music has concentrated on modeling timbral transformations by means of an analysis–resynthesis paradigm, a number of composers beginning in the 1970s have focused on instrumental actions and their structural and performative signification. Helmut Lachenmann's seminal works *Pression* for cello (1969–1970), *Dal niente* for clarinet (1970), and *Guero* for piano (1970) explore the relation of instrumental action and sound result. As Lachenmann writes, it is a music "in which the sound events are chosen and organized such that the mode of their creation is at least as important as the resulting acoustic properties themselves," and continues that, instead of composing particular timbres or pitch relations, *Pression* is composed by means of "relations of pressure in sound actions on the cello" (Lachenmann 1996, p. 381, my translation). The principle "parameter space" is thus determined by the cello as a physical object. In a stark contrast to the aforementioned dematerialized understanding of musical listening formulated by Scruton, Lachenmann's so-called *musique concrète instrumentale* relates sounds back to their physical origin.

Instrumental space appears in a different but equally determining way in the work of Brian Ferneyhough. In his composition *Time and Motion Study II* for cello and live electronics (1973–1976), which the composer describes as the "memory of a production process" (Ferneyhough 1995, p. 114), the cellist's actions on the fingerboard and on the body, as well as right- and left-hand activities, are defined as separate but interacting layers.

Iannis Xenakis's *Synaphaï* for piano and orchestra (1969) may serve as another example for the entanglement of the performer's body, the instrument, and structural concerns. The work is notated in separate staves for each of the pianist's fingers, thereby employing the player's finger dispositions and movements as a structure-generating factor.

More recently, a number of composers have developed new ways of exploring physical actions and instrumental sound production as primary compositional parameters. By drawing on the work of the late composer Klaus K. Hübler, Aaron Cassidy, among others, has developed ways of "decoupling" instrumental techniques. This decoupling consists of the dissociation of individual physical parameters of instrumental performance, such as fingering, embouchure, breath, finger pressure, and finger position. In doing so, he allows for new interactions between these parameters in contrapuntal ways, which can give rise to partially unpredictable results. Cassidy's *Second String Quartet* (2010) entirely relinquishes the notation of pitch. As Cassidy (2002) writes, "physicality drives the musical surface—the harmonic organization of the piece derives not from abstract connections between pitches but instead from a set of carefully mapped hand and finger positions."

The work of the composer Richard Barrett is characterized by a related yet distinct conception of the instrument as a space of possibilities. In Barrett's work, compositional gestures are not primarily the unpredictable result of decoupled physical parameters but are inseparable from the instrumental topography and the performer's movements. Barrett describes his approach in spatial terms: "The instrument becomes not a machine for projecting sequences of notes or sounds which contribute to an abstract compositional structure, but instead a theater of action with its own characteristics, its own landscape, through which the composer is then able to make 'poetic journeys'" (Deforce and Barrett 2001).

The works of Cassidy, Barrett, and others demonstrate how the materiality of instrumental performance can be embraced as a generative and creative musical resource and how spatial notions of instrumental possibilities can be employed to construct meaningful multidimensional compositional frameworks. The question that is of primary relevance for me and for the project described in the following section is how formalized algorithmic procedures can relate to this materiality without merely simulating or modeling its dynamics. I am interested in ways in which the formal and the material can be entangled yet maintain a certain autonomy—that is, how algorithms can formally extend compositional thought while being informed by and open to material contingencies.

OboeJS: A Constraint-Based Approach

OboeJS (https://github.com/lucdoebereiner/OboeJS) is an open-source JavaScript library based on the idea of instrumental spaces of possibilities. The oboe was chosen because of its enormous space of fingering combinations, its rich multiphonic possibilities, and the available data on fingerings suitable for constraint-based search algorithms. The choice is somewhat arbitrary, however, because the project is conceived as a first step towards developing approaches to instrument-specific algorithmic composition. In doing so, the oboe is understood as an explorable multidimensional phase space. I have not aimed at simulating the physical sound production of the instrument (see, for example, Almeida et al. 2004 for approaches to the physical modeling of double reed instruments) but instead I have sought to model this possibility space from the performer's point of view.

