Proceedings of the 15th International Conference on Auditory Display, Copenhagen, Denmark May 18 - 22, 2009

STATISTICAL RULES IN CONSTRAINT-BASED PROGRAMMING

Mikael Laurson, Mika Kuuskankare, Kimmo Kuitunen — Sibelius Academy, CMT, Finland, laurson@siba.fi, mkuuskan@siba.fi

Örjan Sandred — University of Manitoba, Studio FLAT, orjan@sandred.com

ABSTRACT

In this paper we introduce a system that first generates statistical analysis data from a musical score. The results are then translated automatically to constraint rules that in turn can be used in combination with ordinary rules to generate scores that have similar statistical distributions than the original. Statistical analysis rules are formalized using our special rule syntax where our focus will be in the pattern-matching part of the rules. The pattern-matching part has two important tasks in our paper: first, it is used to extract various musical entities from the score, such as melodic, harmonic and voice-leading formations; second, it is used also to generate statistical rules which will be used in the re-synthesis part of our system. We first introduce the rule syntax. After this we discuss a practical case study where we analyze a melodic line. Finally we generate out of this material statistical rules which are used to produce new scores.

1. INTRODUCTION

Constraint-based languages in computer-assisted composition environments have lately found increased interest. Besides our constraint-based system, called PWGLConstraints [4], [6], there are currently several other approaches oriented towards musical search problems such as Situation [7], OMClouds [8] and the more recent system by Anders based on the OZ programming language [1]. PWGLConstraints is written in Common Lisp and CLOS and is currently an integral part of our visual programming environment called PWGL [5]. When using our system we define a search-space and produce systematically potential results from it. Typically we are not interested in all possible results, but constrain these with the help of rules describing an acceptable solution. If the search cannot find an acceptable solution during the search process, then the system will backtrack in the search-space. This scheme thus allows to undo already accepted values and retry to satisfy the rules with a new set of values. The rules have a strict structure where a pattern-matching part (or PM-part) header is followed by a Lisp expression (Lisp-code part).

Although PWGLConstraints was originally designed to work as a tool to generate musical material it can also be used for music analysis purposes. In this case no backtrack search is involved and the system simply traverses through score and applies all analytical rules to the input score. Analytical rules have a similar syntax to the ones used in search problems. This kind of symmetry where a rule can be used either in a generative or analytical context is one of the main corner stones of our system.

The PM-part of a rule is critical in understanding how rules work as the same scheme is used both in the analytical and generative tasks that will be described later in this article. Furthermore the PM-part has important implications for user written rules and our software development. First, the PM-part is responsible for extracting in a uniform way musical information that can then be processed further in the Lisp-code part of the rule. Second, the PM-part handles boundary cases that guarantees that rules are only run when all necessary information is available during search. Third, the PM-part allows to separate user code from the actual implementation of the system. Thus for instance optimizations can be done without touching user defined rules.

This paper discusses a case study where a set of analytical rules are translated automatically to generative rules. Our focus will be on statistical rules where we calculate the frequency distribution of various musical entities found in a score. The idea of using statistics in computer assisted composition, performance research and sound synthesis has been obviously popular and it has a long history, ranging from Lejaren Hiller's [3] and Iannis Xenakis' [10] pioneering compostional work from mid-50s onwards to more recent systems such as style simulations of Palestrina-style counterpoint [2] and performance research [9]. In this paper we will focus on the specific problem how statistical analysis can be useful in a classical constraint-based context. Statistical rules are an important complement to our ordinary rule system as they allow us to express more precisely properties such as 'rare' and 'common'.

We start by an introduction section explaining the rule structure in detail. After this we use this knowledge in a practical case study where we analyze a melodic line. The resulting statistical data is then translated into rules which are used with ordinary rules to produce new musical material. The results are then analyzed with the original analysis rules to verify how close the statistical results correlate with the original. We discuss also a more advanced case study where we utilize the analytical system in our constraint-based environment to generate a 2-part canon.

2. RULE STRUCTURE

In the following we first introduce the general rule syntax of PWGLConstraints.

A rule consists of three main parts : (1) a PM-part, (2) a Lisp-code part, and (3) a documentation string. In the PM-part a pattern-matching language is used to extract relevant information from a score. This information is given to a Lisp expression (situated in the Lisp-code part) that either returns a truth value or executes a side-effect. In the latter case the side-effect can typically be an expression object that is inserted in a score; or, it can also be an operation where some analytical information is written for instance to a hash-table. The Lisp-code part is found after the PM-part and starts with the symbol '?if'.

