Motif Set Assignment Problem in the Compositional Process of “Musical Rally”

Tsubasa Tanaka
Tokyo University of the Arts,
IReMus-CNRS-Université de Paris Sorbonne-UMR8223
tanakatsubas@gmail.com

ABSTRACT

Creating formalized musical structures sometimes requires solving complex constraint satisfaction problems or discrete optimization problems. However, it is not very easy for composers to integrate specific tools such as constraint solvers or integer programming solvers into their compositional practice. To demonstrate the relevance of constraint satisfaction and discrete optimization to the computer-aided composition and the possibility of integrating them in a common musical programming environment, this paper presents the *motif set assignment problem*, a constrained optimization problem that appeared in the process of composing the author's piece *Musical Rally*, which is a guided collective improvisation piece based on motif collections that are assigned to respective sections. Also, this paper shows a practical implementation of an algorithm to solve this problem with OpenMusic and its library OM-Darwin, without using external tools.

1. INTRODUCTION

Musical scores, especially in the form of classical western notation, consist of discrete information such as pitchs, durations, and meters, which can be expressed by natural numbers (midi pitch numbers, rhythmic ratios, etc.). Naturally, discrete mathematics is closely related to musical problems. Since the twentieth century, musical set theory has been developed and musical pieces have been mathematically described with pitch-class sets and their transformations. However, it was only recent decades that large-scale discrete problems in music have been solved by computational methods such as constraint programming (CP) and integer programming (IP) because complex problems are sometimes NP-hard and need fast computers and efficient algorithms. CP and IP are paradigms of declarative programming with which users describe mathematical relationships between variables to be satisfied (constraints, equalities, inequalities, etc.) contrary to ordinary procedural programming. Therefore, the formulation of the problem and the algorithm to find its solutions can be clearly separated. Thanks to this, users of CP and IP can concentrate on formulating the problems because the solvers already have efficient algorithms for general problems. CP is mainly for constraint satisfaction problems [1] and IP is for optimization problems in which variables take only integer values [2].

An example of a typical CP problem is *N Queen problem* and that of IP problem is *traveling salesman problem*. In the field of music, the problem of finding *all interval series*, which is a 12-tone row that has eleven different intervals between adjacent notes, is one of the typical problems. There have been some larger-scale applications of CP and IP. The problem of melodic segmentation of polyphonic music was formalized as a *set partitionning problem* and solved with IP in 2014 [3]. The link between the problem of generating Milton Babbitt's *all-partitions allay* and the *set covering problem* was discussed in [4] and a solution was found with CP in 2016 [5].

Before these applications, CP has been applied to real compositional practice since the 1990s [1] although the problems were relatively small. Notably, musical programming environments PatchWork and OpenMusic were developed in this direction and used by composers of contemporary classical music. In general, music composition consists of many choices of notes or sounds from a vast number of combinatorial possibilities without the capability of knowing most parts of them. This is as if composers have to walk in an unknown dark space without lights. In such a situation, some techniques like the 12-tone technique can be a light that gives them sight. The 12-tone technique can give a lot of objective information about where the composer is in the space of possibilities by rationally limiting possible choices through the specific 12-tone rows and transformations. CP and IP can play a similar role to the 12-tone technique in more generalized ways using the power of computers. Contrary to other techniques such as supervised machine learning, CP and IP have merit that they assure the satisfaction of a set of rules that composers want to pose and that composers can design the objective function from their own criteria explicitly.

Here, the author does not insist that the satisfaction of the rules or the optimality always means that the generated piece is good. This is the same as the fact that a strict application of the 12-tone technique does not mean the goodness of the result. However, CP or IP can at least give the sight to the composers so that they can proceed with the compositions with a certain degree of confidence. This is an important aspect of compositional technique based on rules and indicators.

There have been several libraries of OpenMusic for constraint programming such as OMCloud, OMGecode, OMRC, and OMCS. However, to the author's experience, OMCloud, OMRC, and OMCS can only solve relatively small problems in a realistic sense of time. OMGecode is a library for OpenMusic that uses powerful external constraint solver Gecode [1]. However, OMGecode can use only a part of the functions of Gecode and it is not supported for recent versions of OpenMusic. As for the integer programming, there is no library that is specific to music, to the best of the author's knowledge.

