Symbolic regression as a computer-aided music tool for analysis and composition

José Henrique Padovani[1,2]*, **Jônatas Manzolli** [1]

[1] mus[3] - Musicology, Sonology and Computing – Music Department – UFPB
Federal University of Paraíba, Campus I - Castelo Branco, João Pessoa - PB, 58051-900

[2]NICS - Interdisciplinary Nucleus of Sound Communication – UNICAMP - University of Campinas
Rua da Reitoria, 165 - Cidade Universitária "Zeferino Vaz", Campinas - SP, 13083-872

josehenriquepadovani@pq.cnpq.br, jonatas@nics.unicamp.br

Abstract. *The paper presents a new OpenMusic library that implements a Genetic Programming method of Symbolic Regression on sets of input data-points and seeks for a Common Lisp function (S-expression) that can be used either to create mathematical models that could potentially help to understand the mathematical behavior of the input data or to generate parameters in computer-aided composition. Stressing that a number of issues must still be addressed to improve the proposed library, the paper presents some of the strategies to do this and to make Symbolic Regression a practical tool in computer-assisted music composition and analysis.*

1. Introduction

Despite the established distinction between algorithmic music and computer-aided composition (CAC) and the related differentiation regarding the role and prevalence of algorithms in the creative processes [12, 29, 5], music composition and analysis with algorithmic tools demand, to a greater or lesser extent, the formalization and modeling [38, 7] of musical materials and processes. While algorithmic and mathematical *models* of musical procedures and materials are usually developed by composers, analysts or researchers, *genetic programming* (GP) [20, 18, 28] techniques can be explored to achieve optimal models that could be employed in different analytical and compositional computer-aided music (CAM) contexts.

Our work addresses the question of automatically generating rough *algorithmic models* in CAM environments by using GP techniques such as *Symbolic Regression* (SR). Currently, the process is implemented as library for the CAM environment *OpenMusic* and as a external application in SBCL. While the project is still in its first stages, the proposed approach provides a new compositional and analytical tool to create *mathematical functions* and *symbolic expressions* that could optimally describe or outline data-sets such as two or more interrelated musical parameters (pitch, onset, duration, audio features, etc). Used in iterative processes, those expressions can be employed to generate new values for the selected parameters or as a starting point to develop more accurate models.

2. Algorithmic models for composition and analysis

In computer-aided, algorithmic and generative music, as well as in other specific musical practices that directly use computational algorithms in music composition and analysis,

*Supported by CNPq.

musical materials, procedures and processes are to be described as data structures and *algorithms* or *computational methods* [27, p. 4]. That is to say that they need to be declared in any computer language in way that both the relevant properties of music materials and the steps/methods of musical procedures and processes become unambiguously and formally stated so that they can be automatically created, processed, managed or operated by a computer.

While in the field of music composition this formalization requirement can be identified already in seminal works on composition with algorithmic tools [19, 44] as well as in more recent texts of composers dealing with computer-aided music (CAM) environments [1, 2], in the field of musicology the use of these same tools has lead to new approaches in music analysis, specially *analysis by modelling* or *formalized analysis* [38, 39]. The formalized statement of musical materials and processes results in *algorithmic models* that can be used to generate new musical ideas or to reconstitute given music excerpts (*local models*) or entire pieces (*global models*) [38, p. 91].

Given the fact that an *algorithmic model* does not describe music as a score [42, p. 1-6], being rather a formal description of the structures, procedures and processes that can generate music notation or synthesized sounds, it becomes an invaluable tool both in composition and in musicology. In composition, these tools provide means to prototype a musical piece or a passage with different input parameters and simulate the possible results of a formalized compositional process. In computational musicology, on other hand, algorithmic modelling makes it possible not only to better understand the intrinsic features of materials, processes and procedures of a given piece or compositional technique but also offers a glimpse of other possible results that the *algorithmic model* of the analyzed piece could yield. Whereas this approach to music analysis has lead to a number of *algorithmic models* of different pieces and styles [35, 11, 26, 14], also the modeling of counterpoint and other compositional techniques [3] used in many algorithmic and computer-aided compositions since the *Illiac Suite* (1957) [19] actually apply *formalized analysis* in creative processes.

