Abstract

This article describes several ways to automatically create music from the XML-formatted execution trace of the resolution of a Constraint Satisfaction Problem. These ways are essentially based on associations between trace events and music events. Multiple channels are used to auralize different informations of the solver’s execution. The result is surprisingly musical. The structure of a piece is not linear but follows the shape of a tree (the one used to search for the problem’s solutions) and forks into many different detailed developments. We present some of these associations here, with a description of which information of the trace it is possible to hear through the music generated. Finally, the interests, both for a musician and for a programmer, are listed.

1 Introduction

Using constraint satisfaction resolution problems in music is not a new idea. Many computer-assisted music software and editors are currently searching how to use Constraint Satisfaction Problems (or CSPs) and local search methods in order to include in their applications some powerful automatic composition tools. For example, the Sony research group published several articles about using CSPs in automatic harmonization\(^1\) (Pachet and Roy 2001; Pachet and Roy 2004).

At the same time, Charlotte Truchet presented in Truchet, Assayag, and Codognet (2001) and Truchet (2004a) many CSPs which model problems presented by contemporary composers. A solution of this kind of CSP is a piece of music solving all the defined constraints, in order to give the composer some “raw material” to begin with. Those CSPs describe a large panel of rhythm and harmonization problems, as well as modelling what is physically playable by a real instrumentist. A computer is then able to propose some musical scores which satisfy the constraints defined by the composer.

However, this article proposes a different view: we use the execution trace of a CSP resolution to automatically generate music. Each musical event is associated to a particular number found in the execution trace. A set of associations is called a style and we provide five of them, called “Philip Glass”, “Hard-rock”, “Debug”, “New-Age” and “Apocalyptic”. Because every program has a different execution behavior, it gives birth each time to new musical pieces.

In contrast with sequences created by a random generator, in which the music has most of the time no structure at all – but can nevertheless sound good – our approach produces highly structured music that reproduces the tree-like structure which is at the core of the resolution of a CSP. Other structured approaches seldom use a fractal generator, in which there is a hope to hear the self-similarity that characterize fractals. In our system, the associations between trace events and musical events are freely chosen by the user, from simple pitch assignment (the produced numbers are restricted to integers from a given interval, then a fixed length note, which pitch is function of the obtained number, is appended to the music), to more complex algorithms, based on advanced musical theory notions.

2 Preliminaries

2.1 On the constraint side

A CSP is a set of logical relations called constraints, which involves a finite set of variables having a finite number of possible values (called their domain). A solver is a piece of software which finds all the solutions of a CSP, i.e., all the possible variable assignments such that all the constraints are satisfied. For this it usually implements a chronological backtracking mechanism: a variable is selected and assigned to one of its yet possible values and the search goes on recursively until all variables are instantiated or an inconsistency is found. This assignment is marked by a choice point. In case an inconsistency is found, the control backtracks to the most recent choice point. A consistency algorithm is inter-
leaved between all search step. The purpose of consistency is to prune the domain of unassigned variables in order to reduce the future exploration the search space. Arc-consistency may prune all possible values inside the domain while bound-consistency only contracts the bounds.

An execution trace format in XML has been defined in the OADymPPAC project in July 2004 and is called the Generic Trace Format for Constraint Programming, or GenTra4CP (The OADymPPaC RNTL Project 2004). It precisely defines all the encountered events that have to be reported during the resolution of a CSP, but remains flexible enough to allow quite a large panel of solvers to produce a trace in this format. GenTra4CP format is public and precisely specified.

A constraint solver manipulates variables, and maintains up-to-date their current domain. The trace indicates each creation of a variable by a <new-variable> marker, which attributes are: a unique variable identifier vident, the name vname given to the variable by the user (if it exists) and the internal name vinternal of the variable in the solver. The <variable> marker, which has the same attributes, is then used each time an already created variable is referred.

Those two markers usually contain the <vardomain> marker, which represents the current domain of a variable by a union of several intervals, each being reduced possibly to one particular value.