Naturally, the space is merely a section of the vast phase space of the oboe. Moreover, OboeJS deals with discrete states that are primarily based on *fingerings*. As defined for the purposes of this software, each fingering consists not only of the positions of the fingers and keys, but also air pressure, lip pressure, embouchure, dynamic possibilities, and pitch content. Each of these physical parameters, however, has its own particular thresholds, dependencies, and continuities, such that a fingering as a discrete state is already a reduction of a more elementary complexity of interacting forces. The books *The Techniques of Oboe Playing* (Veale and Mahnkopf 2005) and *Oboe Unbound* (Van Cleve 2014) were the main sources for the fingering data. Trained oboists know large areas of this space tacitly; however, the possibility of navigating parts of this space according to multiple interdependent constraints not only renders this tacitly known space accessible to composers but also can disclose crossconnections and possibilities that go beyond any player's knowledge. The database currently contains 1,465 fingerings, hence, there are already more than three billion possible distinct fingering sequences with the length of only three fingerings. This vast possibility space exceeds human comprehension and navigating it requires algorithmic processing.

Besides the representation of the possibility space, developing ways of creating trajectories through this space is the most crucial part of the system.

Since I understand OboeJS to be an extension of compositional thinking into a realm that is otherwise inaccessible, the means of expressing conditions for structures to be derived from the possibility space is key. The nature of the possibility space lends itself to regard its navigation as a search problem and to use constraint satisfaction methods. In her seminal volume *Constraint Processing*, Rina Dechter (2003, p. xvii) defines a constraint as "a restriction on a space of possibilities," which in the case of OboeJS is the space defined by the fingerings, that is, the *domain*. The programming paradigm of constraint processing allows for a declarative formulation of problems. It allows for the description of conditions that need to be satisfied for a possible solution without specifying how to satisfy these conditions. This frees composers and performers from having to design specific procedures and allows them instead to formulate relationships, requirements, and restrictions, as long as these can be formulated as relations among variables.

Constraint processing is particularly well suited for solving problems involving complex networks of relations and restrictions among sets of elements. It has been utiilized in a number of algorithmic music systems since the 1980s, and there are current implementations specialized for musical applications (e.g., Anders and Miranda 2010; Sandred 2010; Fernández and Vico 2013). Most constraint-based composition systems, however, have focused on traditional musical parameters, on solving voice-leading problems in traditional harmonic and melodic frameworks, and on the modeling of rhythmic, melodic, and harmonic aspects of existing and historical musical styles. There are hardly any applications of constraint processing for modeling the possibilities of instrumental playing (cf. Laurson and Kuuskankare 2000 for one notable exception). Constraint processing facilitates dealing with potentially immense sets of multidimensional data as well as with the complex interdependence of parameters. This makes it a suitable strategy for navigating models of musical instruments. Moreover, constraint processing is characterized by its generality, and the approach described in this article is intended to stipulate novel compositional methods that connect aspects of the physical performance and structural concerns. OboeJS can thus be seen as a tool for discovering structure in a physically informed big data set.

OboeJS has a modular structure and consists of the following components:

1. A general backtracking constraint solver.
2. A database containing 1,465 unique oboe fingerings containing multiphonic and monophonic fingerings. This database forms the domain of the search algorithm.
3. A set of predefined constraints, or predicates, including constraints for the specific evaluation of the difficulty of fingering transitions.
4. Functions for the export of generated fingering sequences in several formats, including files for the notation program LilyPond (http://lilypond.org).
5. A Web application, which forms an easy-to-use "front end" for searching the database according to a combination of predefined constraints.

OboeJS is built around the central `Solver` class, which specifies constraint problems. A search, and thus a `Solver` object, needs to contain three elements: the length of the sequence that is to be generated (the *variables*), the set of possible values (the *domain*), and the constraint function that needs to be satisfied. The `Solver` class is not limited to OboeJS, it can be used to solve arbitrary constraint problems using any type of domain. The `Solver` class performs a look-ahead backtracking search on the domain and instantiates the variables to values that will be tested against the constraints. The constraint function needs to be a function that returns a Boolean value. In each test during the backtracking search, this function is applied to the instantiated variables. This simple design allows constraints to be easily combined using logic operators, such as AND (`&&`), OR (`||`), and NOT (`!`). The search process will be suspended once a solution has been found, and the search can be resumed. Successive calls will find all possible solutions.