Therefore, we currently lack sufficiently powerful tools dedicated to finding exact or optimal solutions of complex constraint satisfaction problems or discrete optimization problems even in OpenMusic. Furthermore, even in the case one uses CP or IP solvers, he/she sometimes has to create a program that generates a program that defines vast number of variables and constraints, which would be a hard work for ordinary composers. Thus, integrating CP or IP in the practice of composition is not very easy. There are large gaps between the practice of scientific researchers and that of composers, which would be worth taking into account.

Örjan Sandred, who has developed OpenMusic library OMRC and OMCS, is one of the prominent composers who have explored constraint-programming-based approaches in composition [6–9]. He has also improved the libraries and created more efficient constraint solvers PWMC (Patch Work Musical Constraints) and Cluster Engine, which are adapted to music-specific structures. The use of such music-specific constraint solvers might be a possible choice for some composers.

However, changing one's platform only for CP or IP would be practically not easy. Not everyone can pay the cost of leaving his/her familiar platform. This motivated the author to search for an alternative implementation of a constrained optimization problem that appeared in the compositional process by using only OpenMusic, a widely-used music programming environment for composers. In common with Max or Pure data, OpenMusic is based on the visual programming paradigm. Thanks to that, it has high readability and intuitiveness. It is used in many music institutions and there are many composers who are used to these environments even if they do not have a coding background of ordinary programming languages. Therefore, in order to extend the possibilities of composition, it would be pragmatic and beneficial to integrate mathematical methods including CP and IP with such composers' customs of music programming.

In this context, this paper presents a case study of such a constrained optimization problem that the author dealt with through the compositional process of a piece *Musical Rally* (2020), which is a guided collective imporovization piece based on motif collections. In this piece, four motifs from a candidate motif set are assigned to respective sections and finding an optimal or semi-optimal solution to this problem is at the core of the compositional process. The author named this problem the *motif set assignment problem* and implemented the algorithm that solve it with OpenMusic.

This paper is organized as follows. Section 2 explains the intention of composing the piece *Musical Rally*. Section 3 formalizes the motif set assignment problem, which determined the score material of the piece. Section 4 describes the implementation of the algorithm to solve it with OpenMusic and its library for the genetic algorithm, OM-Darwin. Section 5 concludes the present paper.

2. A GUIDED COLLECTIVE IMPROVISATION PIECE "MUSICAL RALLY"

Musical Rally is a piece for participatory collective guided improvisation that was inspired by the musico-political movement "Nueva Cancin" (New Song) and the popular protest "cacerolazo" (stew pot hitting) in South America. It is supposed to be performed by a number of musicians (not necessarily professional ones) who improvise according to some instructions and score materials (motif sets). The number of musicians is essentially not fixed and participation of as many people as possible is preferable. The style is also somewhat similar to Terry Riley's *In C* (1964).

The intention of composing this piece is to make musicians possible to play together without practice or special capacities by introducing simple rules and materials for guided improvisation. The score material consists of eleven sections and the score of each section is based on four motifs. There are seven parts and all of the players share the same motifs with some transformations (temporal augmentations, modal transpositions, and superpositions. A player is assigned to one of the parts that he/she can play by his/her instrument. Each player chooses and plays one of the motifs one after another. The motifs can be varied to a certain degree and rests can be inserted between motifs freely. Figure 1 shows an example of the four-motif set for a section of *Musical Rally*. All of the motifs are transformed citations from a Victor Jara's song *Manifesto*, as a mark of respect to his works and the ways people in South America have reacted to the political and economic situations with music.

The timings of the arrival and the leaving of individual players are different. They temporarily occupy the stage of the concert and leave it independently like in a meeting. The players are encouraged to take pictures to be uploaded on social networking services when they leave. In this way, the time and space of the concert are deconstructed.

The process of composition is divided into three steps:

1. transcription of Victor Jara's song *Manifesto* into motifs to form a candidate motif sets for the score material.

2. determination of the sets of motifs for the sections.

3. computer-simulated realization of the guided- improvisation to confirm how the piece sounds.