The creation of each constraint is reported in the trace by a <new-constraint> marker. This one has two attributes: the unique identifier cident of the constraint and its internal name cinternal. When domain reduction step makes that all possible remaining assignments satisfy a particular constraint, a <solved> marker appears in the trace, with the identifier of the concerned constraint as attribute.

During the resolution, the solver tries to reduce the domains of all the variables it is aware of for the moment, by detecting and deleting values which falsify a constraint for all possible assignment of all other variables. Such a reduction is marked in the trace by a <reduce> marker and its attributes are the identifiers of the reduced variable and of the constraint which caused this reduction. This marker also contains the set of deleted values, as well as the updated variable domain. In the case this domain becomes empty, it is immediately followed by a <failure> marker, which indicates that the solver has encountered a failure leaf in the search tree. This marker has a node identifier as attribute.

Once the solver has performed all possible reductions, it comes to either a solution or a choice point, depending on whether all variables are instanciated or not. These events are represented in the trace respectively by a <solution> and a <choice-point> marker. Both have a node identifier nident as attribute. They also contain a general description of the current state, i.e. the list of all current variable domains.

This current state is represented in a <state> marker.

When the solver has reached a choice point, one of the variables is instanciated to each of its yet possible values, creating in turn as many subproblems as needed and each branch is recursively tried. Reaching a solution or a failure leaf is immediately followed by a backtrack to the most recent choice point and the state is restored as it was at this computation point. This event is reported by a <back-to> marker including the identifier node to where we backtrack and the identifier node-before of the solution or failure node where it came from. A description of the state is also given.

Ludovic Langevine created an extension of the Constraint Logic Programming language GNU-Prolog (Diaz and Codognet 2001) called Codeine (Langevine, Ducassé, and Deransart 2003). It creates a trace in this format when solving a CSP via the integrated finite domains solver of GNU-prolog. The following example gives a general idea of the XML trace:

```xml
<solution chrono="241" depth="8" nident="11">
  <state>
    <variable vident="v1" type="int" vinternal="_#3">
      <vardomain min="2" max="2" size="1">
        <range from="2" to="2"/>
      </vardomain>
    </variable>
  </state>
</solution>
```

This trace extract indicates that a solution has just been discovered, at depth 8 in the search tree. This solution leaf has node identifier 11 and the first variable, v1, is an integer internally named in GNU-prolog as _#3 which has been assigned to the value 2.

So, an execution trace contains different events, coming with different quantitative attributes, which will be auralised as well.

### 2.2 On the musical side

#### The MIDI format

The MIDI format is a binary stream of musical events. In making CSP Singer, we did not generate directly a MIDI stream but we have rather used the “MIDI assembly code” text format, which can be compiled to a MIDI file using a freeware called Midi file Assembler (Glatt 2001). A MIDI file contains a stream in the eponym format. It does not directly describe the audio wave to reproduce, as MP3 of wav format do, but only all the elementary musical events a score is composed of, like the beginning and the end of a note, its volume and frequency (pitch) variation. When reading this kind of file, a computer or a synthesizer interprets these informations using a bank of instruments.  

Technically, the MIDI format enables to send musical events on 16 different channels. The main events are:

\[^{2}\text{A set of recordings of several instruments which permit to artificially reproduce a melody.}\]
• **NOTE ON(PITCH, VOLUME):** starts to play a note at the indicated pitch and volume. The note will only be stopped at the encounter of a further **NOTE OFF** event with the same pitch attribute.

• **NOTE OFF(PITCH):** releases a note previously started by a **NOTE ON** event.

• **PROGRAM(INSTRUMENT):** Changes the instrument associated to this channel. Every future **NOTE ON** event will be interpreted using this new instrument.