Figure 1 shows an example of using the `Solver` class. This example, which generates the first

Figure 1. Example of using the `Solver` *class to solve a problem using constraints. In this example, the Solver generates the first eight prime numbers.*

Figure 2. The fingering for a B-flat as represented by the `OboeFingering` *class.*

```
var s = new Solver(
    8, // length of the output sequence
    [ ...Array(20).keys() ], // the domain: [0, 1, 2...19]
    // constraint function
    array =>
        // all elements must be different
    array.predAllDifferent() &&
        // 2 must be the minimum element
    Math.min(...array) == 2 &&
        // there can be no smaller element (el2) that divides
        // any element (el1) without a remainder
    array.predAll(el1 =>
              !array.some(el2 =>
                        (el1 > el2 && el1 % el2 == 0))));
// Generate a solution
var solution = s.solve();
```

Figure 1

eight prime numbers, demonstrates the general functioning of the constraint solver that is at the core of OboeJS. The constructor takes three arguments: the length of the output sequence, the domain, and the constraint function. The generated sequence will contain eight elements and the domain is an array of integer values from 0 to 19 and is expressed using the ECMAScript 6 spread operator (`...`). The constraint function combines three tests using the Boolean AND operator. The first test uses the `predAllDifferent()` function and ensures that all values will be distinct. The second test ensures that the smallest element in the output sequence is equal to two. The last test is a simple primality test. It goes through each of the instantiated variables in the output sequence and ensures that there is no other variable that is smaller than the value and that divides the value without any remainder. The returned `Solver` object can be used to generate solutions to the problem by using the `solve()` function.

Fingerings are objects in OboeJS. Figure 2 shows how a monophonic fingering object can be created. As defined for the purposes of the software, each "fingering" consists of the pitch as a MIDI note number, an octave keys list, the states of the main keys played by the left and right hands, a Boolean value indicating whether the function is a trill fingering, the air pressure, the lip pressure,

```
var bb = new OboeFingering(
    58, // pitch as MIDI note number
    [ ], // octave keys
    [ "x", "x", "x", "bb" ], // left-hand keys
    [ "x", "x", "x", "c" ], // right-hand keys
    false, // trill fingering?
    3, // air pressure
    1.5, // lip pressure
    0, // index
    "ppp", // minimum dynamic value
    "fff", // maximum dynamic value
    false); // multiphonic fingering?
```

Figure 2

minimum and maximum dynamics, and optional indices of multiphonics with the same fingering.

Figure 3 shows how to use the oboe-fingering database together with the `Solver` class to create a fingering sequence. The predefined constraint `predIntervalRange` is used to limit possible intervals in either direction to a given range. In this case, the permitted intervals between successive notes range from a quarter tone to two semitones. The last constraint, `predNumberOfChanges`, is used to limit the finger changes between consecutive fingerings to be either one or two, that is, either one or two fingers will open or close a key from one fingering to the next.

Figure 3. A database search using the `Solver` *class.*

Figure 4. The first solution of the search in Figure 3. The staff at the bottom shows the sounding pitch.

Figure 5. Definition of a constraint. In this example the constraint limits pitches to a range.

```
var s = new Solver(
    // length of the output sequence
    10,
    // the domain (the monophonic fingerings database)
    oboeMonoDB,
    // constraints
    fingerings => {
        // all different
      return fingerings.predAllDifferent() &&
            // possible interval range
        fingerings.predIntervalRange(0.5,2.0) &&
            // number of changing fingers
        fingerings.predNumberOfChanges(1,2);
    });
```