Concerning step 1, the author dictated the melody of the song *Manifesto* and that resulted in 29 short motifs that contain from one note to four notes. From these motif candidates, four motifs are assigned to each section in step 2. Step 2 is the main procedure of the composition and it is formulated as a mathematical problem of assigning

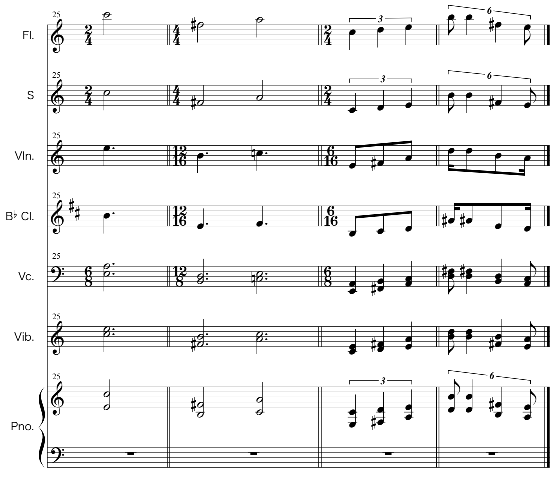


Figure 1. An example of the four-motif set for section 7 from *Musical Rally*, which is denoted X_7 in the next section. The same motifs are used for every instrument with some transformations (temporal augmentations, modal transpositions, and superpositions.

optimal motif sets to the sections. Thus, the sets are determined at once as a solution of the problem. This is explained in detail in the next section. The computer simulation of the realization of the improvisation in step 3 can be listened to on SoundCloud [10].

3. MOTIF SET ASSIGNMENT PROBLEM

The piece consists of $N = 11$ sections and each section of the piece has four motifs (therefore, 44 motifs in total with some duplications of the same motifs between the sections). There are 29 candidates of motifs taken from the song *Manifesto*. This means that there are $({}_{29}C_4)^N$ ways of assigning four motifs to each section, which is an exponential growth over the number of sections N. From these possibilities of assignment, the most desirable one that is suitable for the musical form of the piece should be chosen.

The aesthetical aim of this compositional method is to realize a gathering of musicans that has a dense interaction between them by sharing a small number of motifs and to easily generate hocket-like polyphonic textures. In order to avoid monotony, it is desirable that as many different motifs as possible are selected (diversity of motifs) and that the sets of four motifs are as different as possible over sections (diversity of motif combinations).

Specifically, the author has set the following constraints (Let X_i denote the set of four motifs assigned to section i and S denote the set of all of the 29 motif candidates.):

- Each section has motifs of one note, two notes, three notes, and four notes respectively.
- The lower limit L_i^P and the upper limit U_i^P of the pitches that appear in the four motifs of the section i are posed so that the registers, which are relevant to the transition of tension between the sections and musical form, are controlled.
- $Card(X_i)$, the number of different pitches (the cardinality of pitches) contained in the four motifs in the section i, is fixed as C_i (typically as 6) to assure strong modalities.
- The lower limit L_i^M and the upper limit U_i^M of $MotDurCard(X_i)$, the cardinality of the durations of motifs in the section i, are posed so that the metric complexity is controlled.
- The lower limit L_i^D and the upper limit U_i^D of $DurCard(X_i)$, the cardinality of rhythmic durations contained in the motifs in the section i, are posed so that the complexity of rhythm is controlled.

These criteria can be expressed as the following constraints.

(1) $X_i = \{x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}\} \subset S \; (i = 1, 2, ..., N)$,

(2) $NumNotes(x_{i,j}) = j \; (i = 1, 2, ..., N, j = 1, 2, 3, 4)$,

(3) $Card(X_i) = C_i \;\; (i = 1, 2, ..., N)$,

(4) $\forall x \in X_i, \forall p \in Pitches(x), \; p \in [L_i^P, U_i^P] \;\; (i = 1, 2, ..., N)$,

(5) $DurCard(X_i) \in [L_i^D, U_i^D] \;\; (i = 1, 2, ..., N)$,

(6) $MotDurCard(X_i) \in [L_i^M, U_i^M] \;\; (i = 1, 2, ..., N)$.