In the context of creating *algorithmic models* either to compose or to analyze music, the process of *formalizing* structures and processes to develop compositional procedures is not just a particular stylistic approach. Indeed, it becomes, rather, "a necessity and, even, an imperative" [32, p. 234] to the development of what may be called a "*solfège* of models": the ability to control and "to master both the musical result of a given generative model and the relationship between the graphic and/or textual representations of some music software and the music outcome" [31, p. 142]. However, to compose or to analyze music pieces or excerpts by using algorithmic tools does not imply that all the compositional procedures and materials are readily reducible to *algorithmic models* [39, p. 66]. Indeed, even if the "rule-based nature" [17, p. 108] of some musical ideas, compositional techniques and specific music pieces makes them to be specially suitable for being formally described by *algorithmic models*, the very process of coding them with computer languages is often extraneous to the original creative intuition or, in the case of *formalized analysis*, to the composer techniques. Further, there are cases in which the behavior of given musical parameters (that may be empirically taken from an improvisation or from a musical piece, for example) do not seem to be generated or easily described by any known rule or formalized process: the model, if exists, is unknown.

Bearing in mind that *algorithmic models* for composition and analysis do not necessary reflect the way a composer thought or created a given piece or excerpt [39, p. 65] it is possible to hypothesize that, even music pieces or excerpts that apparently are not based on any rule system or methodical procedures, there are *models* to be found that

could optimally generate similar outcomes for interrelated variables such as pitch, onset, etc.

3. Genetic Algorithms, Genetic Programming and computer-aided music

Among many artificial intelligence approaches to this problem, evolutionary computing provide tools that may be used in the heuristic search for optimal models. Briefly, *genetic algorithms* (GA) [20, 18] and *genetic programming* (GP) [28] processes rely on data-structures and algorithms that are based on evolutionary theories and on the concept of *natural selection*. As the *genome* or digital representation of given *individuals* (an algorithm or a data-structure) is transformed by a series of bio-inspired processes such as *mutation* and *cross-over*, *individuals* are selected to "survive" through a series of successive iterations (*generations*) according to their *adaptation* to any sort of evolutionary pressure – usually given by a *fitness function*. While in GA the *individuals* are usually any sort of digitally represented data-structure, in GP they generally are *algorithms*, *symbolic expressions* or *computational instructions* that are automatically generated and selected according to its *fitness rate* in solving a given problem.

The use of GA and GP techniques in computer music is not new. Since the 1990s, a number of artists and researchers have applied evolutionary computation techniques in music composition and in computer music research. Many of these works use these techniques as a means to generate music [23, 41, 6, 24, 10, 33, 36, 25] or as an heuristic tool to solve specific problems – such as finding optimal FM synthesis parameters [22, 21], to automatically extract descriptors from audio signals [45] or to create mimetic orchestrations[13] from an input audio target, for instance. However, despite being more frequently employed in generative algorithmic music and as tools to solve specific problems in comparison with its use in composition/analysis contexts with CAM environments, the non-deterministic behavior and the heuristic potential of evolutionary computational techniques makes them appealing to the interactive work-flow that typically characterizes computer-aided composition and formalized analysis paradigms. Particularly, we envisage potential benefits for the composer and for the musicologist in using CAM tools not only to model musical materials and procedures but, also, as a heuristic instrument in the search for unknown models in the form of *mathematical functions* and *symbolic expressions* that may be used in iterative computations.

4. S-expressions and symbolic regression

In visual CAM environments such as *OpenMusic* (OM) [7, 8] and *PWGL*[30], both based on *Common Lisp* (CL), algorithms are visually represented in the form of *patches*. "Boxes" may represent functions, data-types (lists, integers, floating-point numbers, strings, etc.), methods or classes that are interconnected by "chords" to create more complex computational procedures. Internally, though, as in any *Common Lisp* code, algorithms are formed by *symbolic expressions* or *s-expressions*: trees of lists that combine atoms (numbers, strings, symbols and some other non parenthesized structures) and lists that may contain simple data-types or that may call functions and macros that are sequentially evaluated to yield their outputs to the respective parent-lists. Hence, more complex algorithms and computer applications can be thought as a tree of lists that either contain data (terminals) or functions.