Figure 3

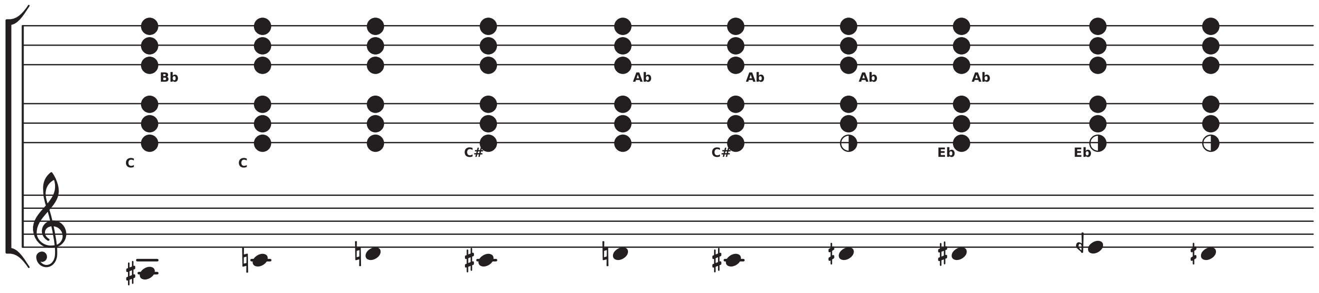


Figure 4

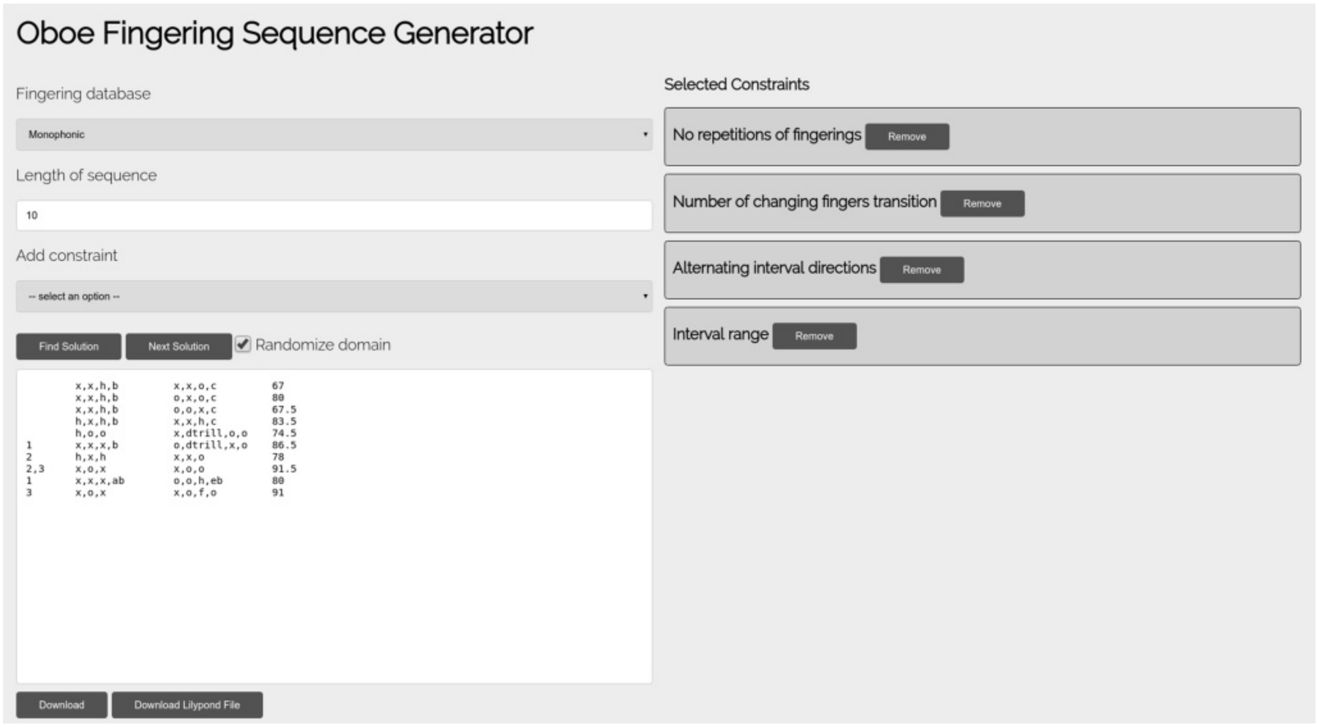
```
Array.prototype.predPitchRange =
    function(pmin, pmax) {
      return this.map(f => f.pitch) // get pitches
      // ensure that all pitches are in the given range
          .predAll(p => (p >= pmin) && (p <= pmax));
    };
```

Figure 5

Figure 4 shows the first solution rendered using the LilyPond export function. The upper two staves above the normal staff lines show the left-hand keys on top and the right-hand keys below, with the lowest staff indicating sounding pitch. Closed keys are denoted by a black circle, half-closed keys by a half-filled circle, and open keys by white circles (none in this example, but some will be shown later, in Figure 7). The letters represent additional keys, and numbers on top of the staff show the octave keys (none in this example, but some will be shown later, in Figure 9).

New constraints can be created by combining predefined constraints, by writing anonymous functions, or by defining functions that take an array of `OboeFingering` objects and return a Boolean value. Figure 5 shows the definition of the pitch-range constraint.

Figure 6. The Web application. On the left-hand side the type of database can be selected (monophonic, multiphonic, or both) and the length of the sequence to be generated can be set. Constraints can be added using a pull-down menu, with the elected constraints listed on the right-hand side. The generated output is printed on the lower left-hand side.



