A constraint-based melody harmonizer

Rafael Ramirez[1] and Julio Peralta[2]

[1] National University of Singapore
Information Systems and Computer Science Department
Lower Kent Ridge Road, Singapore 119260
rafael@iscs.nus.sg
[2] University of Bristol, Computer Science Department
MVB, Woodland Road, Bristol, UK
jperalta@cs.bris.ac.uk

Abstract. The aim of this work is to automatically generate a sequence of chords that satisfactorily harmonize a given melody. We concentrate in a very special task: the automatic harmonization of single-key melodies using triads, i.e. three-note chords consisting of any note of the scale with the third above it and the fifth above it. The algorithm is composed by two steps: in the first step, a whole range of chord sequences is generated, each of which is a 'legal' harmonization of the melody. The second step refines the solution set by constraining the chord sequences to follow standard chord patterns. It turns out that the task of intelligent harmonization of musical melodies can be elegantly described as a constraint satisfaction problem which can be efficiently implemented with traditional constraint logic programming languages.

1 Introduction

Musicians are very often skeptical about computer-generated music arguing that musical activity is, before all, a creative process and as such, it is extremely difficult (if at all possible) to be rationalized. We share this point of view and thus, we concentrate in a very special task: the automatic generation of a sequence of chords that satisfactorily harmonize the melody. In particular, we are interested in the automatic harmonization of single-key melodies using triads, i.e. three-note chords consisting of any note of the scale with the third above it and the fifth above it. The algorithm is composed of two steps: in the first step, a whole range of chord sequences is generated, each of which is a 'legal' harmonization of the melody. The second step refines the solution set by constraining the chord sequences to follow standard chord patterns.

It turns out that the the task of intelligent harmonization of music melodies can be elegantly described as a constraint satisfaction problem which can be efficiently solved with traditional constraint logic programming systems.

The rest of this paper is organized as follows: Section 2 describes related work. Section 3 describes our system. Section 4 identifies the task of harmonizing a musical melody as an instance of the constraint satisfaction problem. Section 5 introduces constraint programming as a framework for the implementation of our system and finally, Section 6 reports on the current status of the system and mention some areas of future research.

2 Related work

The relationship between computers and music goes back as far as the 1950's when sound synthesis programs were being developed at AT&T [5]. Until the mid-eighties, much of the attention was focused on timbres rather than the structure of music (e.g. [6], [4], [8]). Today, with increasingly more powerful computers, samplers and sample-based synthesizers, more attention can be placed on the music structure. There are many examples of systems producing 'correct' music, most of them based on rules or databases containing style descriptions. An interesting system is described in [3]. The system automatically derives a four voice score from a given melody, i.e. it composes the bass and middle voices, in an object-oriented framework. The composition task is represented as a constraint system. The algorithm consists in two steps: in the first one, constraints which represent relations of intervals between notes of a single voice are applied, while the second step states constraints which represent relations between notes of the four voices in a single chord. The expert system CHORAL [1] is a system for harmonizing chorals in the style of J.S. Bach. The input is a basic soprano melody of a choral. The system generates a four voice score according to the harmonic rules of the 17th and 18th century. The system uses a rule-based heuristic search with backtracking. The knowledge base contains 350 rules, which either represent conditions that must be strictly obeyed, or which represent conditions that do not have to be strictly satisfied (and support composition decisions). A simulation of improvisation in Jazz is described in [2]. The system proposed takes as input a chord progression to which a melody is automatically improvised. Initially, the algorithm generates a certain contour which represents the main shape of the melody and is derived by a regular grammar. In a second stage, the system chooses the concrete pitches using constraints.

3 System overview

The system described here is an algorithmic harmonizer system that attempts to model the creativity involved in harmonizing and arranging a piece of popular music. The system was designed to reproduce very closely the creative process that the authors use when harmonizing melodies in the guitar, although the process is independent of the instrument in which the harmonization is based.

The process of harmonization is difficult to formalise and the way a given melody is harmonized varies from person to person according to his/her taste and background. Thus, the system here presented makes no attempts at providing the user with a definite answer. It should be thought of as a tool which produces a variety of 'correct' harmonizations that the user can choose from. We will assume that the melodies provided as input for the system are in the key of C. Note that this assumption does not make the system less general since all melodies are isomorphic to a melody in the key of C.

The system, represented in Figure 1, is composed of two parts: the *chord sequence generator* and the *filter*.