Also, the author has designed the following global characteristics of X that should be as large as possible:

- The number of different pitch patterns of the motifs over the sections (denoted by $P(X)$) should be large to cover as many different motifs as possible to assure the variety of sections ($X = X_1 \cup X_2 ... \cup X_N$ denotes the union of motif sets over the sections.).
- The number of different combinations of the pitch patterns of the motifs over the sections, especially the combination of the three-notes motif and the four-notes motif, sould be large so that the motif sets of the sections are not similar to each other. Let $C(X)$ denote the number of different combinations of the pitch patterns of the three-note motif and four-note motif over the sections.
- The number of different rhythmic patterns of the motifs over the sections should be large so that the "rhythmic mode" of a section is different from those of other sections. Let $R(X)$ denote the number of different rhythmic patterns of the motifs over the sections.

These characterstics should be maximized as much as possible at the same time. To achieve this, the author defines an objective function to be maximized as the product of $P(X)$, $C(X)$, and $R(X)$.

Considering all of these constraints on the sections and the global characteristics, the problem of finding an optimal assigment of motif sets to the sections (Let us call it the *motif set assignment problem*) is formalized as the following constrained descrete optimization problem:

maximize $P(X) \cdot C(X) \cdot R(X)$
subject to $X_i \in A_i \ (i = 1, 2, ..., N)$,

where A_i is the set of solutions of the constraints (1)-(6).

Basically, the "goodness of the solution" is measured by these three indicators. However, the author does not insist that the optimality of the solution immediately means the goodness of the generated music. What is important is that the solution provides a reference point in the space of possibilities (and possibly a candidate for the final decision) whose certain characteristics are verified.

Another good thing about this method is that it can consider the global distribution of many motifs at once using the power of computation. In traditional motif-based manual composition, only a small number of motifs are considered at once and the global optimality is hard to be verified.

4. IMPLEMENTATION

This objective function of this problem is non-linear and the constraints are not in the form of linear constraints. Because IP solvers are mainly designed to solve linear problems, the use of CP solvers would be more suitable for this problem. However, for composers, it is not very easy to adopt such solvers because they are not necessarily used to other programming environments than those for music. From a practical point of view, it is ideal that composers can implement any problems within their familiar programming environments.

Therefore, in this section, an implementation of the algorithm to solve the problem with OpenMusic, one of the most common programming environments for computer-aided composition, and its libraries is shown. This implementation does not use any external programs dedicated to constraint programming or discrete optimization (As for the basic knowledge of OpenMusic, refer to [11].).

The strategy of this implementation is two-fold: firstly, constraint filters for enumerating the candidates of four-motifs sets for each section that satisfy the constraints from (1) to (6) are created. Secondly, an optimizer of the objective function is created using OM-Darwin, an OpenMusic library for the genetic algorithm created by Geof Holbrook [12, 13]. This is possible partly because the part of constraint satisfaction and the part of optimization are designed to be separated by the definition of the problem so that we can find the solutions relativly easily. Combining these two, we can create a solver of the problem only using basic functions and some libraries of OpenMusic. This two-fold strategy is, to some extent, similar to a method for the *nurse scheduling problem* [14] in witch possible patterns are enumerated in advance to deal with the complexity of the problem and tight constraints that make us difficult to find feasible solutions.

The genetic algorithm is an algorithm for optimization that simulates the evolution of genes. It generates a population of genes that represent instances of an object (those of motif sets, in this case) that have better and better values of fitness function by means of crossover and mutation through generations. The genes adopted for the crossover and mutation are selected based on the values of the fitness function. In this way, better genes have more chances to be transmitted to the next generations. Although the genetic algorithm does not assure the optimal solution, it is suited to obtain semi-optimal solutions in a realistic sense of time.

