Learning Regular Grammars to Model Musical Style: Comparing Different Coding Schemes.[*]

Pedro P. Cruz-Alcázar[1], Enrique Vidal-Ruiz[2]

[1] EPSA, Universidad Politécnica de Valencia, DISCA-Alcoy, Alicante, Spain
pcruz@iti.upv.es
[2] Universidad Politécnica de Valencia, DSIC, Valencia, Spain
evidal@iti.upv.es

Abstract. An application of Grammatical Inference (GI) in the field of Music Processing is presented, were Regular Grammars are used for modeling musical style. The interest in modeling musical style resides in the use of these models in applications, such as Automatic Composition and Automatic Musical Style Recognition. We have studied three GI Algorithms, which have been previously applied successfully in other fields. In this work, these algorithms have been used to learn a stochastic grammar for each of three different musical styles from examples of melodies. Then, each of the learned grammars was used to stochastically synthesize new melodies (Composition) or to classify test melodies (Style Recognition). Our previous studies in this field showed the need of a proper music coding scheme. Different coding schemes are presented and compared according to results in Composition and Style Recognition. Results from previous studies have been improved.

1 Introduction

Grammatical Inference (GI) aims at learning models of languages from examples of sentences of these languages. *Sentences* can be any structured composition of primitive elements or symbols, though the most common type of composition is the *concatenation*. From this point of view, GI find applications in all those many areas in which the objets or processes of interest can be adequately represented as strings of symbols. Perhaps the most conventional application areas are Syntactic Pattern Recognition [9] and Language Modeling [18]. But there are many other areas in which GI can lead to interesting applications. One of these areas is *Music Processing.* Here, the very notion of language explicitly holds, where primitive symbols or *"notes"* are adequate descriptions of the acoustic space, and the concatenation of these symbols leads to strings that represent musical sentences.

Classical discretization/idealization of the acoustic space in music, entails

[*] This work has been partially supported by European Union ESPRIT LTR Project 30268 "EUTRANS".

appropriate descriptions of fundamental frequency or *pitch* of sounds and their temporal span or *duration*. These descriptions are called *"scores"*. Other features of sound, such as *timbre* and *tempo* are generally considered less essential for describing music and are often specified just as "side annotations" to the main score. The choice of a discretized, symbolic representation of pitch and duration constitutes the very essence of a musical system. Many different settings have been employed in different cultures, but modern occidental music has adopted the so-called *tempered scale,* which contains 12 symbols to represent different pitches within one *"octave"* (the range of pitch from a frequency *f* to *2f*). The "minimal distance" between pitches is called *"halftone"*. On the other hand, the choice of a symbolic scheme to represent duration may also depend of each musical system and, in modern occidental music this leads to symbols such as *whole note, half note, quarter note,* etc. where each element describes an event whose duration is half that of the previous symbol in the scale. In order to use Grammatical Inference in Music Processing (MP), a proper method to convert scores into symbol strings must be found. In this paper different coding schemes for music are proposed and compared.

By adequately concatenating symbols of a given musical system, a musical event emerges. However, not any possible concatenation can be considered a "proper" event. Certain rules dictate what can or can not be considered an appropriate concatenation, leading to the concept of *musical style.* Main features of a musical style are *rhythm* and *melody*, which are directly related with the rules used to concatenate duration and pitch of sounds, respectively. For the sake of conciseness, in what follows, we will use just the term *"melody"* for the combination of these two features. Our aim is to find these rules for modeling a musical style by means of GI. The interest in modeling musical style resides in the use of these models in MP applications, such as *Automatic Composition* and *Automatic Musical Style Recognition*, which are the areas of our interest. Since the creation of the first computers, many experiments have been carried out on how to use them advantageously in music composition [12], obtaining results that go from anecdotic to true masterpieces. Many Artificial Intelligence techniques have been used in MP [15], being Expert Systems very popular over the past years. Very interesting results have been obtained, but these methods have a strong limitation: it is very difficult to code musical knowledge as a set of rules. This limitation restrains the action field to simple and well known musical styles, such as Bach's chorales and standard Jazz. The use of an inductive method such as GI avoids this restrictions, because musical style is learned without human participation. We have focused our work in music composition in generating pieces in a determined musical style, such as Renaissance, Baroque, etc. This leads to the need of a model for musical style. On the other hand, the field of *Automatic Musical Style Recognition* does not seem to have been studied so much. It deals with giving certain skills to a computer so that it will be able to recognize the musical style to which supplied melodies belong. It is still a field to explore and applications can be found in musicology, music teaching, etc. These areas of MP were studied using GI in a previous work [4] [5] [6], that has been extended in the work presented in this paper.