Oboe Fingering Sequence Generator

Fingering database: Monophonic

Length of sequence: 10

Add constraint: -- select an option --

Randomize domain [✓]

Selected Constraints:
- No repetitions of fingerings
- Number of changing fingers transition
- Alternating interval directions
- Interval range

```
        x,x,h,b       x,x,o,c          67
        x,x,h,b       o,x,o,c          80
        x,x,h,b       o,o,x,c          67.5
        h,x,h,b       x,x,h,c          83.5
        h,o,o         x,dtrill,o,o     74.5
1       x,x,x,b       o,dtrill,x,o     86.5
2       h,x,h         x,x,o            78
2,3     x,o,x         x,o,o            91.5
1       x,x,x,ab      o,o,h,eb         80
3       x,o,x         x,o,f,o          91
```

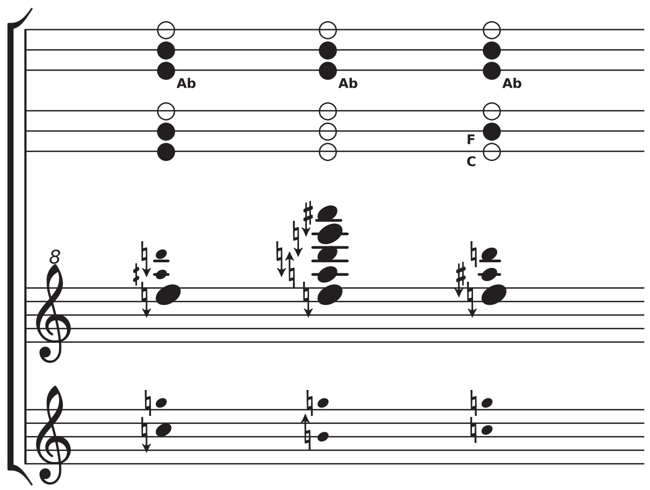
Besides the use of OboeJS as a JavaScript library, there is a Web application (http://doebereiner.org/OboeJS/oboeseq.html) that provides easier access to the database. A number of predefined constraints can be selected and combined. Figure 6 shows the Web application. It is included in the source code repository (see the files main.js and oboeseq.html).

OboeJS and its components can easily be integrated into other applications. Owing to its modular design and the generality of the search algorithm, the system can also be adapted to databases representing other instrumental spaces. Undoubtedly, it lends itself particularly well to instruments with a rather discrete possibility space. Woodwind instruments are characterized by continuous physical parameters such as air and lip pressure, but they can also be described in discrete terms, since each fingering can be regarded as a distinct state. Nevertheless, there are complex and highly interesting and unstable states in between fingerings, such as when slowly opening or closing one or several keys. These transitions cannot be dealt with in the present system. Such transitions would require other forms of modeling, as would instruments with a generally more continuous possibility space, including bowed string instruments.

Examples and Applications

OboeJS is intended to constitute the basis for individual compositional strategies of exploring the instrumental space of the oboe. Moreover, it may be used as a pedagogical tool or as a practice aid for performers. It formalizes much of the knowledge given in books on instrumentation and playing techniques and makes it computationally accessible. The results of the search algorithm may be interpreted as temporally consecutive sequences or as unordered groups of fingerings that have the distinction of being related in specific ways.

Figure 7. The only possible sequence of "neighboring" fingerings with three beating mulitphonics.



The systems also serve to find sequences and structures with exceptionally rare features. There is, for example, only one possible sequence of three multiphonics with strong beatings that are *neighbors*, that is, that allow for particularly easy and legato transitions among all fingerings (see Figure 7).

OboeJS contains a number of predefined utility functions and constraints that facilitate dealing with the pitch content of multiphonics. Such constraints can use the strongest pitches, set operations on the pitch content from all multiphonics in a generated sequence, and specific properties, such as beating multiphonics.

Figure 8 shows an example of using a constraint to create a sequence of nine distinct multiphonics. The strength of each partial pitch is indicated by the size of the respective notehead. In this example, consecutive multiphonics always share at least one of their strongest partials. These shared pitches will be heard continuously while the other partials change, thus creating a form of timbral polyphony.

