Learning Counterpoint Rules for Analysis and Generation

EDUARDO MORALES M.
ITESM - Campus Morelos, Apto. Postal C-99,
Cuernavaca, Morelos, 62050, México
email: emorales@rs970.mor.itesm.mx

ROBERTO MORALES-MANZANARES
Laboratorio de Informática Musical (LIM)
Escuela de Música de la Universidad de Guanajuato
Universidad de Guanajuato
Centro de Investigaciones en Matemáticas (CIMAT)
Paseo de la Presa # 152
Guanajuato, Gto., México
email: roberto@kaliman.cimat.conacyt.mx

Abstract
History in composition and analysis have shown that composers using the same patterns in structure and harmony get different results depending on the way these patterns are resolved. In terms of musical analysis, a particular piece can be described by a sequence of states and transitions between states representing the personal criteria that each composer pursues when solving a musical structure. A first-order learning system, called Pal, is used to learn transition criteria for counterpoint analysis, in the form of Horn clauses from pairs of musical states (given as sets of notes) and general purpose musical knowledge. It is shown how the rules learned by Pal can be used for musical analysis of simple two-voice counterpoint pieces. Similarly, a counterpoint voice can be generated from a single voice (cantus firmus) using the learned rules. Conclusions and future research directions are given.

1 Introduction

Musical composition generates symbolic representations (i.e., musical scores) of musical ideas. Such ideas are based on subjective temporal interpretations of auditive events. The events are characterized by their frequency, amplitude and its envelope (which determines the quality of tone or pitch). Such elements, which define the musical characteristics of the musical instruments, are part of the material which a composer uses to propose its aesthetic solutions. During this process, a composer can follow a set of implicit or explicit rules to guide his/her preferences and express his/her ideas. Our goal is to induce musical criteria rules that can be used for musical analysis and generation. As a first step, we looked at counterpoint analysis, which is well understood and defined with a finite set of known rules (Fux, 1971). Counterpoint rules can be expressed in a compact and understandable way using first-order logic. In general, musical rules express relations between notes. In order to learn such rules we used Pal (Morales, 1991; Morales, 1992a) an Inductive Logic Programming (ILP) system (Muggleton, 1991) capable of learning a subset of Horn clauses from examples and background knowledge expressed as logic programs. It is shown how Pal can learn counterpoint rules of the first specie (to be defined below) and used them for musical analysis and generation of counterpoint notes from *cantus firmus* (a sequence of single notes to which counterpoint rules are applied to generate harmonic notes). The constraints used by Pal to guide its inductive search for hypothesis are discused, in the context of music and in general to other areas where Pal has been used.

Section 2 describes some musical concepts required for counterpoint analysis. Section 3 provides some definitions from logic. The concepts and notation will be used in the sections to follow. Section 4 briefly describes an ILP inductive framework and Pal. Section 5 shows the main results and finally conclusions and future work are given in section 6.

2 Musical background

The concept of musical counterpoint emerge in the 14th. century and evolve up to *Gradus ad Parnassum* by Johann Joseph Fux published in 1725 (Fux, 1971). This is the first book which synthesize in form of rules the art of polyphony considered to be correct by that time. Those rules can be considered as the culmination of musical analysis from the 14th. until 18th. century.

Counterpoint rules can be classified between two voices (sequences of notes) into several species, according to the number of notes involved at the same time on each voice:

- 1st: one note on one voice against one note on the other
- 2nd: two notes on one voice against one note on the other
- 3rd: four notes on one voice against one note on the other
- 4th: a whole note (of four times) on one voice against half notes (of two times) on the other
- 5th or florid: three or more notes in combination with the previous species

Our goal is to obtain similar rules as those described by Fux from examples of musical pieces and basic musical knowledge from traditional music. Such knowledge includes the classification of intervals (distances in height between two notes) into: *consonances* and *dissonances*. *Unison*, *fifth* and *octave* are *perfect consonances* while *third* (mayor and minor) and *sixth* (mayor and minor) are *imperfect consonance*. *Second* (mayor and minor), *fourth*, *augmented fourth*, *diminished fifth* and *seventh* (mayor and minor) are *dissonances*.