The pitch, volume and instrument parameters are integers from 0 to 127. The pitch difference between two consecutive numbers is one semitone. So, Midi format can directly describe all the music where notes are all on a chromatic scale\(^3\), i.e. virtually all the occidental music, but not, for example, some mid-oriental pieces of music in which notes can vary of one quarter tone, or even less.

Channel 10 is by convention reserved to percussions. So, in this channel, the pitch parameter of **NOTE ON** events represents the percussive instrument played (bass drum, snare drum, cymbal, etc.), and **PROGRAM** is used to indicate the percussive set used (normal, room, power, etc.). This list is far from being exhaustive. However, these events are enough to generate more than the essence of a piece of music.

### 3 CSP Singer

The CSP Singer software is an implementation of the following concept: a set of associations between trace and musical events is used to generate music according to the execution trace of a given CSP. The music is produced in the Midi Assembly format, which needs to be converted into actual Midi format in order to be listened to.

At opening, a dialog box invites the user to choose a trace XML file to be auralized. Then, after a first global analysis, it tells the maximum depth of the search tree and the number of markers the document contains. It also displays a list of all encountered choice points and backtracking, with their corresponding marker number. This information gives a first idea of the search tree, as well as which set of markers describes which part of the search tree.

The user is then invited to specify which part of the file will be used to generate the music by giving a start and a stop marker number. This is very useful when auralizing large problems, with several millions markers, because a complete auralization would produce several days long pieces of music. It is then asked if all the variables must be auralized or not, and which one should be. This can be interesting in order to concentrate on one or a few particular variables, for example for debugging the CSP, or to eliminate some variables whose domain is too small to produce a really interesting music (a variable with a domain of 0, 1 is likely to generate some morse code or a french police siren sound, rather than music...).

Then the style of music has then to be chosen between “Philip Glass”, “Hard rock”, “Debug”, “New age” and “Apocalyptic”. Each style generate its own kind of music, we believe in a way related to its name. Here is now a complete description of associations between musical and trace events CSP Singer contains for the moment:

**“Philip Glass” style.** Historically, this style has been the first implemented in the system. It has been named after the contemporary composer, for the generated music sounds quite like his work. Some adaptations have been made such that the generated score could be actually played by a human instrumentist. Here is the list of the associations of this style:

- Tempo is function of the current depth in the search tree (deeper we are, faster the music plays).
- Each backtrack changes the mode (minor/major).
- The encounter of a `<state>` marker changes the key signature, without changing the mode.
- Each variable domain description is noted by a one beat long raise played with a piano. each note corresponds to a possible value. If the domain is too large, the notes within the raise would be too short to be humanly played and so, the raise is replaced by a simple chord.
- A solution is represented by a crash cymbal sound.
- A failure yields a snare drum sound.
- Each new constraint is noted by a chord of clarinet which ends at the next new constraint, key signature or tempo change.

This style asks for a base tempo, and a modifier. The actual tempo is obtained by multiplying the modifier by the current depth in the search tree, and adding the base tempo. A reasonable base tempo setting would be between 60 and 100 bpm (beats per minute), and the modifier should be set such as the tempo never reach 150 bpm: after this limit, the melody would not be playable by an instrumentist. In order to sound more “Glass”-like, we recommend no modification at all. This style produces very minimalist pieces of music. However, we can hear the general structure of the search tree (by concentrating on tempo variations), or the variable domains sizes (proportional with the speed the piano rises). Nevertheless, it would be very difficult to identify

---

\(^3\) Set of notes spaced of one semitone. Major and minor scales are subsets of the chromatic scale.
Figure 1: This is a part of music sheet from a generated music in “Hard rock” style. The lead guitar describes one variable per beat. We can see here that the domain of variables 1 and 2 is composed of two different intervals, whereas the domain of variables 3 and 4 is reduced to a single interval.

precisely every distinct value of a variable domain: in fact, as Paul Vickers also mention in Vickers (1999), fast and accurate identifying of a note pitch is much less natural for a non-musician person than other events, such as rhythmic difference.