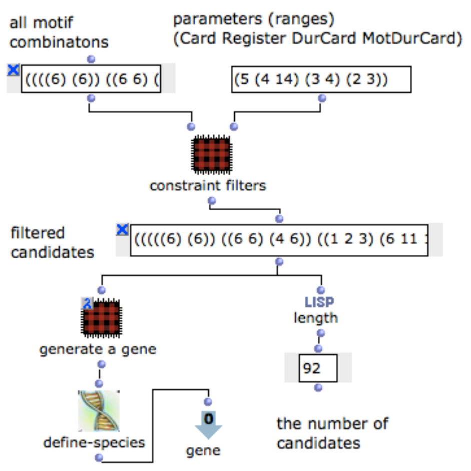
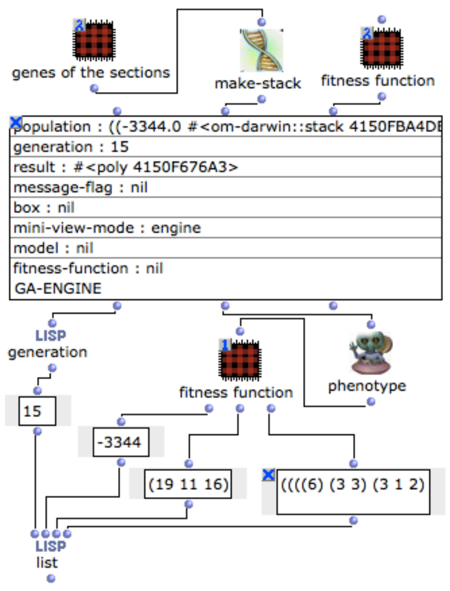


Figure 2. Constraint filters for the motif sets of a section and the definition of a gene of a section for OM-Darwin.

Figure 2 shows the part of constraint filters that create every possible list of four motifs (the sets of four motifs that satisfy the constraints) that can be assigned to a section. Inside the upper object "constraint filters", motif sets that do not satisfy at least one of the constraints (1)-(6) are excluded. From the motif sets that passed this flter, the subpatch "generate a gene" selects one of them (by the object *nth-gene* of OM-Darwin inside the subpatch) and the object *define-species* of OM-Darwin encodes it into a gene of the section. For generating all of combinations of the sets of four motifs that are passed to the constraint filters, the function *combine* of the OM library Combine was used.

Figure 3 shows the optimizer part. It uses *GA-ENGINE*, a central function of OM-Darwin as the engine of the optimization. All of the genes of eleven sections are concatenated by the *make-stack* object. GA-engine takes two inputs, the gene definition patch "genes of the sections" and the fitness function patch "fitness function". This fitness function is -1 times the objective function defined above, which is to be minimized. GA-ENGINE evaluates the values of the fitness of genes and generates genes of the next generation from those of current generation by crossover and mutation.

The values of the fitness function of the best genes for the respective generations are shown in Figure 4. Table 1 shows the values $P(X)$, $C(X)$, $R(X)$ of the best solution found, whose fitness value is -3344. The numbers inside the parentheses are the upper limits of the three values, which correspond to, respectively, the number of different pitch patterns in the 29 motif candidates, the number



```
population : ((-3344.0 #<om-darwin::stack 4150FBA4DE
generation : 15
result : #<poly 4150F676A3>
message-flag : nil
box : nil
mini-view-mode : engine
model : nil
fitness-function : nil
GA-ENGINE
```

Figure 3. A GA-ENGINE object as the optimizer.

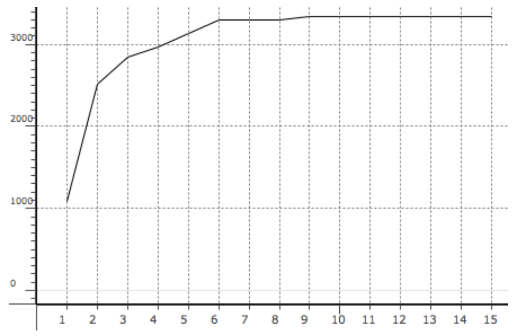


Figure 4. The transition of the fitness values (timed -1) of the best genes through the generations of the genetic algorithm.

of sections, and the number of different rhythmic patterns in the 29 motifs. Figure 5 shows the number of different candidates for respective sections that were not excluded by constraint filters. The motifs of Figure 1 correspond to section 7 of the best solution found. It took several minutes of calculation time until the 15th generation of the genetic algorithm.

From Table 1, we sees that C(X) and R(X) achieved the optimal values. As for P(X), the optimality is not confirmed. However, the upper limit 25 only means there are 25 different pitch patterns in the candidates and does not mean there necessarily exists a solution that achieves it. In this sense, the best solution found is acceptable and has the possibility of being an optimal solution. The author, as the composer, admitted that the indicators are sufficiently good and used the solution in the composition without modifying it.