Since that in *Common Lisp* algorithms can be written as *s-expressions* and that *s-expressions* themselves are represented as lists and trees, algorithms and iterative computation processes can be generated automatically by creating well-formed trees, sub-trees

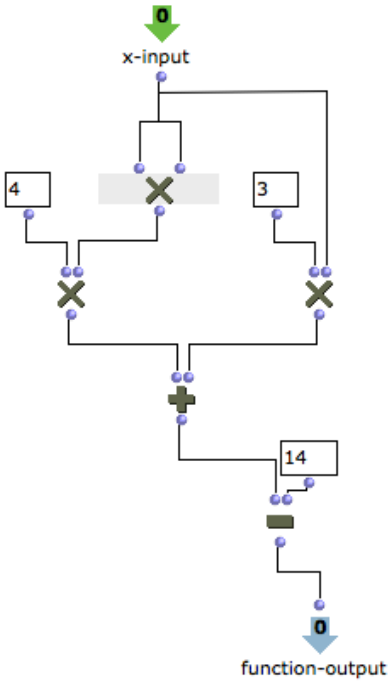

Figure 1: An *OpenMusic patch* representing the polynomial $4x^2 + 3x - 14$. In raw *Common Lisp*, the function would be written as the symbolic expression `(- (+ (* 4 (* x x)) (* 3 x)) 14)`.

and lists from a given set of functions and terminals and specific instructions regarding each function's number of arguments and respective data-types. If the tree has one or more mathematical functions, one of the terminals is an independent variable x and numerical constants are created within a range of random numbers, the generated *s-expression* can be interpreted as a mathematical function $f(x)$ that may be used in iterative processes, as an algorithm, to generate outputs from given inputs.

While the generation of blind random *s-expressions* would not be an efficient method to cover the search space of all possible *s-expressions* for a given set of func-

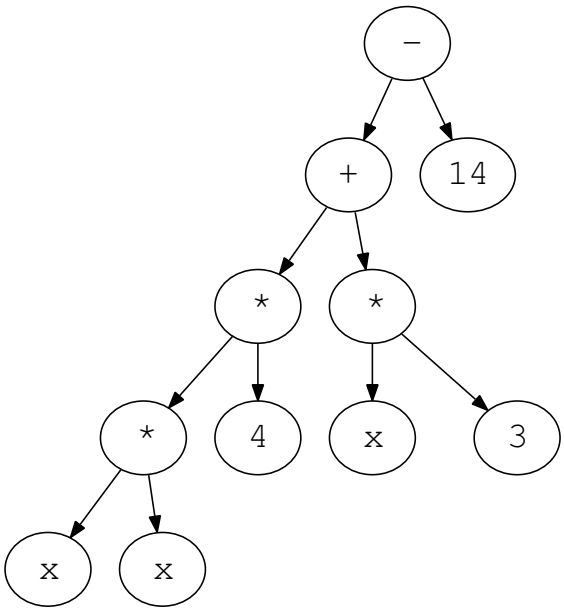

Figure 2: A tree representation of the same polynomial

.

tions, range/type of terminals, and tree depth, GP techniques are far more effective in the heuristic search for expressions that optimally describe the behavior of the variables. Particularly, *Symbolic Regression* (SR) proves to be handy to seeking models (represented as a *s-expressions*) that best fulfill the requirement of optimally describe a set of two or more variables.

SR is a GP method of *sequence induction* that

> *involves finding a mathematical expression, in symbolic form, that provides a good, best, or perfect fit between a given finite sampling of values of the independent variables and the associated values of the dependent variables. That is, symbolic regression involves finding a model that fits a given sample of data.* [28, p. 11-12].

SR has been used in a number of different applied fields such as economics, chemistry and physics to retrieve mathematical models from experimental data [34, 16, 40, 4]. Unlike other regression analysis methods used as statistical process for estimating relationships among variables (linear, quadratic, polynomial or logistic regressions, for instance) [15, 37], in symbolic regression there is no assumption about the kind of function that will model the input data: generally, only the mathematical operators and the type and range of random constants to be generated are pre-specified.

Furthermore, SR processes not necessarily need to use only usual mathematical operators as functions: in the same way that it is possible to use sum, subtraction and trigonometric operations as computational functions to create *s-expressions*, it is also possible to use custom defined functions or macros to generate compound computer algorithms written as *s-expressions*. Indeed, this flexibility is ordinarily used to avoid errors during the fitness evaluation process: the subtraction operation, for example, is usually replaced by a custom-defined "protected division" function "%", that typically yields 1 when the divisor is 0. Similarly, it is also possible to structure symbolic regression processes that, instead of dealing with individual numbers and with functions that expect their arguments to be individuals numbers, work with other data-types such as arrays or matrices, for example.

