Journal of Theoretical and Applied Information Technology
20th April 2016. Vol.86. No.2

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

GELISP: A FRAMEWORK TO REPRESENT MUSICAL CONSTRAINT SATISFACTION PROBLEMS AND SEARCH STRATEGIES

[1]MAURICIO TORO, [2]CAMILO RUEDA, [3]CARLOS AGÓN, [4]GÉRARD ASSAYAG
[1]Asstt Prof., Department of Informatics and Systems, UNIVERSIDAD EAFIT, Colombia
[2]Prof., Department of Computer Science, PONTIFICIA UNIVERSIDAD JAVERIANA CALI, Colombia
[3]Prof., Music Modeling Team, IRCAM, UMR 9912 CNRS, France
[4]Dr., Music Modeling Team, IRCAM, UMR 9912 CNRS, France

E-mail: [1]mtorobe@eafit.edu.co, [2]crueda@cic.puj.edu.co, [3]carlos.agon@ircam.fr, [4]assayag@ircam.fr,

ABSTRACT

In this article we present Gelisp, a new library to represent musical Constraint Satisfaction Problems and search strategies intuitively. Gelisp has two interfaces, a command-line one for Common Lisp and a graphical one for OpenMusic. Using Gelisp, we solved a problem of automatic music generation proposed by composer Michael Jarrell and we found solutions for the All-interval series.

Keywords: *Constraint satisfaction problems, openmusic, automatic music generation, search strategies, visual programming.*

1. INTRODUCTION

A *Constraint Satisfaction Problem* (CSP) is a formalism to represent combinatorial problems. To solve a CSP we need to find objects that satisfy a number of constraints (i.e., criteria over those variables). CSPs provide a declarative way to represent combinatorial problems, specifying cons-traints instead of a sequence of steps to find the solution (as used in imperative programming). Additionally, it is possible to specify strategies to choo-se between branches during search. CSPs in computer music can be used to solve harmonic, rhythmic or melodic problems. In addition, they can be used for automatic generation of musical structures satisfying a set of rules. For instance, we can find solutions for the *All-interval series* [6], where we need to find a sequence of 12 different pitches with 12 different intervals.

In order to solve a CSP, we can use constraint programming languages such as Prolog or Mozart-Oz [30]. In order to solve a CSP, those languages use a *Constraint Solving Library (CSL)* such as Gecode [12]. CLSs are usually written in C++.

1.1 The problem

Using traditional CSL's or programming languages to solve CSPs is time-demanding and it is intended for specialized users because they usually require deep knowledge on C++ or logic programming. This makes these tools often unpractical to specify musical CSPs. Furthermore, these tools do not provide a representation for musical data structures.

1.2 Our solution

Gelisp[1] is a wrapper for Gecode to Common Lisp. *Gelisp* was originally developed by Rueda in 2006 and we modified it to work with current version of Gecode. Furthermore, we added support to model CSPs and search strategies graphically on *OpenMusic (OM)* [1]. In addition, *Gelisp* can take advantage of the musical data structures and functions defined for OM.

The novelty of *Gelisp* is to provide a graphical representation for search strategies (e.g., Depth First Search) and global constraints (e.g., "all the intervals of a sequence must be different"), based on an efficient CSL.

1.2 Related work

Several graphical CSLs for OM have been developed in the last decade. Situation [10] generates music based on constraints, OmRc [11] finds structures corresponding to rhythmical constraints, OmClouds [29] finds approximated solutions to a CSP, and OMBacktrack

[1] http://gelisp.sourceforge.net/

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(http://www.ircam.fr/equipes/repmus/) is a wrapper for the CSL Screamer [13] (a CSL written on Lisp).

A graphical CSL to solve musical CSPs should be able to setup search strategies in a graphical way, post multiple kinds of constraints graphically without declaring explicitly loops and recursion, and solve the problem using state-of-art algorithms.

Unfortunately, OmRC and OmSituation are designed to solve specific problems. OmBacktrack is no longer available for current versions of OM. Finally, OmClouds does not guarantee a solution satisfying all the constraints (i.e., a complete solution).

2. GECODE

Gecode is a Constraint Solving Library (CSL) written in C++. Gecode provides a *propagator* for each type of constraint. Propagators translate a constraint into basic constraints supplying the same information. Basic (finite domain) constraints have the form x□[a..b]. For instance, in a store (i.e., a set with all the constraints asserted) containing and , a propagator for the constraint would add constraints and .

As described in the above example, the action of propagators ends up narrowing down the set of possible values for each variable. This, however, does not guarantee that it will eventually be inferred a single value for each variable. Gecode thus include *search engines*. The purpose of a search engine is to choose additional basic constraints to add into the store until all variables have reduced their domain to a single value. Using them we can find one, many, or all the solutions for a CSP.