The PM-part of a rule uses a special pattern-matching syntax. It can contain variables (symbols starting with a '?'), anonymous-variables (plain '?'s), index-variables (symbols consisting of an 'i' and an index number), and one or two wild cards (denoted by the symbol '*'). These variables can be followed by special keywords (typically accessors and selectors) that further specify the behavior of the PM-part.

A variable extracts single objects. By contrast, an anonymous-variable is never bound to an object, i.e. it only acts as a 'place-

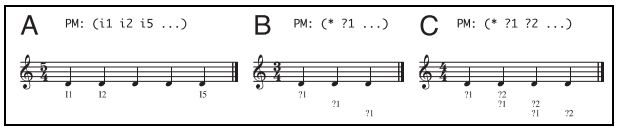

Figure 1: Three PM-part examples where possible matches are given below the staff in separate rows. In (A) there is only one match, in (B) and (C) there are three matches.

holder' in the pattern. The wild card matches any continuous part of the score. Finally, an index-variable extracts objects from an absolute position given by the index number. In Figure 1 we give some basic pattern-matching examples with their respective bindings (we show here only the PM-part of the rule; the Lisp-code part of the rule is denoted with three dots).

Using this information let us define a simple interval rule where we check that all melodic intervals belong to a given interval list. This rule is run for each melodic note-pair:

```
(* ?1 ?2                          ; (1) PM-part
  (?if (member (- (m ?2) (m ?1))  ; (2) Lisp-code part
          '(-1 3 -5 -4 -13 -8 -9 8 -11)))
        "Interval rule")          ; (3) doc string
```

The PM-part allows the user to specify after the variables special score accessor keywords. The variables given in the PM-part of a rule can now refer to the structural entity the user is interested in, such as note, chord, beat, measure and harmony. A more special case, called 'score-sort', will be dealt with in Section 4. If no accessor keyword is given then the rule operates with melodic notes (see for instance the "Interval rule" in the previous subsection).

The following examples demonstrate how we can use some of the accessor keywords in conjunction with a wild card and two variables:

```
(* ?1 ?2 ...)              ; 2 adjacent melodic notes
(* ?1 ?2 :chord ...)       ; 2 adjacent chords
(* ?1 ?2 :beat ...)        ; 2 adjacent beats
```

The PM-part may also contain selectors which allow to restrict the scope of the rule. A selector is a keyword/value pair, or it can also be a Lisp expression that returns a truth value. For instance, the following the melodic rule is applied only to parts 1 and 3:

```
(* ?1 ?2   :parts '(1 3) ...) ; only parts 1 and 3
```

3. STATISTICAL DISTRIBUTION AND AUTOMATIC RULES

Until now we have discussed rules that either accept or reject a candidate. Typically these kind of rules operate in a fairly local context. It can, however, be interesting to have also a more holistic view and to be able to say something about the commonness of a property in the musical output. For instance let us consider the interval rule from the previous section. This rule works only in the context of two adjacent melodic notes: it does not state anything about the distribution of the intervals. We have no obvious way of saying that we would like to have a lot of interval -1, or that interval 6 should be very rare. Another problem is that this rule gives no guarantee of how the given interval repertoire will be be used. In a pathological case a result might use only one or two of the allowed intervals, although the rule contains 9 intervals. One solution to these problems is to make during search statistics of the partial solution and compare this result with the desired distribution. The purpose of the statistical rules is to force the result to approximate gradually the desired distributions.

3.1. Analysis

For statistical rules we need first some data. An apparent starting point is to analyze a musical score and use this material later to generate a score with similar properties. Thus next we will discuss a case study that utilizes four rules to analyze a melodic line. Our starting point is the soprano line from a 3-part canon by Anton Webern (Op. 16. No. 3). This piece stems from the middle-period songs that have been analyzed much less than the late-period works.