5. The OMGP library

The *OpenMusic Genetic Programming* (OMGP) is a new library for the *OpenMusic* (OM) CAM environment that implements *Symbolic Regression* by means of GP techniques. The library is based on the original proposition by John Koza [28, p. 237] and, in its current version (0.1), the library use works in three steps: (1) parametric configuration, (2) search, and (3) best function retrieve/application.

In the first stage, a series of parameters must be configured to set-up the target data-points (to be used in the fitness evaluation of each generated *s-expression* and the SR algorithm properties). This is done with the functions:

- `omgp-functions_and_terminals_vars`,
- `omgp-symbolic_regression_vars`
- `omgp-settargetsarray`.

There are also three auxiliary functions that may be used to inspect or set the parameter values:

- `omgp-getvars`,
- `omgp-setfvars`

- `omgp-mapsetfvars`

In practical situations, these functions may be useful, for example, to check specific values or to create custom configurations.

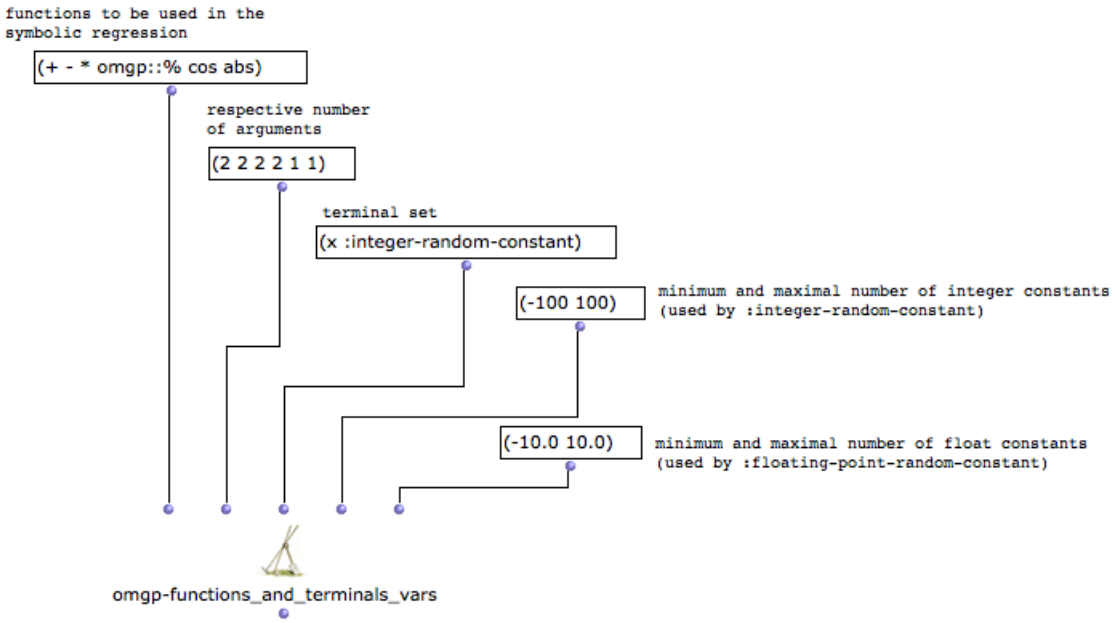

Figure 3: Function `omgp-functions_and_terminals_vars`

.

The function `omgp-functions_and_terminals_vars` is evaluated to indicate the functions repertoire or vocabulary to be used by the SR algorithm and the respective number of arguments that these functions expect to be given as input. Here the user also specifies the "terminals" of the *s-expression* tree, i.e., the kind of random constants to be generated and the independent variable symbol. Given that the user can choose between integer and floating-point numbers generators to create the constants, it is also in this function that one should specify the range of values to be used to the pseudo-random number generators. Here, it is also possible to define invariant constants (like e or π) that could potentially be used in the final expression.