The example shown in Figure 9 is closely related to the previous example. It is a sequence of ten multiphonics generated using the `Solver` object shown in Figure 10. The sequence is a transition of similar but slightly different complex timbres. The pitch content of consecutive multiphonics differs at most in two pitches (there is a tolerance of an eighth tone), creating smooth spectral transitions.

There are a number of predefined constraints for defining the difficulty of fingering transitions and the number of successive finger changes. There are functions that assign fingers to keys, since a certain key may not always be played by the same finger. If fingering transitions do not contain changes from half-depressed to quarter-depressed keys, if the fingers of one hand do not change in different directions, if the little fingers do not need to change or depress several keys at once, if air and lip pressure remain constant, and if a number of other conditions concerning the octave keys are met, the involved fingerings are considered *neighbors*. These fingerings can, for example, be used in trill or tremolo groups. Figure 11 shows a neighbor group, which is a set of fingerings that are all neighbors of each other.

The fingering transitions in Figure 12 contain a sequence of ten fingerings. The transitions become increasingly more difficult starting with one key change (lifting or depressing a key) and ending with eight key changes between consecutive fingerings. Additionally, there are constraints that enforce alternating interval directions and that prohibit intervals smaller than a minor sixth. This sequence was produced with the Web app, and Figure 6 displayed a screenshot of the session in which this example was produced. Although there is currently no representation for rhythmic or other temporal values in OboeJS, measuring the difficulty of fingering transitions can be used to derive relative durations for the performance of these transitions (e.g., allowing the performer more time to make the harder transitions).

Conclusion and Future Work

OboeJS is an experimental attempt to bring together abstract algorithmic procedures and concrete, material performance realities. This is not done merely to simulate musical performance, but rather to extend abstract thought into concrete realities and vice versa. Owing to the nature of the system and the search algorithm, it can almost exclusively capture discrete aspects of instrumental performance. Other

Figure 8. Consecutive multiphonics sharing at least one of their strongest partials.

Figure 9. Consecutive multiphonics differing at most in two pitches.

Figure 10. Constraint on pitch content of consecutive multiphonics.