These are the elements which account for all harmony in music. The purpose of harmony is to give pleasure by variety of sounds through progressions from one interval to another. Progression is achieved by motion, denoting the distance covered in passing from one interval to another in either direction, up or down. This can occur in three ways: direct, contrary and oblique:

- *direct motion*: results when two or more parts ascend or descend in the same direction
- *contrary motion*: results when one part ascends and the other descends, or vice versa.
- *oblique motion*: results when one part moves while the other remains stationary

With these concepts the counterpoint rules of 1st. specie are defined as follows:

First rule: from one perfect consonance to perfect consonance one must proceed in contrary or oblique motion

Second rule: from a perfect consonance to an imperfect consonance one may proceed in any of the three motions

Third rule: from an imperfect consonance to a perfect consonance one must proceed in contrary or oblique motion

Fourth rule: from one imperfect consonance to another imperfect consonance one may proceed in any of the three motions

In section 4 it is shown how these rules are learned from a small set of examples, represented as pairs of notes in two voices, and general musical knowledge about musical intervals. First we give a short description of Pal.

3 Preliminaries

A *variable* is represented by a string of letters and digits starting with an upper case letter. A *function symbol* is a lower case letter followed by a string of letters and digits. A *predicate symbol* is a lower case letter followed by a string of letters and digits. A *term* is a constant, variable or the application of a function symbol to the appropriate number of terms. An *atom* or *atomic formula* is the application of a predicate symbol to the appropriate number of terms. A *literal* is an atom or the negation of an atom. Two literals are *compatible* if they have the same symbol, name and number of arguments. The negation symbol is $\neg$. A *clause* is a disjunction of a finite set of literals, which can be represented as $\{A_1, A_2, \ldots, A_n, \neg B_1, \ldots, \neg B_m\}$. The following notation is equivalent:

$$A_1, A_2, \ldots, A_n \leftarrow B_1, B_2, \ldots, B_m.$$

A *Horn clause* is a clause with at most one positive literal (e.g., $H \leftarrow B_1, \ldots, B_m$). The positive literal (H) is called the *head*, the negative literals (all B_is) the *body*. A clause with empty body is a *unit clause*. A set of Horn clauses is a *logic program*. F_1 *syntactically entails* F_2 (or $F_1 \vdash F_2$) iff F_2 can be derived from F_1 using the deductive inference rules. A *substitution* $\Theta = \{V_1/t_1, V_2/t_2, \ldots, V_n/t_n\}$ consists of a finite sequence of distinct variables paired with terms. An *instance* of a clause C with substitution Θ, represented by $C\Theta$, is obtained by simultaneously replacing each occurrence of a component variable of Θ in C by its corresponding term. A *model* of a logic program is an interpretation for which the clauses express true statements. We say that F_1 *semantically entails* F_2 (or $F_1 \models F_2$, also F_1 logically implies or entails F_2, or F_2 is a logical consequence of F_1), iff every model of F_1 is a model of F_2.

4 Pal

Inductive Logic Programming (ILP) is a fast growing research area which combines Logic Programming and Machine Learning (Muggleton, 1991). The general setting for ILP is, given a background knowledge $\mathcal{K}$ (in the form of first-order clauses) and sets of positive ($\mathcal{E}^+$) and negative ($\mathcal{E}^-$) examples, find a hypothesis $\mathcal{H}$ (another set of clauses) for which $\mathcal{K} \wedge \mathcal{H} \vdash \mathcal{E}^+$ and $\mathcal{K} \wedge \mathcal{H} \nvdash \mathcal{E}^-$. That is, find a hypothesis which can explain the data in the sense that all the positive ($\mathcal{E}^+$) but none of the negative ($\mathcal{E}^-$) examples can be deduced from the hypothesis and the background knowledge. This inductive process can be seen as a search for logic programs over the hypothesis space and several constraints have been imposed to limit this space and guide the search. For learning to take place efficiently, it is often crucial to structure the hypothesis space. This can be done with a model of generalization. Searching for hypothesis can then be seen as searching for more general clauses given a known specialized clause.

Plotkin (Plotkin, 1969; Plotkin 1971a; Plotkin, 1971b) was the first to study in a rigorous manner the notion of generalization based on Θ-subsumption. Clause $C\Theta$-subsumes clause D iff there exists a substitution σ such that $C\sigma \subseteq D$. Clause C_1 is more general than clause C_2 if C_1 Θ-subsumes C_2. Plotkin investigated the existence and properties of least general generalizations or *lgg* between clauses and the *lgg* of clauses relative to some background knowledge or *rlgg*. That is, generalizations which are less general, in terms of Θ-subsumption, than any other generalization.