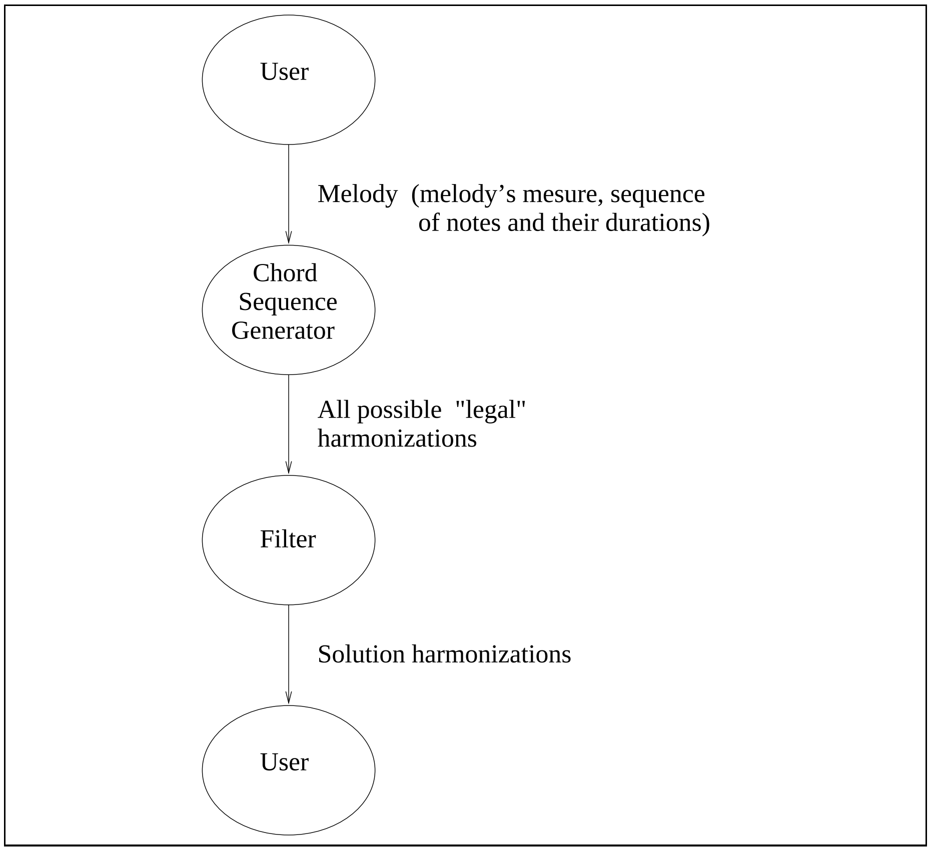
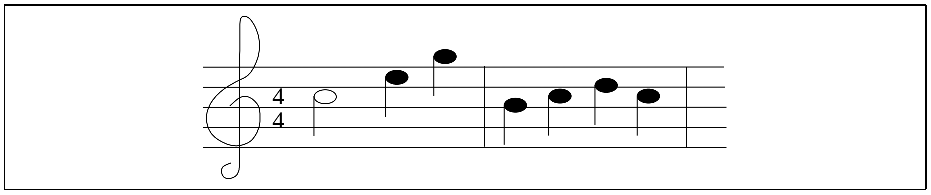

User
Melody (melody's mesure, sequence of notes and their durations)
Chord Sequence Generator
All possible "legal" harmonizations
Filter
Solution harmonizations
User

Fig. 1. System overview

The chord sequence generator produces all possible chord sequences which are not in conflict with the notes of the melody. This is, for each bar, the system generates chords which harmonize the main notes in the melody. At this stage, the criteria used for generating such chords consists of identifying the main notes in the melody and for each note, producing the triads which contain the note.

For instance, given the following melody fragment



the chord generator produces chords C, Am and F for the first note and chords Em, C, Am for the second note. In general, given a melody, the chord generator produces a sequence of sets of the form

$$\{X_1, Y_1, Z_1\}, \{X_2, Y_2, Z_2\}, \ldots \{X_n, Y_n, Z_n\}$$

which represents a set of 3^n possible harmonizations. X_i, Y_i and Z_i $(1 \leq i \leq n)$ are triads harmonizing the i-th main note in the melody.

Once this set of possible harmonizations has been generated, the filter progressively refines the solution set by constraining the chord sequences to follow standard chord patterns. This is done by representing the solution chord for each note in the melody by a variable and imposing constraints on the relative values of these variables. The constraints express popular music chord patterns such as

$$I,\ IV,\ V,\ I$$

where I, IV and V denote the first, fourth and fifth grades of the scale, respectively, e.g. in the key of C, they denote C, F and G chords.