“Hard Rock” style. This style produces music which can be played by a rock band composed of a drummer, two guitarists (rhythmic and lead), and a bass-guitarist. Consequently, we obtain a very typical music, where trace events would not be as recognisable as in other styles. The drum score is created from precomposed patterns. Here is the list of the associations of this style:

- The depth in the search tree gives the tempo (in the same way than the preceding style), and the pattern played by the drummer (deeper we are, the more violent is this pattern).

- The encounter of a node in the search tree (choice point or backtrack after a failure or a solution) changes the scale. The new scale is obtained from the node identifier. In general, this identifier is incremented each time a node is created, so a backtracking is more likely to cause a radical change of the scale, whereas progressing deeper in the search tree causes a slighter modification (the melody is transposed one semitone up).

- Variable domains describe the melody played by the lead guitar. Each variable is represented within one beat, and each note represents a bound of one of the intervals composing the domain. Hence more notes are played if the domain is composed of several disjoint intervals (see Figure 1).

- The reduction of a variable domain is indicated by a bass guitar sixteenths descent which start note is given by the number of values taken from the domain: the greater this number is, the higher the starting note will be.

- The rhythmic guitar is function of the pitch of the notes played by the lead guitar, plus another value which is incremented each time it is read.

This style also asks for a base tempo and a modifier. This time, the base tempo should be set between 150 and 170 bpm. As the drum pattern changes according to the depth in the search tree, it is not necessary to give a great modifier. It can even be set to 0. In all cases, the final tempo should never be more than 220 bpm to remain listenable. Even if it was not the first aim of this style, we can hear several information by listening to the generated music: the most obvious one is the fact that arc-consistency has been used or not to solve the problem. Actually, if only bound-consistency has been applied, every domain is described by only the two heights of an interval. In contrast, if arc-consistency is applied, the lead guitar will play faster notes. The general shape of the search tree can also be heard via the changes in the tempo and the drum pattern. Reduction events are also pointed by the sixteenths in the bass guitar, which normally only plays eights.

For example, let us concentrate on the second piece of music provided with this article (2 - Hard rock - 4 queens.ogg), solving the n-queen problem for n = 4. The n-queens problem consists in placing n queens on a n × n board, in such a way that no queen can attack another one, according to the chess game rules, i.e., any horizontal, vertical or diagonal line contains at most 1 queen. It begins calmly, but its tempo and drum violence rises every 4 beats. Hearing this, we can guess that we are going deep in the search tree very quickly (and actually, the search tree of this problem has a 5 nodes long
As the current state is described each time a node is encountered, we can also guess that this problem has 4 variables. As the pitch difference between two notes is always the same, the domains are not reduced during this time. Twelve eights later, the melody changes radically. Hearing two consecutive lead guitar notes with the same pitch is a sign that a variable has been assigned a particular value. The fact that the lead guitar also plays some sixteenths results from the use of arc-consistency. We can guess from the alternation of the three most violent drum patterns that the branches of the subtree we are currently exploring are quite short (we already are deep in the search tree) and numerous. Finally, the music calms down as the final backtracks lead us back to the root of the tree.

“Debug” style. The idea of this style came from the Paul Vickers’s PhD thesis (Vickers 1999), dealing with a Pascal programs auralization called CAITLIN. This work is quite close to what we are doing here: it consists of an extension of the Borland’s Turbo Pascal software which allows an existing Pascal program to produce sounds during its execution. The different instructions such as while loops or conditional produce at execution time different precise little melodies called earcons, each one being recognizable between all. The aim of this project was to help beginner programmers to debug their programs by hearing its execution.

The main difference with our approach is that in CAITLIN, all melodies are predefined and just appended according to execution. In contrast, our approach allows the melody to change according to the program’s internal state, hopefully giving information not only on the control sequence, but also on the reasons why a control sequence is followed. However, a precise comparison of the two approaches must acknowledge that the goals are different. Adding minimal and recognizable sound information to Pascal programs was probably the best choice to help novice programmers to masterize the tool in a short time, i.e. before they learn how to program by other means, in which case the software would not have been useful. From a musical point of view, CAITLIN’s music is for us too repetitive, but of course, the goal was not to produce “music” but more to produce sound information the same way a sound signal may help to quickly identify a problem, for example in a plane cockpit.