More recently, Buntine (Buntine, 1988) defined a model-theoretic characterization of Θ-subsumption, called *generalized subsumption* for Horn clauses (see (Buntine, 1988) for more details). Buntine also suggested a method for constructing *rlgg*s using Plotkin's *lgg* algorithm between clauses. The general idea of the *rlgg* algorithm is to augment the body of the example clauses with facts derived from the background knowledge definitions and the current body of the example clauses, and then generalized these "saturated" clauses using the *lgg* algorithm. Pal's learning algorithm is based on this framework which is more formally described in Table 1.

A direct implementation of it is impractical for all but the simplest cases, as it essentially involves the deduction of all ground atoms logically implied by the theory (see (Niblett, 1988) for a more thorough discussion on generalization). However, *rlgg* exists for theories without variables (as in Golem (Muggleton & Feng, 1990)), theories without function symbols (as in Clint (deRaedt & Bruynooghe, 1988)), and when only a finite number of facts are deducible from the theory, either by limiting the depth of the resolution steps taken to derive facts and/or by constraining the theory, as in Pal. Even with a finite set of facts, the *lgg* of two clauses can generate a very large number of literals and some additional constraints are required

Table 1: A plausible *rlgg* algorithm for a set of example clauses

```
• given:
    – a logic program ($\mathcal{K}$)
    – a set of example clauses ($SC$)
• Take an example clause ($C_1$) from $SC$. Let $\theta_{1,1}$ be a substitution grounding the
  variables in the head of $C_1$ to new constants and $\theta_{1,2}$ grounding the remaining
  variables to new constants
• Construct a new clause ($NC$) defined as:
  $NC \equiv C_1\theta_{1,1} \cup \{\neg A_{1,1}, \neg A_{1,2}, \ldots\}$ where
  $\mathcal{K} \wedge C_{1body}\theta_{1,1}\theta_{1,2} \models A_{1,i}$, and $A_{1,i}$ is a ground atom
• Set $SC = SC - \{C_1\}$
• while $SC \neq \{\emptyset\}$
    – Take a new example clause ($C_j$) from $SC$. Let $\theta_{j,1}$ be a substitution ground-
      ing the variables in the head of $C_j$ to new constants, and $\theta_{j,2}$ grounding
      the remaining variables to new constants
    – Construct a new clause ($C'_j$) defined as:
      $C'_j \equiv C_j\theta_{j,1} \cup \{\neg A_{j,1}, \neg A_{j,2}, \ldots\}$ where
      $\mathcal{K} \wedge C_{jbody}\theta_{j,1}\theta_{j,2} \models A_{j,k}$ and $A_{j,k}$ is a ground atom
    – Set $NC = lgg(C'_j, NC)$
    – Set $SC = SC - \{C_j\}$
• output $NC$
```

to achieve practical results. PAL (i) uses a pattern–based background knowledge representation to derive a finite set of facts and (ii) applies a novel constraint which identifies the role of the components in different example descriptions to reduce the complexity of the *lgg* algorithm (these are explained below).

Examples in Pal are given as sets of ground atoms (e.g., descriptions of musical scores stating the notes involved on each voice). In general, a musical score can be completely described by the tone and height of each note involved, its time interval and the voice where it belongs. For counterpoint rules of the first specie, time intervales can be ignored (we are beginning to investigate how to include time intervals in the descriptions of scores) and the examples were described by two–place atoms (*note/2*) stating the tone and height of each note and its voice. For instance, *note(c/4, voice1)* states that a "*c*" note in the center of the piano scale (*4*) belongs to voice one. Other notes of the same or different voices can be described in the same way. Other examples descriptions have been used in chess (Morales, 1991; Morales, 1992a) and qualitative modelling (Morales, 1992b). Each example description is added to the background knowledge from which a finite set of facts are derived.

In the musical context Pal induces pattern definitions with the following format:

$$Head \leftarrow D_1, D_2, \ldots, D_i, F_1, F_2, \ldots$$

where,

- *Head* is the head of the musical rule (pattern definition). Instantiations of the head are regarded as musical patterns recognized by the system.
- The D_is are "input" predicates used to describe scores (e.g., *note/2*) and represent the components which are involved in the pattern.

- The F_is are instances of definitions which are either provided as background knowledge or learned by Pal, and represent the conditions (e.g., relations between notes and voices) to be satisfied by the pattern.