Gecode works on different operating systems and it will be used as the CSL for Mozart-Oz, therefore it is very likely to be maintained for a long time. Furthermore, it provides an extensible API, allowing the user to create new propagators and user-defined search engines. For instance, we can extend Gecode to reason about trees and graphs, which are useful in musical CSPs.

3. GELISP

Gelisp provides an interface for Common Lisp and another for OM. In *Gelisp*, sequences of variables are represented by lists, as opposed to Gecode, where they are represented by arrays. This makes the power of list processing (provided by Lisp and OM) available for *Gelisp* users.

3.1 Interface for Common Lisp

To solve a problem using this interface, we need to write a script. A script is a function to define the problem variables and their domains (the possible values that a variable can take), post constraints over the variables, and setup a search strategy.

This interface allows the user to call most of Gecode propagators for both, Finite Domain (FD) and Finite Set (FS) constraints. Basic FD constraints deal with expressions of the form x□R, where R is a range or a set of ranges of integers. On the other hand, FS constraints deal with expressions among sets of FD variables. In what follows, we present some propagators that *Gelisp* provides for FD and FS.

Gelisp provides FD propagators for defining domains (e.g., *Domain*(X)=[2,5]), equalities and inequalities (e.g., X+Y<Z), cardinality (e.g., 1 occurs two times in [XYZ]), boolean constraints, regular expression constraints and the all-distinct constraint. The all-distinct constraint makes the elements of a sequence pairwise different. On the other hand, for FS we provide constraints for defining domains (e.g., V□{1,2,3}) and set relations (e.g., X□A∪ B).

In addition, *Gelisp* includes two search engines, Depth Search First (DSF) and Branch-and-bound (BAB). The DFS engine works by choosing some variable, then a value for that variable, if this does not succeed (a constraint does not hold) then chooses another value. If the value succeed, then chooses another variable, then a value for it, etc.

The BAB engine works in a similar way, but solutions are computed in such a way that each subsequent solution increases or decreases the value of some user specified FD variable. Both engines can be used for both FS and FD. In addition, we can define search heuristics for value (i.e., the order to assign a value to a variable) and variable order (i.e., the order to choose a variable). These heuristics are parameters for the search engines.

3.2 Graphical Interface for OpenMusic

Instead of writing a script, in the graphical interface we represent a program with a special patch, called *CSP patch*. A patch is a visual algorithm, in which boxes represent functional calls, and connections are functional compositions. Inside a *CSP patch*, we can place special boxes to define a constraint in the CSP, variable and value heuristics, the variable to be optimized during the search, and a time limit in the search.

For instance, we provide a variety of boxes to represent simple constraints (e.g., a=2) and global

Journal of Theoretical and Applied Information Technology
20th April 2016. Vol.86. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

constraints (e.g., "all the intervals from a sequence must be different").

Using the graphical interface we can express a variety of problems declaratively with global constraints. Global constraints have parameters. For instance, the graphical box to find the intervals of a list has a parameter to choose among absolute, non-absolute, or modulo *n* intervals (calculated as). Additionally, it has a parameter to post an *all-distinct* constraint over the intervals.

Moreover, the output of a CSP patch can be connected to a box to find one solution or a box to find *n* the solutions

4. CASE STUDIES

In this section, we describe both, an intuitive and formal definition of two CSPs and we explain how to solve them with *Gelisp*. Formally, a CSP is triple <*X*,*D*,*C*>, where *X* is a set of variables, *D* is the domain for each variable, and C is a set of constraints (read as conjunction) over the variables.

4.1 All-interval series

In this problem, we need to find a sequence of 12 different pitches with 12 different intervals (fig. **1**). This problem can be generalized to find *n* different pitches with *n* different intervals equivalent under inversion [2]. For instance, a value of *n*=24 represents the *all-interval series* for microtones.
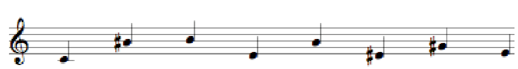
Figure 1: An all-interval serie for *n*=12

Therefore, a solution to this CSP is a sequence of *n* pairwise different variables with domain [1..*n*], where all modulo *n* intervals of the sequence are pairwise different. We give bellow a formalization of this problem

Variables: $V_1 \ldots V_n$
Domains: [1..*n*] ... [1..*n*]
Constraints:

- alldiff(*V*)
- alldiff $((V_{i+1} - V_i)\%n, i <= n-1)$

There is not a constraint over the interval) because that interval is always six, according to the literature. Furthermore, it is enough to calculate the series where because the other ones can be obtained from that one using transposition. In addition, we know that if is an all-interval serie, is also one. For those reasons, we include these two constraints to avoid symmetrical solutions:

- C3 $V_0 = 0$
- C4 $V_0 < V_n$

We represent graphically this CSP (fig. **2**) with a box to create *n* all-different variables with domain [1..*n*], an box for with an all-different parameter , an *equality* box for , and an *inequality* box for .