We first extract only the pitch information in MIDI-values (Figure 2, upper part). Then we apply the following analysis rules to this pitch material. The Lisp-code part of all rules contain the function 'add-fd-entry' ('fd' = frequency distribution) which accumulates during the analysis process the occurrences of any given criteria. Thus, for instance, the first rule, called "interval", will count for interval occurrences for each melodic note-pair. The second and third rules, "3-card scs" and "4-card scs", perform pitch-class set-theoretical analysis for 3-note and 4-note melodic formations. Finally, the fourth rule, "+-movement", analyzes up/down movement distributions for all 4-note successions.

```
(* ?1 ?2
   (?if (add-fd-entry :int (- (m ?2) (m ?1))))
 "interval")

(* ?1 ?2 ?3
   (?if (add-fd-entry :3-card-scs
            (sc-name (list (m ?1) (m ?2) (m ?3)))))
 "3-card scs")

(* ?1 ?2 ?3 ?4
   (?if (add-fd-entry :4-card-scs
          (sc-name (list (m ?1) (m ?2) (m ?3) (m ?4)))))
 "4-card scs")

(* ?1 ?2 ?3 ?4
   (?if (add-fd-entry :+-movement
             (list (signum (- (m ?2) (m ?1)))
                   (signum (- (m ?3) (m ?2)))
                   (signum (- (m ?4) (m ?3))))))
   "+-movement")
```

For instance, the first interval rule results in the following distribution (each sublist consists of a interval/count pair: 7 instances of interval -1, 5 of 3, 5 of -5, etc.):

```
((-1 7) (3 5) (-5 5) (11 5) (-4 4) (-13 3) (-8 3) (-9 2)
 (8 1) (-11 1) (15 1) (20 1) (14 1) (-6 1) (9 1) (-2 1)
 (5 1) (6 1))
```

3.2. Automatic Rule Translation

Let us next consider how statistical rules will work in a generative context. For this we make a simplification of the general case and assume that we will create an example that has the same number of notes as the original melodic line has. For instance a generative interval rule that uses the interval distribution described in the previous subsection would accept a partial solution that has less than or equal to 7 times interval -1, less than or equal to 5 times interval 3, etc. In other words the count value determines the high limit of instances of a given interval. If, say, the interval -1 is found 8 times or more, then the rule will not accept the partial solution. This rule may be too strict and time consuming and therefore we add a tolerance to the rule. The tolerance indicates how much the high limit can be exceeded without failing. This means that we will get only an approximation of the desired distribution, but giving a tolerance value will speed the calculation considerably.

The automatic translation is quite straightforward due to the PM-part of the analysis rule. We simply use the PM-part of the analysis rule and compare in the Lisp-code part the desired (analyzed) distribution with the actual distribution during search.

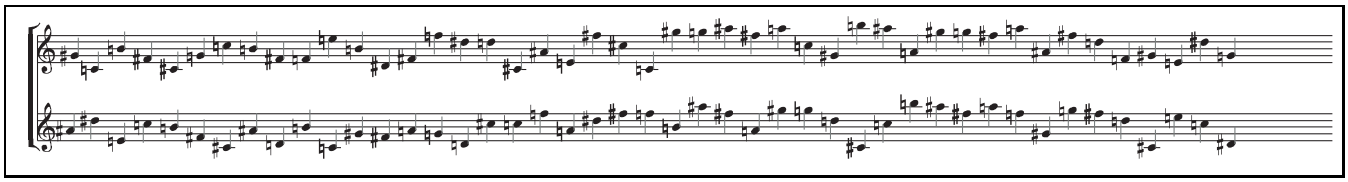

Figure 2: The original melodic line and a result produced by four automatic rules and two ordinary rules. Upper part: original melodic line. Lower part: generated melodic line.

3.3. Using Distribution Rules for Dissimilar Scores

Our example in the previous subsection is somewhat artificial as we used absolute counts. This means that our example works only if the search problem has exactly the same amount of notes than is found in the original. If the original and target scores differ, then an extra analysis phase is needed before generating the final statistical rules. After the first distribution analysis of the original we run the same analytical rules for the target only to count how many times a given analytical rule is called for the target score. Using this information we can normalize the statistics that are used for the final calculation. For instance, if the original score calls 44 times the interval rule, and the target score calls the same rule, say, 88 times, we must multiply the count values by 2 (= 88 / 44).

4. GENERATION

In this section we utilize the four analysis rules described in the previous section to generate a melodic line with similar distributions than the original (see Figure 2, lower part). To make the example more interesting we add two ordinary rules. The first one, called ”ballistic”, works in the context of two consecutive melodic intervals: a ballistic movement allows two jumps in the same direction, but the larger jump has to be below the smaller one. The second rule, called ”skyline”, first performs a data reduction of all previous melodic pitches [4]. The reduction preserves only the highest peek values, or skyline, of the melodic sequence. After this we check that there are no large jumps between two skyline values. We also avoid repetitions of values. The net effect is that the peek values evolve relatively smoothly without repetitions.