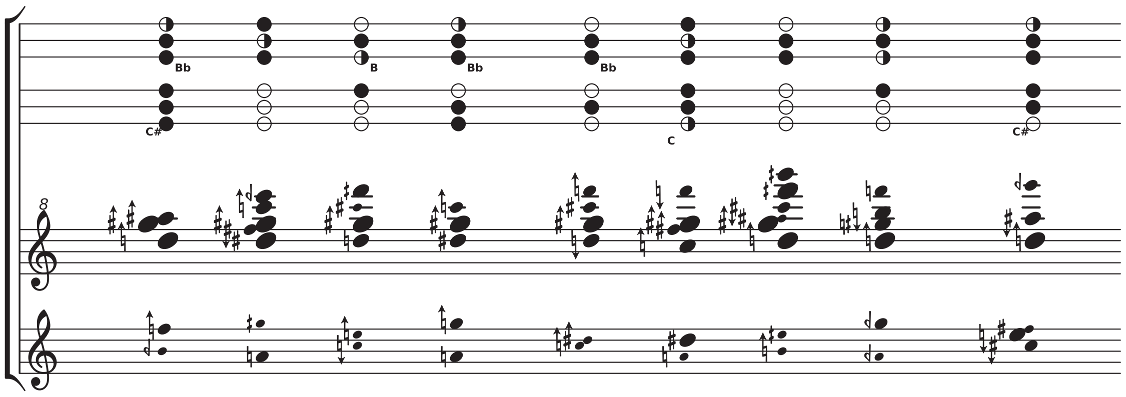


Figure 8

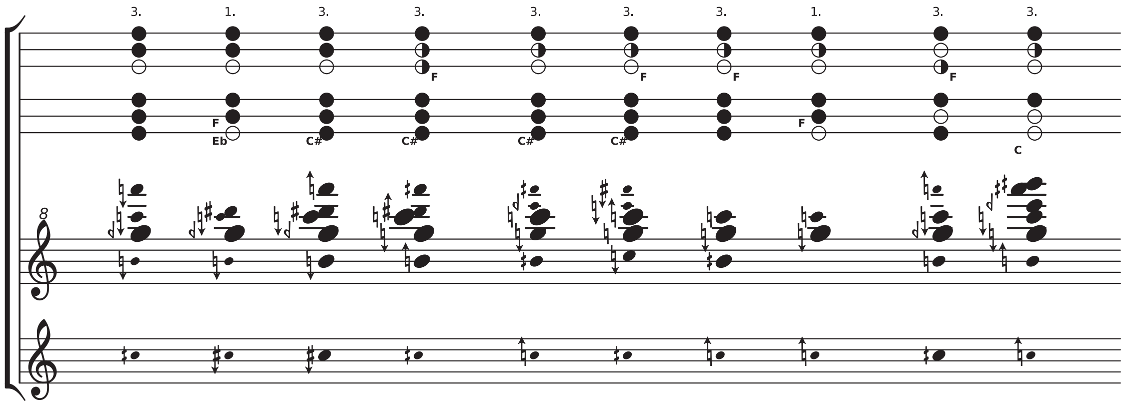


Figure 9

```
var example = new Solver(
    10, // length of the output sequence
    oboeMultiDB, // multiphonics database
    // constraint
    fingerings => {
      // all different
      return fingerings.predAllDifferent() &&
      // constraint on successive fingerings
      fingerings.predSuccessive(
          (f1,f2) =>
            // fewer than three different pitches with 0.25 tolerance
            f1.pitchesList()
            .mutualSetDifferenceTolerance(
                f2.pitchesList(),0.25).length < 3);
    });
```

Figure 10

Figure 11. A neighbor group: a set of fingerings that are all neighbors of each other.

Figure 12. A sequence of ten fingerings in which the transitions become increasingly difficult from fingering to fingering.

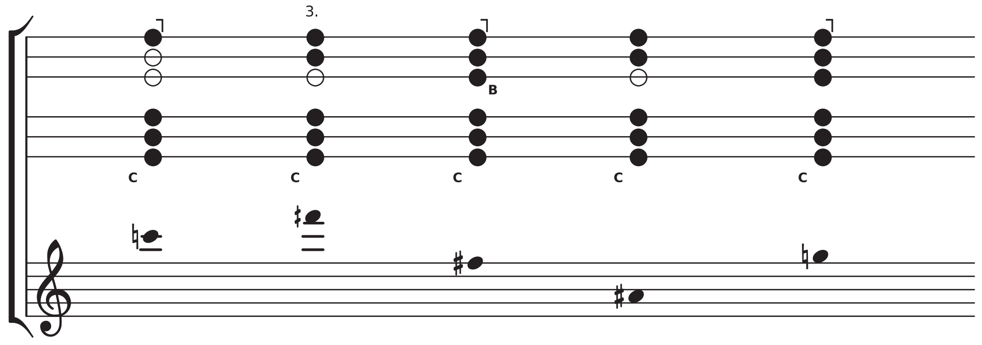


Figure 11

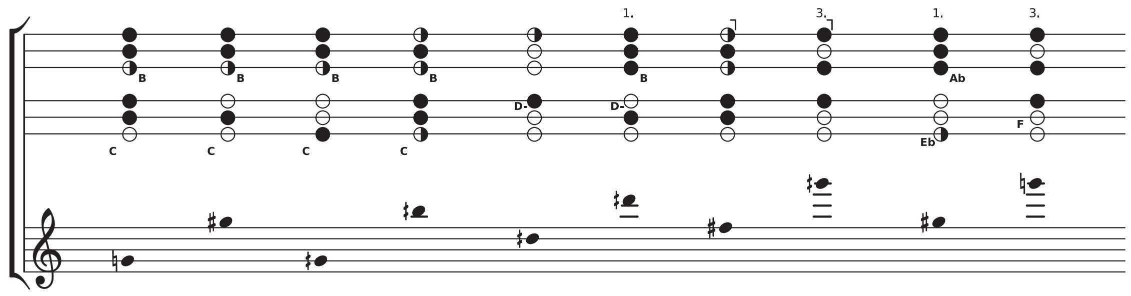


Figure 12

types of musical instruments will require other forms of modeling.

There are a number of improvements and extensions that can be made to the backtracking search function. A more-optimized search algorithm will enable the system to produce longer structurally rare sequences. At the moment, all constraints are hard constraints. I plan to enable the definition of soft constraints that define cost functions.

Moreover, the system and interface for defining representations of other woodwind instruments will be provided. The entering of fingerings will also be made available through the Web application.

Acknowledgments

This article is an extended and updated version of the paper "OboeJS: A Constraint-Based Approach to Instrument-Specific Algorithmic Composition" presented at the 2018 International Computer Music Conference (Döbereiner 2018). I would like to thank the Editor, Doug Keislar, for his constructive comments. This project was supported by the Transpositions: Artistic Data Exploration research project by Gerhard Eckel, Michael Schwab, and David Pirrò, funded by the Austrian Science Fund FWF (PEEK, AR 257).