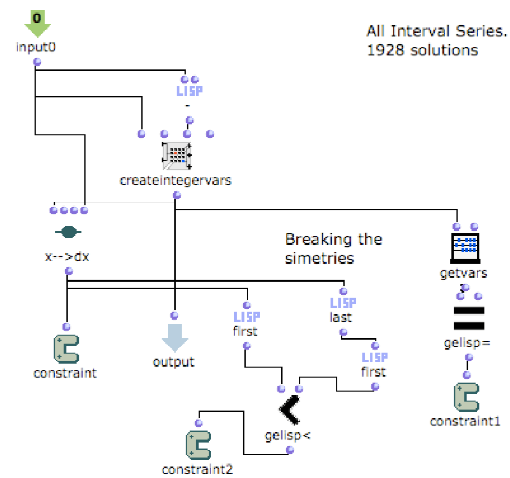

Figure 2: All-interval Series CSP on OM

4.2 Jarrell's CSP

Composer Michael Jarrell proposed an idea for automatic music generation [4]. The goal is to generate a sequence of *n* notes. There is a fix number of occurrences ... for each sequences of intervals (called motives) ... over the sequence of non-absolute intervals of the output sequence. In addition, each note of the output sequence belongs to a Chord *Ch*. Moreover, the first and the last note of the output sequence are fixed. We give bellow a formalization of this problem

Inputs:

- Motives [...], Limits and ,

  Occurrences [...], Chord *Ch*

Variables: ...
Domains: [0..127] ... [0..127]
Constraints:

[2] For instance, an interval C-E is equivalent to E-C.

Journal of Theoretical and Applied Information Technology
20th April 2016. Vol.86. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- C_1 $\forall_{1<i<A}$ $|\{j, M_i$ is a subsequence of the variables' intervals that starts on $j\}| = OM_i$
- C_2 $\forall_{1<i<n}$ $V_i \in Ch$
- C_3 $V_1 = L_1 \wedge V_n = L_2$

We represent graphically (Fig.**3**) the constraint . We use the *x→dx* and *motives-occurs=* boxes to fix the number of occurrences of each motive over the intervals of the output sequence.

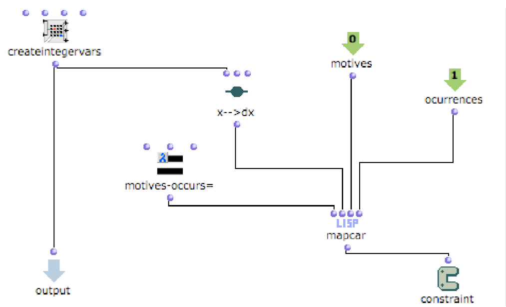

Figure 3: Constraint for Jarrell's CSP on OM

Jarrell also proposes in [4] to consider absolute intervals and octaviation for the chords, the limits and the motives. For instance, using absolute intervals, an interval is equal to and using octaviation, a pitch G4 is equivalent to G1,G2,G5, etc. Finally, he also proposes to have specific motives and chords for each segment of the output sequence, according to a user-defined segmentation. For simplicity, we do not present those constraints in this paper. However, a complete model of this problem can be found at *Gelisp* website.

REFERENCES:

[1] C. Agón, G. Assayag, O. Delerue, and C. Rueda. Objects, Time and Constraints in OpenMusic. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, Michigan, October 1998.
[2] A. Allombert, M. Desainte-Catherine, and M. Toro. Modeling temporal constrains for a system of interactive score. In G. Assayag and C. Truchet, editors, *Constraint Programming in Music*, chapter 1, pages 1–23. Wiley, 2011.
[3] J. Aranda, G. Assayag, C. Olarte, J. A. Pérez, C. Rueda, M. Toro, and F. D. Valencia. An overview of FORCES: an INRIA project on declarative formalisms for emergent systems. In P. M. Hill and D. S. Warren, editors, *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *Lecture Notes in Computer Science*, pages 509–513. Springer, 2009.
[4] F. Courtot and M. Jarrell. *L'utilisation de la CAO dans Congruences*. Cahiers d'exploitation Ircam, 1990.
[5] F. Fages and J. Martin. From rules to constraint programs with the rules2cp modelling language. 13th Worshop on Constraint Solving and Constraint Logic Programming, CSCLP'08, 2008.
[6] R. Morris and D. Starr. The structure of the all-interval series. *Journal of Music Theory*, 2(13), 1974.
[7] C. Olarte, C. Rueda, G. Sarria, M. Toro, and F. Valencia. Concurrent Constraints Models of Music Interaction. In G. Assayag and C. Truchet, editors, *Constraint Programming in Music*, chapter 6, pages 133–153. Wiley, Hoboken, NJ, USA., 2011.
[8] A. Philippou and M. Toro. Process Ordering in a Process Calculus for Spatially-Explicit Ecological Models. In *Proceedings of MOKMASD'13*, LNCS 8368, pages 345–361. Springer, 2013.
[9] A. Philippou, M. Toro, and M. Antonaki. Simulation and Verification for a Process Calculus for Spatially-Explicit Ecological Models. *Scientific Annals of Computer Science*, 23(1):119–167, 2013.
[10] C. Rueda, M. Lindberg, M. Laurson, G. Block, and G. Assayag. Integrating constraint programming in visual musical composition languages. In *ECAI 98 Workshop on Constraints for Artistic Applications, Brighton*, 1998.
[11] O. Sandred. Searching for a rhythmical language. In *PRISMA 01 Review. EuresisEdizioni, Milano*, 2003.
[12] C. Schulte and P. J. Stuckey. Efficient constraint propagation engines. *CoRR*, abs/cs/0611009, 2006.
[13] J. M. Siskind and D. A. Mcallester. Nondeterministic lisp as a substrate for constraint logic programming. In *proceedings of AAAI-93*, pages 133–138. AAAI Press, 1993.
[14] M. Toro. Probabilistic Extension to the Factor Oracle Model for Music Improvisation. Master's thesis, Pontificia Universidad Javeriana Cali, Colombia, 2009.
[15] M. Toro. Towards a correct and efficient implementation of simulation and verification

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

tools for probabilistic ntcc. Technical report, Pontificia Universidad Javeriana, May 2009.

[16] M. Toro. Structured interactive musical scores. In M. V. Hermenegildo and T. Schaub, editors, *Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010, July 16-19, 2010, Edinburgh, Scotland, UK*, volume 7 of *LIPIcs*, pages 300–302. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[17] M. Toro. *Structured Interactive Scores: From a simple structural description of a multimedia scenario to a real-time capable implementation with formal semantics* . PhD thesis, Univeristé de Bordeaux 1, France, 2012.

[18] M. Toro. Structured interactive music scores. *CoRR*, abs/1508.05559, 2015.

[19] M. Toro. Probabilistic Extension to the Concurrent Constraint Factor Oracle Model for Music Improvisation. *ArXiv e-prints*, Feb. 2016.

[20] M. Toro and M. Desainte-Catherine. Concurrent constraint conditional branching interactive scores. In *Proc. of SMC '10*, Barcelona, Spain, 2010.

[21] M. Toro, M. Desainte-Catherine, and P. Baltazar. A model for interactive scores with temporal constraints and conditional branching. In *Proc. of Journées d'Informatique Musical (JIM) '10*, May 2010.

[22] M. Toro, M. Desainte-Catherine, and J. Castet. An extension of interactive scores for multimedia scenarios with temporal relations for micro and macro controls. In *Proc. of Sound and Music Computing (SMC) '12*, Copenhagen, Denmark, July 2012.

[23] M. Toro, M. Desainte-Catherine, and C. Rueda. Formal semantics for interactive music scores: a framework to design, specify properties and execute interactive scenarios. *Journal of Mathematics and Music*, 8(1):93–112, 2014.

[24] M. Toro, A. Philippou, S. Arboleda, C. Vélez, and M. Puerta. Mean-field semantics for a Process Calculus for Spatially-Explicit Ecological Models. Technical report, Department of Informatics and Systems, Universidad Eafit, 2015. Available at http://blogs.eafit.edu.co/giditic-software/2015/10/01/mean-field/.X

[25] M. Toro, A. Philippou, C. Kassara, and S. Sfenthourakis. Synchronous parallel composition in a process calculus for ecological models. In G. Ciobanu and D. Méry, editors, *Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing - ICTAC 2014, Bucharest, Romania, September 17-19*, volume 8687 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2014.

[26] M.Toro, C.Rueda, C.Agón, and G.Assayag. Ntccrt: A concurrent constraint framework for soft real-time music interaction. *Journal of Theoretical & Applied Information Technology*, 82(1), 2015.

[27] M. Toro-Bermúdez. ntccrt: A ntcc generic real-time interpreter. more information available at https://sourceforge.net/projects/ntccrt . Internet, 2008.

[28] M. Toro-Bermúdez. Towards non-threaded concurrent constraint programming for implementing multimedia interaction systems, July 2008.

[29] C. Truchet, G. Assayag, and P. Codognet. Omclouds, a heuristic solver for musical constraints. In *MIC03, Metaheuristics International Conference*, Kyoto, 2003.

[30] P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, Mar. 2004.