Our “Debug” style describes the execution trace in a similar way, associating each event with a particular sequence. We tried to improve it a little bit by adding some composition elements, like changing the pitch of some notes or changing the frequency of a sine wave during search. Here is the list of these associations:

- A new variable definition is represented by a glockenspiel major rising arpeggio.
- A new constraint definition produces a bassoon minor descending arpeggio.
- A constraint resolution is described by a brass section major rising arpeggio.
- A solution encountering is described by a a sequence of brass section quad chord (“Victory” sound).
- A failure encountering produces a descending brass section melody (“Catastrophe” sound).
- A domain reduction is represented by a quick bassoon scale descent.
- A sinus wave, which frequency is determined by the depth in the search tree, is played all the time.
- Every complete description of a domain leads to a single piano note, which pitch is defined by the number of values remaining in the domain.

The aim of this style was not to produce beautiful music, but to give as much information as possible about the trace. For CSP execution, this aim in only partially reached: in fact, auralization of all events by a precise sound is more fitted to imperative languages. We have here a direct description of the execution trace, as well as some information about the domains evolution: lower piano notes indicates a domain with only a few values left. But in CSP, the control is much less known by the programmer, and we found this style not to be very helpful for this kind of problems.

“New Age” style. This style, built from the Philip Glass one, went through important modifications and thus produces somehow different pieces of music than its ancestor. They are very calm and sound like their eponym music, sometimes reminding the Japanese composers Kitaro and Nobuo Uematsu. It is composed of two melodic lines (harp and Aaah choirs), plus a violin accompaniment. Here again, it has been built with a purely musical aim. However, its associations remains quite close from the ones of the Philip Glass style:

- Tempo is related to depth in the search tree.
- Each domain description produces a one beat long harp rise, each note corresponding to a value of the domain. If this later one is too wide, only one fifth chord is played, as in the Philip Glass style. This limit width is however higher, as a harpist can play notes much faster than a pianist.
- Each value taken away from a variable domain produces a note in the Aaah choirs melodic line.
- Violin chords are provoked by a new constraint.
• A solution encounter is represented by a fifth chord played with a Brillance sound.

The pieces of information we can hear with this style of music are quite the same as in the Philip Glass mode, plus the localisation and width of the reduction, indicated by the Aaah choirs. As the music generated by this style is intended to be calm, the tempo should not be set too high, neither have too brutal modifications. A base tempo comprised between 80 and 120 bpm and a modifier set to less than 5 should give quite good results.

“Apocalyptica” style. This style was named after the Finnish cello quartet. It has been created in order to try to associate a trace event suggested by Charlotte Truchet (Truchet 2004b): hearing the average of all variable domain sizes. In this style, one of the cellos is used to play a rhythmic score using low notes, whereas the two others are sharing the melodic scores, the first melody being played one octave above the second one. The rhythm played by the first cello becomes faster as the variable domains tend to narrow. The associations of this style are as follows:

• Tempo is defined by the depth in the search tree.
• The rhythm played by the rhythmic cello is function of the average of the domains sizes.
• A sixteenths trill on the second melodic line represents a new variable.
• A new constraint is described by a descending arpeggio on the second melodic line.
• A constraint solving causes the apparition of a rising arpeggio on the second melodic line.
• Variable domains descriptions are responsible of the first melodic line, in the very same way than the lead guitar in the Hard-Rock style.
• The intervals taken away from a variable domain during a reduction are represented by two eighth notes whose pitch is function of the interval bounds values.