2 Grammatical Inference Algorithms

In this section, we concisely explain the three algorithms used in our study to infer the grammars employed for *composing* and *recognizing* musical styles. These algorithms are fairly well known in the GI community and have been proven useful in other fields.

2.1 The Error-Correcting Grammatical Inference (ECGI) Algorithm

ECGI is a GI heuristic that was explicitly designed to capture relevant regularities of concatenation and length exhibited by substructures of unidimensional patterns. It was proposed in [13] and relies on error-correcting parsing to build up a stochastic regular grammar through a single incremental pass over a positive training set. Let R_+ be this training sequence of strings. Initially, a trivial grammar is built from the first string of R_+. Then, for every new string that cannot be parsed with the current grammar, a standard *error-correcting* scheme is adopted to determine a string in the language of the current grammar that is error-correcting closest to the input string. This is achieved through a Viterbi-like, error-correcting parsing procedure [8] which also yields the corresponding optimal sequence of both non-error and error rules used in the parse. From these results, the current grammar is updated by adding a number of rules and (non-terminals) that permit the new string, along with other adequate generalizations, to be accepted. Similarly, the parsing results are also used to update frequency counts from which probabilities of both non-error and error rules are estimated.

2.2 The k-TSI Algorithm

This algorithm belongs to a family of techniques which explicitly attempt to learn which symbols or substrings tend to follow others in the target language. It is a "characterizable" method that infers *k-Testable Languages in the Strict Sense* (k-TSSL) in the limit. A k-TSSL is usually defined by means of a four-touple $Z_k = (\Sigma,I,F,T)$ where Σ is the *alphabet*, *I* is a set of *initial substrings* of length smaller than *k*; *F* is a set of *final substrings* of length smaller than *k* and *T* is a set of *forbidden substrings* of length *k*. A language associated with Z_k is defined by the following regular expression: $\mathcal{L}(Z_k)=I\Sigma^*\cap\Sigma^*F-\Sigma^*T\Sigma^*$. In other words, $\mathcal{L}(Z_k)$ consists of strings that begin with substrings in *I*, end with substrings in *F* and do not contain any substring in *T*. Stochastic k-TSSL are obtained by associating probabilities to the rules of the grammars, and it has been demonstrated that they are equivalent to N-GRAM's with N=k [16]. The inference of k-TSSLs was discussed in [11] where the k-TSI algorithm was proposed. This algorithm consists in building the sets Σ,I,F,T by observation of the corresponding events in the training strings. From these sets, a Finite-State automaton that recognizes the associated language is straightforwardly built.

2.3 The ALERGIA Algorithm

ALERGIA infers stochastic regular grammars in the limit by means of a State Merging Technique based on probabilistic criteria [3]. It builds the prefix tree acceptor (PTA) from the sample and, at every node, estimates the probabilities of the transitions associated to the node. The PTA is the canonical automaton for the training data (R+) and only recognizes L(R+). Next, ALERGIA tries to merge pairs of nodes, following a well-defined order. Merging is performed, if the tails of the states have the same probabilities within statistical uncertainties. The process ends when further merging is not possible. By merging states in the PTA the size of the recognized language is increased, and so, training samples are generalized. In order to cope with common statistical fluctuations of experimental data, state-equivalence is accepted within a confidence interval calculated from a number between 0 and 1, which is introduced by the user, and which is called accuracy (a). This parameter will be the responsible for the higher or lower generalization of the language recognized by the resulting automaton.

2.4 The Synthesis Algorithm

In this work, automata will be inferred for different musical styles (languages). “Composing” melodies in specified style amounts to synthesizing strings (melodies) from the automaton that represents the desired style. To achieve this, we use an algorithm that performs random string generation from a stochastic automaton. It randomly follows the transitions of the automaton from the initial state to some final state, according to the probabilities associated to each transition. In order to listen to the synthesized strings, we developed a program to convert them into MIDI files [14], so they can be listened to with any MIDI file player.