Example. Consider the following melody:

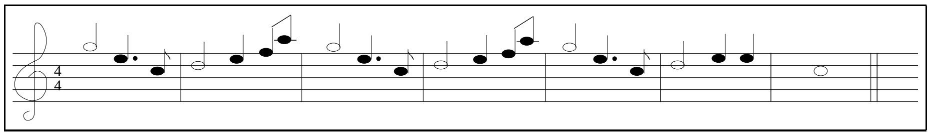


Fig. 2. Sample melody

Abstracting the relative height of the notes in the melody, we encode every note as a pair (N_i, D_i) where each N_i represents the note, e.g. C, $F\#$, and D_i is the duration associated with N_i. The resulting sequence is

$$(G,2),(E,1.5),(C,0.5),(D,2),(E,1),(F,0.5),(A,0.5),$$

$$(G,2),(E,1.5),(C,0.5),(D,2),(E,1),(F,0.5),(A,0.5),$$

$$(G,2),(E,1.5),(C,0.5),(D,2),(E,1),(E,1),(C,4)$$

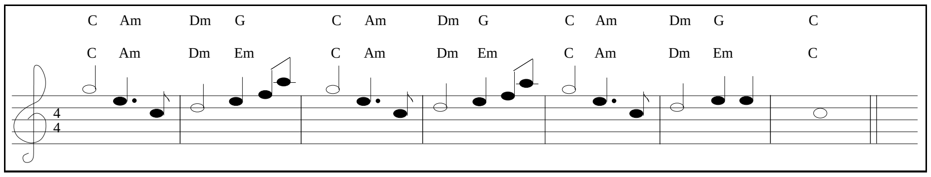
Further processing of the melody identifies the sequence of variables associated with the main notes as

$$V_G^1, V_E^1, V_D^1, V_E^2, V_G^2, V_E^3, V_D^2, V_E^4, V_G^3, V_E^5, V_D^3, V_E^6, V_C^1$$

Domains for the above variables, regardless of their subscripts, are

$$\begin{aligned} V_G &= \{C, Em, G\} \\ V_E &= \{Am, C, Em\} \\ V_D &= \{G, Bdis, Dm\} \\ V_C &= \{F, Am, C\} \end{aligned}$$

Given the chord pattern $\{I, IV, II, III, I\}$ and the use of substitution chords we obtain two possible harmonizations for the melody as shown in Figure 3.



C Am Dm G C Am Dm G C Am Dm G C
C Am Dm Em C Am Dm Em C Am Dm Em C

Fig. 3. Harmonizations

4 Harmonizing as an instance of the constraint satisfaction problem

The theory of constraint satisfaction problem [7] is an attractive framework in which to formalize the harmonizing task described above. In its general form, a constraint satisfaction problem consists of a finite set of n variables

$$S = \{V_1, \ldots V_n\}$$

where each variable V_i $(1 \leq i \leq n)$ has a domain $D_i = \{V_{i1}, \ldots V_{ik}\}$ associated with it, and consisting of k values. Related to these variables is a set of constraints $C = \{C_1, \ldots C_m\}$ each specified over a subset of the n variables in S. These constraints limit the possible combinations of values that the variables in that subset can take. To solve the problem it is necessary to find an assignment of values to the variables which satisfies all the constraints simultaneously.

A large number of problems in artificial intelligence, operations research and symbolic logic can be viewed as special cases of the general constraint satisfaction problem. Examples of its use can be found in machine vision, belief maintenance, scheduling, planning, database consistency checking, temporal reasoning, etc. The task of harmonizing a musical melody can be seen as an instance of the constraint satisfaction problem:

- There are n variables, where n is the number of main notes identified in the melody.
- The domain of these variables is given by the sequence of sets of chords produced by the chord generator as described in the previous section.
- The standard chord patterns can be formulated as constraints ruling two or more variables.

Operations research approaches are not suitable for accounting the variety of constraints that our problem involves. Thus, we use *constraint programming* which we briefly describe in the next section.

5 Constraint programming

The aim in constraint programming is to progressively restrict the set of possible values for a set of variables using some given constraints until finally, a unique value has been found for each variable.