What we immediately hear – and it was the main goal of this style – is the average evolution of the variable domains, due to the rhythmic cello score. Trills and arpeggios caused by new variable, new constraint, and solved constraint events are quite recognizable when only the second melodic cello plays, but become less pinpointable when the two melodies play together.

Depending on the number of variables the CSP contains, the two melodic lines will be more or less dissociated. In all cases, we can determine that the music is describing a general state. when the melodic cello plays without interrupting whereas the second one remains silent. Reduction causes, on the contrary, a mixing between the two melodies (the first describes the domains, and the second the values taken away). Taken alone, the second melodic line is composed of repeating patterns (arpeggios and trills) alternating with more free sequences (the description of the values taken away from the domains). This proves that it is possible to build quite interesting melodies by using several predefined patterns rather than uniques notes. The pieces of music produced by this style do not have a good balance in many Midi synthesizers (the rhythmic cello is in general not loud enough), and need manual adjustment.

4 Discussion

4.1 Creating your own style

It is necessary to take into account the quantity and disposition of the information contained in each trace events in order to associate them properly to musical events. In fact, although it is always possible to associate anything to anything, it is a better strategy to associate trace events to musical events having about the same layout in a piece of music: for example, a melody can hardly be created only from the backtracking events, as these events are way too sparse to provide enough information to build a melody. In order to achieve this, a little study of this trace events layout in the execution trace is needed. Of course what follows are only advices coming from our experience.

We must also take into account the amount of information each event gives. For example, an event which keeps repeating all along the trace could be used to build the main melody (theme) of the music, which consists of a regular stream of notes. However, if this trace event has no attribute, it cannot provide sufficient essential information to properly build a melody, like the pitch of each notes. Hence such an association would not be appropriate. For the simplicity of exposition, let us consider in what follows that each attribute can be represented as an integer that can be used as a musical event parameter.

Variable domains description. A variable domain is described by several \texttt{<range>} blocks included in a \texttt{<variable> block}, itself included in a \texttt{<variable>} or a \texttt{<reduce>} block. Hence, we have for each variable domain at least six integers defined (variable identifier, min and max of the whole domain, domain size, min and max of each range), with strong relations between them: the min and max of the variable domain enclose every other available integer, which are contained in the variable domain, except the vari-

\footnote{Purely synthetic sound present in the General MIDI instrument base.}
able identifier.

Variable domains descriptions are the most numerous and recurring trace events. Furthermore, the large amount of information each one provides makes them a natural candidate for building the main theme. Generating a melody with this event can be done using various methods (the different styles CSP Singer provides are using two of them). We find a huge set of these events in each general state description (in a `<state>` block), where each variable domain is described, plus other sparser ones, included in `<reduce>` and `<new-variable>` blocks.

**Reductions.** The reduction of a variable domain is described in a `<reduce>` block which contains the list of deleted values, as well as a domain description of the reduced variable. Hence we have at least four integers (variable and constraint identifier, and variable domain bounds), plus each bounds of all the intervals which have been taken out from the domain. These integers are of course also strongly related, as the variable domain and the set of deleted values are disjoint, and are both included in the previous variable domain (before reduction).

At the beginning of the trace, reduction occurs usually rarely, and sometimes not at all the beginning of the trace. Then, the density of this event progressively rise as we go deeper in the search tree. Thus, they are numerous enough to generate an accomplishment score, which needs a continuous flow of information to avoid repetitivity, but less than for a melody. It is also possible to generate a second melody using domain descriptions and deleted values. However, this melodic line is likely to contain silences (for example when a general state is described) if every other variable domain descriptions are used to generate other musical events.

**Choice points and backtracking.** These events are the only ones which can modify the depth of the search tree. They are respectively described by a `<choice-point>` and a `<back-to>` block. Both contain a general state description (`<state>` block). Consequently, they can be seen as precise and instantaneous events if we consider the state description as a different event, or as huge events, constituting the majority of the execution trace. Apart from the general state description, these events provide a couple of integers (node identifier, node-before identifier, and new current depth in the search tree).