Pal starts with some pattern definitions as background knowledge and use them to learn new patterns. For instance, the definition of *inter_class2/3* was given to Pal as follows:

```
inter_class2(Note1,Note2,Type) ←
        note(Note1, Voice1),
        note(Note2, Voice2),
        interval(Inter,Note1,Note2),
        int_class(Valid,Inter,Type).
```

where *interval/3* is a background knowledge definition that returns the musical interval between two notes, while *int_class/3* returns if an interval is valid/invalid with its type for valid intervales (i.e., perfect conssonance, imperfect conssonance, or dissonance). This definition gets instantiated only with example descriptions with two notes of different voices and returns if they form a perfect/imperfect consonance or a dissonance.

Given an example description, Pal "collects" instantiations of its pattern–based background knowledge definitions to construct an initial hypothesis clause. The head of the clause is initially constructed with the arguments used to describe the first example description. The initial head, in conjunction with the facts derived from the pattern definitions and the example description, constitutes an initial concept clause. This clause is generalized by taking the *lgg* of it and clauses constructed from other example descriptions.

Even with a finite theory for music, the large number of plausible facts derivable from it, makes the finiteness irrelevant in practice (e.g., consider all the possible intervals between notes in music). In Pal a fact F is *relevant* to example description D if at least one of the ground atoms of D occurs in the derivation of F. Since PAL constructs its clauses using pattern–based definitions, only a finite set of relevant facts are considered.

The size of the generalized clauses is limited by requiring all the variable arguments to appear at least twice in the clause. In addition, PAL uses a novel constraint based on labelling the different components which are used to describe examples to guide and constrained the *lgg* algorithm. For instance, notes in the following example are assigned unique labels as follows:

$$\text{note(c/4,voice1)} \longrightarrow \text{note}(c_\alpha/4_\beta,\text{voice1})$$
$$\text{note(c/5,voice2)} \longrightarrow \text{note}(c_\gamma/5_\delta,\text{voice2})$$

The labels are kept during the derivation process, so the system can distinguish which component(s) is(are) "responsible" for which facts derived from the background knowlegde, by following the labels. For example, instances of *inter_class2/3* will use the same labels:

$$\text{inter\_class2(c/4,c/5,perf\_cons)} \longrightarrow \text{inter\_class2}(c_\alpha/4_\beta,c_\gamma/5_\delta,\text{perf\_cons})$$
$$\ldots$$

The *lgg* between compatible literals is guided by the associated labels to produce a smaller number of literals, as *lgg*s are produced only between compatible literals with common labels (a simple matching procedure is used for this purpose). In music this constraint identifies corresponding notes (i.e., notes which are involved in the same relation in different examples). The labels used in one example for the first note are associated with the first note of another example. Thus Pal requires that the corresponding components are presented in the same order.

5 Experiments and results

The following musical knowledge was provided to Pal:

- *inter_class1(Note1,Note2,Valid)*: describes if two notes from the same voice have a valid/invalid interval. Where valid intervals can be consonances or dissonances which follow the same modality of the *cantus firmus* (i.e., a 7th. or augmented 4th. would be invalid)

- *inter_class2(Note1,Note2,Conso)*: describes if two notes of different voices form a perfect or imperfect consonance or a dissonance

Pal was given manually the examples for each rule. It should be noted that the second author suggested all the examples and musical knowledge without knowing the exact functioning of the system. The number of examples required to learn each rule is given below:

Rules	Rule1	Rule2	Rule3	Rule4
Number of examples	6	4	5	5

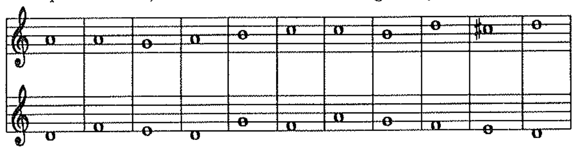
The first rule induced by Pal is shown below (the other three rules are very similar changing only in the different combinations of *perf_cons* and *imperf_cons*).

```
rule(Note1, Note2, voice1, Note3, Note4, voice2) ←
    note(Note1, voice1),
    note(Note2, voice1),
    note(Note3, voice2),
    note(Note4, voice2),
    inter_class1(Note1, Note2, valid),
    inter_class1(Note3, Note4, valid),
    inter_class2(Note1, Inter1, perf_cons),
    inter_class2(Note2, Inter2, perf_cons).
```

The rules learned by Pal were tested for analysis on simple counterpoint pieces. We add an extra argument to each rule to distinguished them from the rest. The analysis was made with the following program:

```
analysis([Note1,Note2|RVoice1], [Note3,Note4|RVoice2],
        [NumRule|Rules]) ←
    rule(Note1, Note2, Note3, Note4, NumRule),
    analysis([Note2|RVoice1],[Note4|RVoice2],Rules).
analysis([ _ ],[ _ ],[ ]).
```