To complete our example we give below the interval distribution of the result. As can be seen the original distribution is preserved within the tolerance which was in our case equal to 1:

```
((-1 6) (11 6) (-4 5) (-5 4) (-9 3) (3 3) (-11 2) (-13 2)
 (-2 2) (-8 2) (5 2) (8 2) (9 2) (-6 1) (15 1) (6 1))
```

The other three statistical rules (see Section 3.1) produce similar results and stay well within the limits as specified by the original distributions and tolerance values.

5. 2-PART CANON GENERATION

In this final section we briefly outline the realization of a 2-part canon (Figure 3), where the pitch content of the two melodic lines are based on statistical and ordinary rules. The statistical rules are similar to those given in Sections 3 and 4. Our starting point here is the 2-part mirror canon (i.e. the subsequent voice imitates the initial voice in inversion) Op. 16. No. 2 by Anton Webern. This example is clearly more complex (and realistic) than the previous one: the search utilizes six statistical rules and 23 ordinary ones.

The ordinary rules consists of a canon rule, general melodic rules, and more special rules that control musical phrases, melodic contours, forbidden harmonic formations and voice-leading. All rules can access the expression markings and the text of the original score, thus making it possible to control, for instance, individual phrases in a detailed manner. The six statistical rules, in turn, consist of four melodic rules that are based on the same principles as discussed in previous sections.

The two remaining statistical rules are quite special: they are based on our ordering scheme, called score-sort, how the engine proceeds in a polyphonic score during search [4]. The ordering is accomplished as follows. We read a score from left to right and sort notes in the order they appear in it. If two or several notes share the same attack time, they are sorted so that the longest notes are placed before the shorter ones. If two or more notes have the same attack time and the same duration, we use the convention that notes having the highest part number are considered first (i.e. we start from bass notes). Thus score-sort is a kind of ’inter-melodic’ formation where the notes, which are typically distributed between several parts, are processed in the order they appear in a sequence defined by the score-sort algorithm. Our reason in using this kind of special score access is motivated by the very sparse texture of our example where each new note entry is important from a perceptional point of view. This kind of control that traverses across different parts could not be gained using ordinary melodic rules.

In Figure 3 the end part of the score-sort order structure is denoted by arrow-head curves (see the last row of the piece, measures 11-13; we omit here the original expression markings for greater clarity). These score-sort analysis rules use our accessor keyword scheme (Section 2) and they are based on a pitch-class set-theoretical analysis of the original 2-part canon (thus they are similar to the ”3-card scs” and ”4-card scs” rules found in Section 3). The 3-card variant is defined as follows:

```
(* ?1 ?2 ?3  :score-sort
   (?if (add-fd-entry :3-card-score-sort-scs
              (sc-name (list (m ?1) (m ?2) (m ?3)))))
 "score-sort 3-card scs")
```

6. CONCLUSIONS

This paper describes an important addition to our constraint-based system helping the user to capture important musical features more precisely than using ordinary rules only. Statistical rules can be generated automatically from existing repertoire. The system is evaluated with the help of two case studies. The results can be listened to at: `http://www.siba.fi/pwgl/demos/cmmr09-statscsp.html`.

7. ACKNOWLEDGEMENTS

The work of Mikael Laurson and Mika Kuuskankare has been supported by the Academy of Finland (SA 105557 and SA 114116).