If we consider them as instantaneous events (i.e., we consider the `<state>` block as a different event), they are quite rare, but well distributed all over the trace. This makes them ideal to be associated with global music events (affecting all the musical score at the same time), like tempo or key signature modification.

**New variable and constraint declaration.** These last events do not come with much information: only one integer, representing the identifier of the new object. The new variable also come with an initial domain, consisting generally in one interval. Since the identifiers are unique in the trace, these events are different from all other ones.

The quantity and layout of these events can change a lot according to the initial CSP. However, we can generally observe that a great majority of the new variables, including all the user variables, are declared at the very beginning of the trace. Then, internal variables can be declared everywhere in the trace. New constraints come with the same layout: they can be massively found at the beginning of the trace, especially if some global constraints are used (like the Alldifferent constraint, splitted into many basic constraints by GNU-Prolog). They become then sparser as we progress in the trace. It might be possible to make a melody with these events, but it is most likely to be very irregular, and contain very long silences. It is also possible to use them to change the way an accompaniment plays.

### 4.2 CSP Singer viewed from different angles

**For a composer.** The different available styles shows that it is possible, using different association sets, to create a wide variety of music. It would also be possible, using music formalizations other than classical harmony, to define many other musical events to be associated with trace events (especially in jazz music, which is virtually impossible to describe using only classical harmony).

Moreover, the tree-like structure of the execution trace tends to be reproduced by the generated pieces of music. It gives to these pieces an uncommon, but nevertheless pleasant general structure. It should be mentioned that, as the variable domains go only through slight modifications between two consecutive descriptions, the corresponding musical events are repetitive, but slightly modified at each “iteration”, and can lead to some sort of progressive music. Lastly, there are as much execution traces as CSPs, and these traces are very different from many angles. So, the raw material necessary to generate music using this way is quite infinite.

**For a CSP programmer.** One of the initial question besides composition was to investigate the possibility of extracting useful informations on a CSP solving process by listening to the music produced its trace. It would have been necessary, in order to fully answer this question, to have a quite large number users over a long enough period of time, to enable them to properly assimilate the different available musical styles, which had been impossible to set up. However, experimenting with CSP Singer during its development allows us to have a rough idea of naturally hearable events:

- The general structure of the search tree is well represented in all the generated music pieces. We can hear
where we are in the search tree.

• The melody brings us some information about the domains bounds, and so, we can hear where the most important reductions take place. Arc-consistency is also clearly heared in some styles (especially the “Hard Rock” one).

Of course, this list is far from being exhaustive, as auditive reflexes acquisition enables to identify events more and more rapidly and accurately but requires a regular practice of the CSP Singer software. Only a complete study, over a long time and with a large number of programmers, would enable to identify precisely the most important pieces of information needed for debugging, and to concentrate on the ones which are less visible with a purely visual debugger (like Pavot (Arnaud 2003), which uses the same trace format as CSP Singer). The large amount of information we can listen with the music is also encouraging, however it is necessary to do more study to exhibit the limits of this way of debugging. One interesting possibility is to associate CSP Singer to a visual debugger, in order to get visual and auditive information at the same time.

5 Conclusion

Music generation using a CSP trace proves to be very interesting from a musician point of view. Through associations between trace events and musical events cast in different styles, the execution of the resolution of a Constraint Satisfaction Problem is able to compose interesting music pieces with a good balance between variety and structure. Only a glimpse of the possibilities offered by this method have been detailed here. CSP Singer can be largely improved, particularly by including a more complete set of classical harmony notions, and notions from other music formalism (like the jazz or contemporary ones). Another improvement of CSP Singer would be to enable the user to completely describe his own associations through a language, which would extend the software possibilities to the user’s imagination.

Acknowledgements. We would like to thank Yann Lefevre for his precious help in musical theory and the Hard-Rock style drum patterns composition, as well as Charlotte Truchet for her ideas of associations between musical and trace events.

References