The set of possible values for each variable is kept in the *constraint store*. For instance, the initial constraint restricting the possible chords for the note C in the melody is specified by placing the constraint $T_1 \in \{C, Am, F\}$ in the constraint store, where T_1 is the variable whose final value is the triad harmonizing the

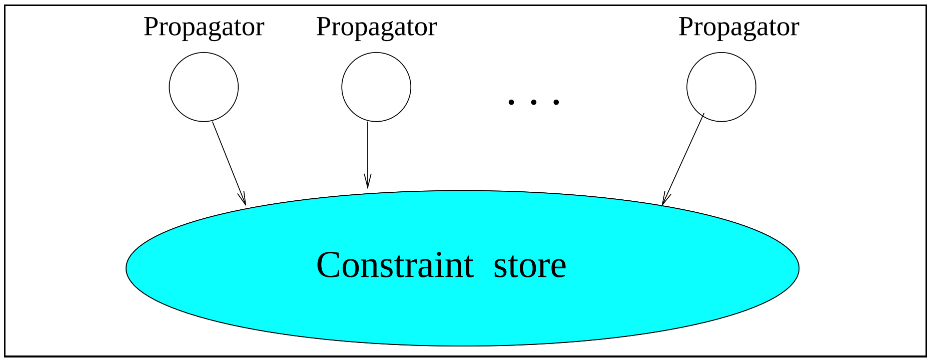


Fig. 4. Propagators and constraint store

initial part of the melody. More complex constraints are expressed by *propagators* that observe the constraint store and modify it if possible (see Figure 4).

A propagator inspects the constraint store with respect to a fixed set of variables. When more information about a particular variable is available in the store it may add further information on other variables to the store, i.e. it modifies the store by adding more constraints to it. For example, suppose T_i, T_{i+1}, T_{i+2} and T_{i+3} are four variables representing four consecutive chords in the harmonization which are constrained to $\{C, Em\}, \{Am, F, Dm\}, \{G, Em\}$ and $\{C, Am, F\}$ respectively, and we have a unique propagator implementing the common chord sequence C, F, G, C. If T_i is further constrained to $\{C\}$, the propagator will reduce the domains of T_{i+1}, T_{i+2} and T_{i+3} to $\{F\}, \{G\}$ and $\{C\}$, respectively.

Propagators remain active, waiting for more information to be added to the store. A propagator only ceases to exist when it becomes clear that it will never modify the store again. Propagators are viewed as concurrent entities that observe the constraint store and modify it whenever possible.

Constraint propagation typically does not suffice to determine the values for all the variables of the constraint propagation problem. Thus, after exhaustive propagation, a variable is speculatively constrained to one of its remaining values. This decision typically enables some propagators to further constrain other variables. In this way, the search space is continuously pruned while it is being explored.

6 Current status

We have a prototype implementation of the system described in this paper, which takes a melody input in the form of a list of pairs $[(N_1, D_1), (N_2, D_2) \ldots (N_k, D_k)]$

where each N_i $(1 \leq i \leq k)$ represents a note, e.g. C, D, $F\#$, and D_i is the duration associated with the note N_i for all i $(1 \leq i \leq k)$. The system allows the user to visualize the search tree as it is searched to find the harmonization solutions. The user can interactively listen to a solution by clicking on the solution tree nodes.

Further work includes the implementation of a good user interface for the input of the melodies to the system and the generation of MIDI output. An important research direction is to consider more complex chords with more than three notes.

References

1. Ebcioglu, K. 1992. *An expert system for harmonizing chorales in the style of J.S. Bach.* In Balaban, M. et al, editors, Understanding Music with AI: Perspectives on Music Cognition, The AAAI Press, pp. 3-28. The MIT Press, Cambridge, Menlo Park, London.
2. Johnson-Laird, P. 1991. *Jazz improvisation: a theory at the computation level.* In Howell, P. et al editors, Representing Musical Structure, pp. 291-325. Academic Press, London.
3. Pachet, F. and Roy, P. 1995. *Mixing constraints and objects: a case study in automatic harmonization.* In Proceedings of TOOLS Europe, Versailles.
4. Risset, J.C. 1985. *Computer music experiments 1964-....* Computer Music Journal, 9(1).
5. Roads, C. 1989. *An interview with Max Mathews.* In Roads, C., editors, The Music Machine. MIT Press, Cambridge MA.
6. Truax, B. 1982. *Timbral construction in Arras as a stochastic process.* Computer Music Journal, 6(3).
7. Tsang, E. 1993. *Foundations of constraint satisfaction.* Computation in Cognitive Science. academic Press, San Diego CA.
8. Vaggione, A. 1982. *The making of Octuor.* Computer Music Journal, 8(2).