For instance, for the piece below, we obtained the following analysis:



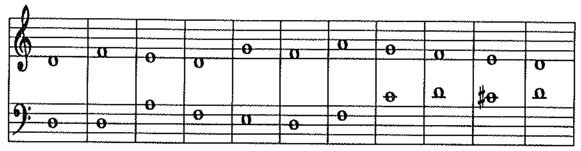
```
?- analysis([d4,f4,e4,d4,g4,f4,a4,g4,f4,e4,d4],
        [a4,a4,g4,a4,b4,c5,c5,b4,d5,cs5,d5],
        Rules).
```

Rules = [r1,r4,r3,r2,r3,r2,r4,r4,r4,r2].

The same program can be used for musical generation. For example, given the *cantus firmus* below, we can generate the required counterpoint notes:

?- analysis(Notes,[d4,f4,e4,d4,g4,a4,g4,f4,e4,d4],[r1,r3,r3,...]).

Notes = [d3,d3,a3,f3,e3,d3,f3,c4,d4,cs4,d4].



6 Conclusions and future work

In (Widmer, 1992), Widmer describes a system capable of learning counterpoint rules using an Explanation–based learning approach (de Jong, & Mooney, 1986). Unlike Pal, a generalization of the target counterpoint rules is required as background knowledge, from which the more specific counterpoint rules are derived. By contrast, Pal uses a much simpler background knowledge to induce equivalent rules.

In this paper, it is shown how Pal, an ILP system, can effectively learn simple counterpoint rules from general purpose musical knowledge and simple example descriptions. The learned rules can be used for musical analysis and generation. This is an initial step towards learning more complicated musical rules expressing personal criteria follow by composers. If we succeed in our goal, the learned rules would provide explicit knowledge of preference criteria follow by composers. This knowledge could be used for analysis and provide suggestions for musical compositions.

References

W. Buntine (1988). Generalised subsumption an its applications to induction and redundancy. *Artificial Intelligence*, **36**(2):149–176.

De Raedt, L. & Bruynooghe M. (1988). On Interactive Concept-Learning and Assimilation. In D. Sleeman & J. Richmond (Eds.), *Proc. of the third european working session on learning*, EWSL–88, pp. 167–176, London, Pittman.

Fux, J.J. (1971). *Gradus ad Parnassum*, 1725. Translated and edited by Alfred Mann, W.W. Norton & Company.

Morales, E. (1991). Learning Features by Experimentation in Chess. In Y. Kodratoff (Eds.) *Proceedings of the European Working Session on Learning*, (EWSL–91), pp. 494–511, Berlin, Springer–Verlag.

Morales E. (1992a). Learning Chess Patterns. In S. Muggleton (Eds.), *Inductive Logic Programming*, pp. 517–537, London, Academic Press, The Apic Series.

Morales, E. (1992b). First-Order Induction of Patterns in Chess, *Ph.D. Thesis*, The Turing Institue – University of Strathclyde, Glasgow.

Muggleton S. & Feng, C. (1990). Efficient induction of logic programs. In S. Arikaxa, S. Soto, S. Ohsuya, & T. Yokomari (Eds.), *Proceedings of First International Workshop on Algorithmic Learning Theory*, (ALT90), pp. 368–381, Tokyo, Japan, Ohmsha.

Muggleton, S. (1991). Inductive Logic Programming, *New Generation Computing* **8**: 295–318.

Niblett, T. (1988). A study of generalisation in logic programs. In D. Sleeman & J. Richmond (Eds.), *Proc. of the Third European Working Session on Learning* (EWSL–88), pp. 131–138, Glasgow, Pittman.

Plotkin, G.D. (1969) A note on inductive generalisation. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 5*, pp. 153–163, Edinburgh, Edinburgh University Press.

Plotkin, G.D. (1971a). A furhter note on inductive generalisation. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 6*, pages 101-124, Edinburgh, Edinburgh University Press.

Plotkin, G.D. (1971b). Automatic Methods of Inductive Inference. *Ph.D. Thesis*, Edinburgh, Edinburgh University.

Widmer, G. (1992). The importance of basic musical knowledge for effective learning, In M. Balaban, J. Ebcioğlu & O. Laske (Eds.), *Understandig Musica with AI: Perspectives on Music Cognition*, Cambridge - Menlo Park: AAAI Press/MIT Press.