References

Almeida, A. L. F., et al. 2004. "Physical Model of an Oboe: Comparison with Experiments." In *International Symposium on Musical Acoustics*, pp. 112–115.

Anders, T., and E. R. Miranda. 2010. "Constraint Application with Higher-Order Programming for Modeling Music Theories." *Computer Music Journal* 34(2):25–38.

Cassidy, A. 2002. *String Quartet*. Available online at aaroncassidy.com/music/stringquartet.htm#Notes. Accessed January 2019.

Dechter, R. 2003. *Constraint Processing*. San Francisco: Morgan Kaufmann.

Deforce, A., and R. Barrett. 2001. "The Resonant Box with Four Strings: Interview on the Musical Esthetics of Richard Barrett and the Genesis of His Cello Music." Available online at issuu.com/arnedeforce/docs/the_resonant_box_with_four_strings. Accessed January 2019.

DeLanda, M. 2011. *Philosophy and Simulation. The Emergence of Synthetic Reason.* London: Continuum.

Döbereiner, L. 2018. "*OboeJS*: A Constraint-Based Approach to Instrument-Specific Algorithmic Composition." In *Proceedings of the International Computer Music Conference.* Available online at icmc2018.org/papers-pgm-2. Accessed January 2019.

Essl, K. 2007. "Algorithmic Composition." In N. Collins and J. d'Escrivan, eds. *Cambridge Companion to Electronic Music*. Cambridge: Cambridge University Press, pp. 107–125.

Fernández, J. D., and F. Vico. 2013. "AI Methods in Algorithmic Composition: A Comprehensive Survey." *Journal of Artificial Intelligence Research* 48(1):513–582.

Ferneyhough, B. 1995. *Collected Writings.* London: Routledge.

Hiller, L., and L. Isaacson. 1959. *Experimental Music: Composition with an Electronic Computer*. New York: McGraw-Hill.

Knuth, D. E. 1997. *The Art of Computer Programming: Fundamental Algorithms*. Vol. 3. Boston: Addison-Wesley.

Koenig, G. M. 1983. "Aesthetic Integration of Computer-Composed Scores." *Computer Music Journal* 7(4):27–32.

Lachenmann, H. 1996. *Musik als existentielle Erfahrung: Schriften 1966–1995.* Wiesbaden, Germany: Breitkopf & Härtel.

Laurson, M., and M. Kuuskankare. 2000. "Towards Idiomatic Instrumental Writing: A Constraint-Based Approach." In *Proceedings of the Symposium on Systems Research in the Arts*.

Longo, G. 2009. "Critique of Computational Reason in the Natural Sciences." In E. Gelenbe and J.-P. Kahane, eds. *Fundamental Concepts in Computer Science.* London: Imperial College Press, pp. 43–70.

McCormack, J., et al. 2009. "Generative Algorithms for Making Music: Emergence, Evolution, and Ecosystems." In R. T. Dean, ed. *The Oxford Handbook of Computer Music*. Oxford: Oxford University Press, pp. 354–379.

Miranda, E. R. 2001. *Composing Music with Computers*. Newton, MA: Butterworth-Heinemann.

Nierhaus, G. 2009. *Algorithmic Composition: Paradigms of Automated Music Generation*. Berlin: Springer.

Rutz, H. H., and D. Pirrò. 2018. "Körper." In *Proceedings of the Conference on Computation, Communication, Aesthetics, and X*, pp. 176–180.

Sandred, O. 2010. "PWMC, a Constraint-Solving System for Generating Music Scores." *Computer Music Journal* 34(2):8–24.

Scruton, R. 1997. *The Aesthetics of Music*. Oxford: Oxford University Press.

Terzidis, K. 2003. *Expressive Form: A Conceptual Approach to Computational Design*. New York: Spon Press.

Van Cleve, L. 2014. *Oboe Unbound: Contemporary Techniques*. Lanham, MD: Rowman and Littlefield.

Veale, P., and C.-S. Mahnkopf. 2005. *The Techniques of Oboe Playing: A Compendium with Additional Remarks on the Whole Oboe Family*. Kassel, Germany: Bärenreiter.

Whitehead, A. N. 1978. *Process and Reality*. New York: Free Press.