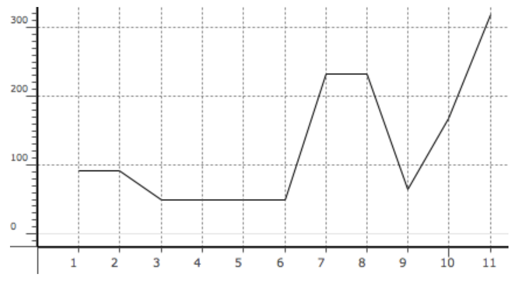


Figure 5. The number of four-motif-set candidates that satisfy the constraints (1)-(6) for respective sections.

P(X)	C(X)	R(X)
19 (25)	11 (11)	16 (16)

Table 1. The numbers of different pitch patterns (P(X)), that of different combinations of pitch patterns of the three-note motif and the four-note motif (C(X)), and that of different rhythmic patterns (R(X)) across the sections (for the best solution). The numbers inside the parentheses are the upper limits.

5. CONCLUSION

In this paper, the motif set assignment problem, which was at the core of the compositional process of the author's piece *Musical Rally* was presented. A solver of this constrained optimization problem was implemented with OpenMusic and OM-Darwin without using external programs so that certain composers can easily implement it. The algorithm has found a solution that satisfies the constraints and that has optimal values at least for two of the three criteria. To the best of the author's knowledge, OM-Darwin is currently the only powerful library that can be used as an engine of optimization in OpenMusic. As a composer, the author hopes that more libraries for mathematical tools such as integer programming, neural networks, reinforcement learning are created in the future so that composers can easily integrate mathematical thinkings into their compositional practice. OM-AI, which is a library for a clustering algorithm, at this moment, is another good example of recent libraries in music programming environments [15]. This type of library will promote composers to develop new compositional methods that utilize the power of mathematics and computation.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 19K13024.

6. REFERENCES

[1] C. Truchet and G. Assayag, *Constraint programming in Music*. ISTE Ltd and John Wiley & Sons, Inc., 2011.

[2] G. Nemhauser and L.Wolsey, *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.

[3] T. Tanaka and K. Fujii, "Melodic pattern segmentation of polyphonic music as a set partitioning problem," in *Proceedings of International Congress on Music and Mathematics*, Puerto Vallarta, 2014, pp. 291–298.

[4] T. Tanaka, B. Bemman, and D. Meredith, "Integer programming formulation of the problem of generating milton babbitts all-partition arrays," in *Proceedings of International Society for Music Information Retrieval Conferencs*, New York, 2016, pp. 171–177.

[5] ——, "Constraint programming approach to the problem of generating milton babbitts all-partition arrays," in *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming*, Toulouse, 2016, pp. 802–810.

[6] Ö. Sandred, "Kalejdoskop for clarinet, viola and piano," *C. Agon, G. Assayag and J. Bresson (eds.) The OM composer's book 1*, pp. 223–235, 2006.

[7] ——, "Approaches to using rules as a composition method," *Contemporary Music Review*, vol. 28, no. 2, pp. 149–165, 2009.

[8] ——, "Pwmc, a constraint-solving system for generating music scores," *Computer Music Journal*, vol. 34, no. 2, pp. 8–24, 2010.

[9] ——, *The Musical Fundamentals of Computer Assisted Composition*. Audiospective Media, 2017.

[10] T. Tanaka, "Musical rally (computer-simulated realization)," *https://soundcloud.com/tsubasa-tanaka*, 2020.

[11] C. Agon, G. Assayag, and J. B. (eds.), "Appendix: Openmusic," *The OM composer's book 1*, pp. 261–268, 2006.

[12] G. Holbrook, *Optimizing Future Perfect: A Model for Composition with Genetic Algorithms*. Columbia University, 2015.

[13] ——, *"OM-Darwin: Generative and descriptive aspects of genetic algorithms" in J. Bresson, C. Agon, G. Assayag (eds.) The OM composer's book 3*. DELATOUR FRANCE/Ircam-Centre Pompidou, 2016.

[14] A. Ikegami and A. Niwa, "A subproblem-centric model and approach to the nurse scheduling problem," *Mathematical Programming*, vol. 97, pp. 517–541, 2003.

[15] A. Vinjar and J. Bresson, "Om-ai: A toolkit to support ai-based computer-assisted composition workflows in openmusic," in *Proceedings of 16th Sound and Music Computing conference*, Málaga, 2019, pp. 84–85.