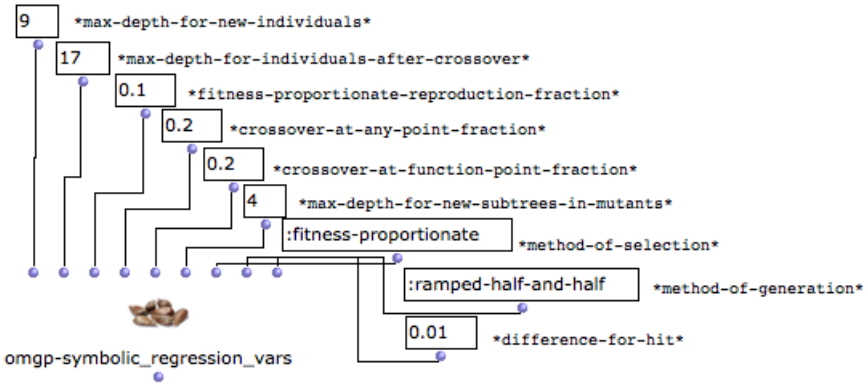

Figure 4: Function `omgp-symbolic_regression_vars`

.

The function `omgp-symbolic_regression_vars` sets parameters that are specific related to the SR algorithm and the genetic programming paradigm. For instance, one must specify the maximum depth of new individuals *s-expressions* trees, the maximum depth of individuals created through *crossover*, fractions related to the percentage

of the population that will experience *crossover* at any point or at the function points of the tree, the depth of new subtrees, the method of generation of new branches and other values.

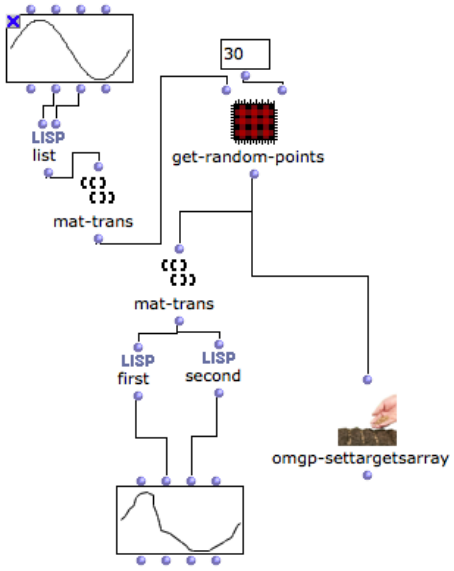

Figure 5: Getting random point of a sinusoidal function to use as target values using the function `omgp-settargetsarray`

.

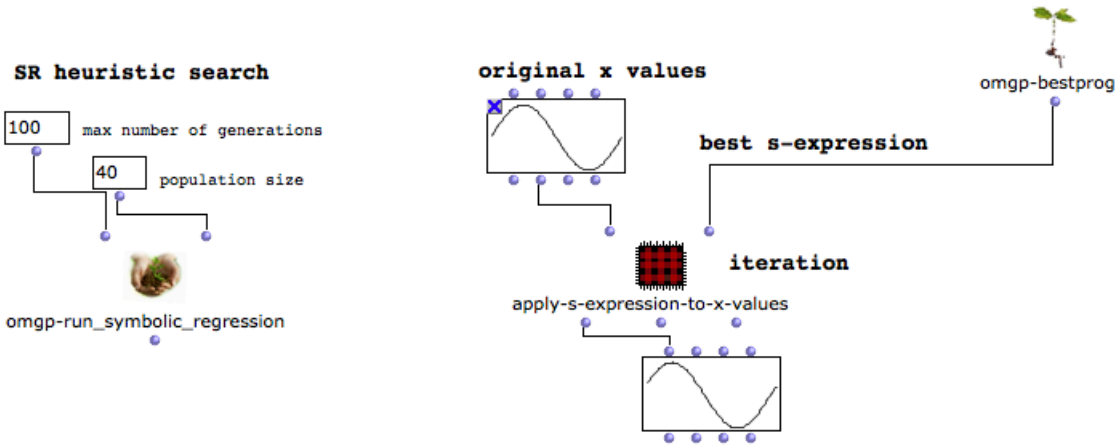

Figure 6: Functions `omgp-run_symbolic_regression` and `omgp-bestprog`, used to run the SR heuristic search and to retrieve/apply the best fitted *s-expression* to a series of independent variable values.

The function `omgp-settargetsarray` is used to store pairs of dependent and independent variables in an array that will be used to evaluate the *fitness* of a given *s-expression*. The fitness of the expression will be equal to its error to yield the expected dependent variable values when they are evaluated with the respective independent variable. Clearly, then more the *raw fitness* (sum of absolute values of the difference between obtained $f(x)$ value for all target X independent variables) is closer to 0, then more the generated *s-expression* is adapted to solve the problem.