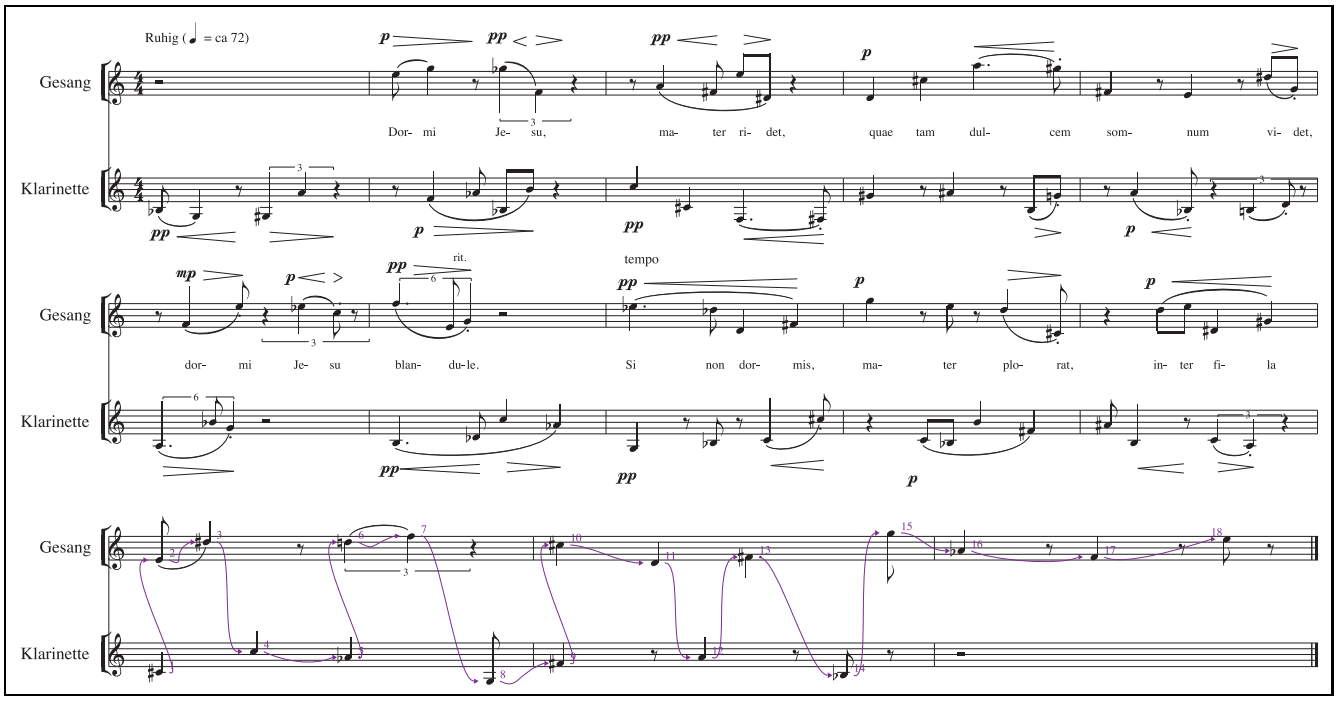

Figure 3: A 2-part mirror canon realization based on statistical and ordinary rules. The rhythmic structure, text and expression markings come from the original canon by Anton Webern (Op. 16. No. 2). The pitch information, in turn, is generated by our constraint solver.

8. REFERENCES

[1] Torsten Anders. *Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System*. PhD thesis, Queen's University, Belfast, 2007.

[2] M. Farbood and B. Schoner. Analysis and synthesis of palestrina-style counterpoint using markov chains. In *Proceedings of International Computer Music Conference*, Havana, Cuba, 2001.

[3] L. Hiller and L. Isaacson. Musical composition with a high-speed digital computer. *Journal of the Audio Engineering Society*, 1958.

[4] Mikael Laurson. *PATCHWORK: A Visual Programming Language and some Musical Applications*. Studia musica no.6, doctoral dissertation, Sibelius Academy, Helsinki, 1996.

[5] Mikael Laurson and Mika Kuuskankare. Recent Trends in PWGL. In *International Computer Music Conference*, pages 258–261, New Orleans, USA, 2006.

[6] Mikael Laurson, Mika Kuuskankare, and Kimmo Kuitunen. The Visualisation of Computer-assisted Music Analysis Information in PWGL. *Journal of New Music Research*, 37(1):61–76, 2008.

[7] Camillo Rueda, Magnus Lindberg, Mikael Laurson, Georges Bloch, and Gerard Assayag. Integrating constraint programming in visual musical composition languages. In *ECAI 98 Workshop on Constraints for Artistic Applications*, Brighton, 1998.

[8] Charlotte Truchet, Gerard Assayag, and Philippe Codognet. Visual and adaptive constraint programming in music. In *International Computer Music Conference*, pages 346–352, Havana, Cuba, 2001.

[9] Gerhard Widmer and Werner Goebl. Computational models of expressive music performance: The state of the art. *Journal of New Music Research*, 33(3):203–216, 2004.

[10] Iannis Xenakis. *Formalized Music: Thought and Mathematics in Composition*. Hillsdale, NY: Pendragon Press, 2001.