The second step in using the OM-gp library is to run the SR algorithm by using the function `omgp-run_symbolic_regression`. This functions expects two arguments: the numbers of generations to be evaluated and the size of the initial population.

In the third step, after running the search function, one can evaluate the function `omgp-bestprog` to retrieve the CL tree *s-expression* best fitted to mathematically model the target data. This function can then be used as an algorithm — by being evaluated inside any kind of iterative process such as `mapcar` or `loop` macros. The process can then be used to generate new numerical values for any desired independent variable or be manually modified to better model specific parameters, for example.

6. Discussion

The OMGP functions described above have been applied to the regression analysis of data-sets composed by onsets and pitches of monophonic excerpts taken from the musical literature and from melodies created with the use of mathematical functions. While the proposed SR process, depending on the parameter settings, is able create functions that roughly resemble the curves identified in the original content, the first experiments gave rise to a number of issues that must be addressed before the system can be exhaustively applied and tested in practical cases of music composition and analysis with CAM tools.

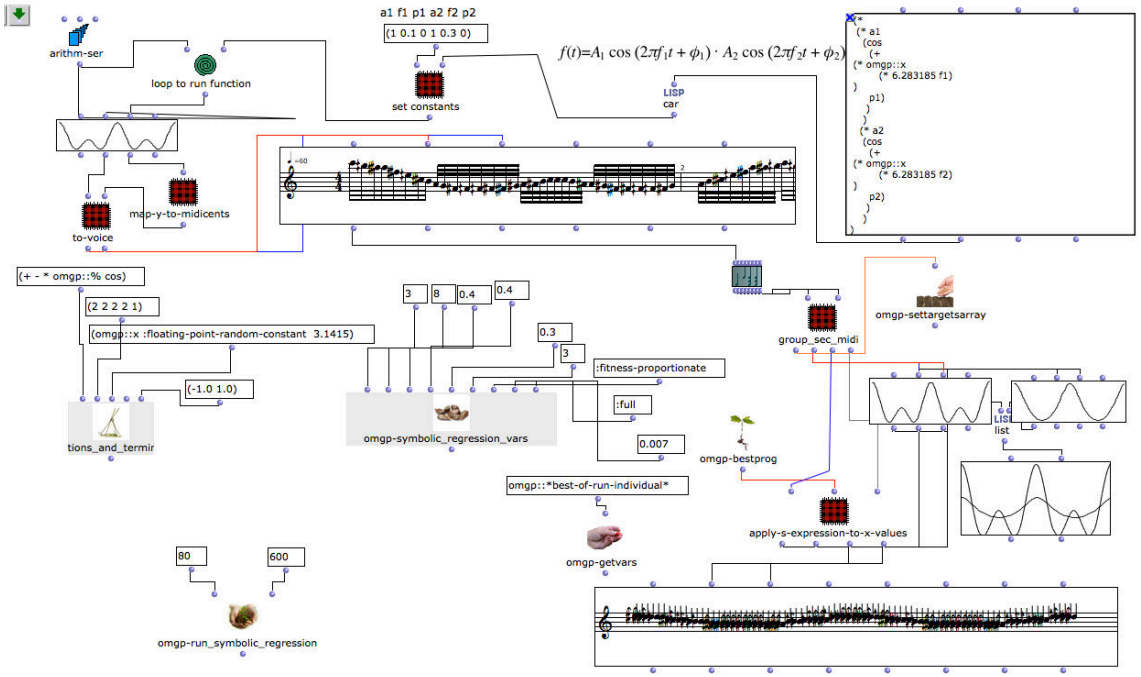

Figure 7: Experimental application of OMGP to generate an *s-expression* that models a melody generated by using a trigonometric formula. Although the resulting s-expression output resembles the input data, the library must yet be improved to generate more accurate algorithms.

.

A major issue on the current version of OMGP is related to the computation time of SR processes. Besides the evident impact of population size and of the number of generations in the computation time, the SR process is also strongly affected by the complexity and depth of trees being created, modified and evaluated as well as by the kind of fitness function being applied (in our case, *absolute mean error*). Additionally, since functions in OM are evaluated rather than being compiled [9, p. 6] and considering that long evaluations hang the environment to further computations until they are accomplished, it would be better to have an external application running the SR computations and that

could at same time be controlled and retrieve optimal results to OM through OSC commands, for example.

Additionally, there are a number of strategies that could potentially boost both the computation time and the accuracy of the results. Specially, we plan to rewrite the functions used to rune SR process to make them more compact and efficient, using specific data-structures to represent and manipulate data, as well as declared types for arguments/functions (techniques that previous works have shown to be particularly effective to make CL applications run faster [43]).

As these issues are related to different characteristics of the system, we are currently rewriting the whole library so that it works, as well, as an external application in SBCL. The server will work on input data autonomously and receive commands and retrieve data by using network OSC commands and, when necessary, other streams (such as text files, for example). As the same code that will be used in this external app may be used inside OM (as a library), the idea is to benchmark the SR system being run both in OM/LispWorks and in the independent SBCL server, adapting the code to work more accurately and fast.

7. Conclusion

We have presented a new tool to be used in CAM environments that brings together modern analysis/composition techniques and powerful heuristic and artificial intelligence instruments by applying *Symbolic Regression* and *Genetic Programming* to the problem of formalize *algorithmic models* of musical processes and procedures. In this paper we have exposed the main concept and the implementation of a new OM library that, despite being in its initial stage and demanding further development, proves to be promising to address the proposed objective. In future works we intend to apply and test this new tool to specific analytical and compositional projects, potentially expanding the features and the effectiveness of OMGP and evaluating its performance in different situations.

References

[1] Agon, C., Assayag, G., and Bresson, J., editors (2006a). *The OM composer's book 1*, volume 1 of *Musique-sciences*. IRCAM-centre Pompidou ; Delatour, Paris : [Sampzon].

[2] Agon, C., Assayag, G., and Bresson, J., editors (2006b). *The OM composer's book 2*, volume 2 of *Musique-sciences*. IRCAM-centre Pompidou ; Delatour, Paris : [Sampzon].

[3] Agustín-Aquino, O. A., Junod, J., and Mazzola, G. (2015). *Computational Counterpoint Worlds: Mathematical Theory, Software, and Experiments*. Springer.

[4] Alaa F. Sheta, Sara Elsir M. Ahmed, and Hossam Faris (2015). Evolving stock market prediction models using multi-gene symbolic regression genetic programming.

[5] Ariza, C. (2005). *An open design for computer-aided algorithmic music composition: athenaCL*. Dissertation.com, Boca Raton, Fla.

[6] Biles, J. (1994). GenJam: A Genetic Algorithm for Generating Jazz Solos. pages 131–137.

[7] Bresson, J. (2007). *La synthèse sonore en composition musicale assistée par ordinateur : modélisation et écriture du son*. phdthesis, Université Pierre et Marie Curie - Paris VI.

[8] Bresson, J., Agon, C., and Assayag, G. (2009). Visual Lisp/CLOS programming in OpenMusic. *Higher-Order and Symbolic Computation*, 22(1):81–111.

[9] Bresson, J., Agon, C., and Assayag, G. (2011). OpenMusic: Visual programming environment for music composition, analysis and research. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 743–746, New York, NY, USA. ACM.

[10] Burton, A. R. and Vladimirova, T. (1999). Generation of Musical Sequences with Genetic Techniques. *Computer Music Journal*, 23(4):59–73.

[11] Buteau, C. and Anagnostopoulou (2011). Motivic Topologies: Mathematical and Computational Modelling in Music Analysis. In *Mathematics and computation in music third international conference, MCM 2011, Paris, France, June 15-17, 2011: proceedings*. Springer, Berlin.

[12] Buxton, W. (1977). A composer's introduction to computer music. *J. of New Music Res. Journal of New Music Research*, 6(2):57–71.

[13] Carpentier, G., Tardieu, D., Assayag, G., Rodet, X., and Saint-James, E. (2007). An Evolutionary Approach to Computer-Aided Orchestration. In Giacobini, M., editor, *Applications of Evolutionary Computing*, number 4448 in Lecture Notes in Computer Science, pages 488–497. Springer Berlin Heidelberg.

[14] Charles De Paiva Santana, J. M. (2015). Towards a Borgean Musical Space: An Experimental Interface for Exploring Musical Models.

[15] Draper, N. R. and Smith, H. (1981). *Applied regression analysis*. Wiley, New York.

[16] Duffy, J. and Engle-Warnick, J. (2002). Using symbolic regression to infer strategies from experimental data. In Chen, P. S.-H., editor, *Evolutionary Computation in Economics and Finance*, number 100 in Studies in Fuzziness and Soft Computing, pages 61–82. Physica-Verlag HD.

[17] Essl, K. (2007). Algorithmic composition. In *The Cambridge companion to electronic music*, Cambridge companions to music, pages 107–125. Cambridge University Press, Cambridge ; New York.

[18] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[19] Hiller, L. and Isaacson, L. M. (1959). *Experimental music; composition with an electronic computer*. McGraw-Hill, New York.

[20] Holland, J. H. (1975). *Adaptation in natural and artificial systems an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor.

[21] Horner, A. (1996). Double-Modulator FM Matching of Instrument Tones. *Computer Music Journal*, 20(2):57–71.

[22] Horner, A., Beauchamp, J., and Haken, L. (1993). Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. *Computer Music Journal*, 17(4):17–29.

[23] Horner, A. and Goldberg, D. (1991). Genetic Algorithms and Computer-Assisted Music Composition. In *Proceedings of the 1991 International Conference on Genetic Algorithms and Their Applications*, pages 337–441.

[24] Jacob, B. L. (1995). Composing with genetic algorithms. In *Proceedings of 1995 International Computer Music Conference*, Alberta. ICMA.

[25] Jacobs, J. P. and Reggia, J. (2012). Evolving musical counterpoint: The chronopoint musical evolution system. *arXiv:1207.5560 [cs]*.

[26] Keller, D. and Ferneyhough, B. (2004). Analysis by Modeling: Xenakis's ST/10-1 080262. *Journal of New Music Research*, 33(2):161–171.

[27] Knuth, D. (1968). *The art of computer programming*. Addison-Wesley, Reading (Mass.) Menlo Park (Calif.) London etc.

[28] Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Mass.

[29] Laske, O. (1981). Composition Theory in Koenig's Project One and Project Two. *Computer Music Journal*, 5(4):54–65.

[30] Laurson, M., Kuuskankare, M., and Norilo, V. (2009). An overview of PWGL, a visual programming environment for music. *Computer Music Journal*, 33(1):19–31.

[31] Malt, M. (2000). *Les Mathématiques et la Composition Assistée par Ordinateur: concepts, outils et modèles*. Thèse de Doctorat, École des Hautes Études en Sciences Sociales, Paris.

[32] Malt, M. (2006). Concepts et modèles, de l'imaginaire à l'écriture dans la composition assistée par ordinateur. *Musique, instruments, machines*, (19):213–234.

[33] Manzolli, J., Moroni, A., Gudwin, R., and Zuben, F. v. (2000). Vox Populi: An Interactive Evolutionary System for Algorithmic Music Composition. *Leonardo Music Journal*, 10(1):49–54. Volume 10, 2000 .

[34] McKay, B., Willis, M., and Barton, G. (1997). Steady-state modelling of chemical process systems using genetic programming. *Computers & Chemical Engineering*, 21(9):981–996.

[35] Mesnage, M. and Riotte, A. (1988). Un modèle informatique d'une pièce de Stravinsky. *Analyse Musicale*, 10:51–66.

[36] Miranda, E. R. and Biles, A. (2007). *Evolutionary computer music*. Springer, London.

[37] Rawlings, J. O., Pantula, S. G., and Dickey, D. A. (1998). *Applied regression analysis: a research tool*. Springer texts in statistics. Springer, New York, 2nd ed edition.

[38] Riotte, A. (1996). Formalismes, modèles: un nouvel atout pour la composition et l'analyse. *Musurgia*, 3(3):90–105.

[39] Riotte, A. (2003). Quelques réflexions sur l'analyse formalisée. *Musurgia*, 10(1):61–71.

[40] Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science (New York, N.Y.)*, 324(5923):81–85.

[41] Spector, L. and Alpern, A. (1994). Criticism, Culture, and the Automatic Generation of Artworks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 3–8, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[42] Taube, H. (2004). *Notes from the metalevel : introduction to algorithmic music composition*. Taylor & Francis Group, London New York.

[43] Verna, D. (2006). How to make lisp go faster than c. *IAENG International Journal of Computer Science*, 32(4):499–504.

[44] Xenakis, I. (1963). *Musiques formelles: nouveaux principes formels de composition musicale*. Richard-Masse, Paris.

[45] Zils, A. and Pachet, F. (2004). Automatic Extraction of Music Descriptors from Acoustic Signals Using EDS. Audio Engineering Society.