2.5 The Analysis Algorithm

As each inferred automaton represents a musical style, “recognizing” the style of a test melody consists in finding the automaton which best recognizes this melody. This can be best achieved by using an algorithm that performs stochastic Error-Correcting Syntactic Analysis through an extension of the Viterbi algorithm [8]. The probabilities of error rules (Insertion / Deletion / Substitution) can be estimated from data [1] [2]. The Analysis Algorithm returns the probability that the analyzed string (melody) is (error-correcting) generated by the automaton. By analyzing the same melody with different automata, we classified it as belonging to the musical style (language) represented by the automaton that gave the greatest probability.

3 How to Code Music. Different Coding Schemes

As it was mentioned in the introduction, GI algorithms work with symbol strings, so it is necessary to code music in the same way. In this section we present four different coding schemes for music that have been used in our experiments. These coding schemes were developed along with the experiments in order to improve the results. They are conversions from the traditional musical notation into simple codes that preserve the main attributes of music and are formed of symbols.

The first coding scheme we proposed [4] was called *implicit notation* because each note "length", or duration, is implicit in the coding as in a musical score. As every *melody* is characterized by the *pitch* and *length* of the sounds that compose it, we coded only these two attributes. Each note is coded as a symbol or word composed of three "fields", one to code the pitch and two for the length. A space character is used between symbols to separate them. To code the *pitch*, a number was assigned to each note of the adopted scale, beginning with the deepest sound, corresponding to number 1, and ascending progressively halftone by halftone to the highest sound, corresponding to number 43. If the note is a *rest* (a period of time with no sound), the pitch will be coded with number 0. The *note length* was coded using one or two characters after the note number. The second character is optional and we call it "*extra_length*". Thus, a note in implicit notation is coded as follows:

Note number + length + extra_length.

The way of coding the field *length* can be seen in Fig. 1 (left side). The field *extra_length* contains a symbol that modifies the length given by the previous field, normally increasing it. We only needed the symbols described in Fig. 1 (right side) to code the training samples, but there are more. *Tied notes* were coded with dots and it was necessary to introduce the *half dot* (which does not exist in music notation) to code notes tied with notes whose length is a quarter of their value, like a *half note* tied to an *eighth note*. An example of implicit notation of a score can be seen in Fig. 2. The idea of employing a discrete representation of the pitches and lengths has been used before by other authors, like Eberhart [7] and Todd [17] in their experiments in music composition with Neural Networks. It is a direct translation from traditional musical notation into symbol strings.

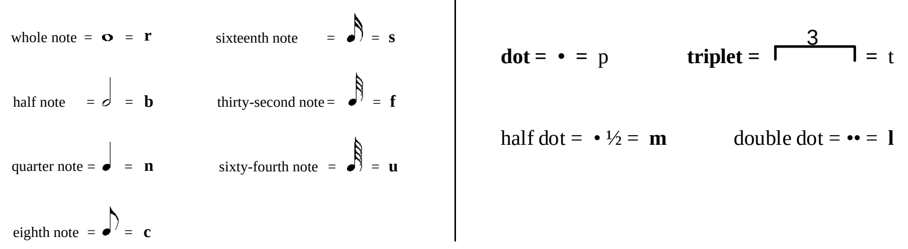
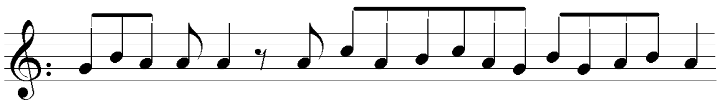


Fig. 1. How to code the length of notes in implicit notation (left side). How to code the "extra_length" field (right side)

After performing Automatic Composition experiments with implicit notation many results presented a non-desired effect, which we called the "*tonality problem*". The term *tonality* alludes to the set of relationships established between the sounds of a scale and the first note from this scale, which is called *tonic*. These relationships make us to consider the sounds of the scale organized by reference to a tonic sound or *fundamental note,* having the other notes from the scale a tendency towards it. Therefore, *tonal music* is the music written inside a *tonal system*, that is, the one that it has a tonal center or fundamental note.

Every musical style modeled in this work belongs to tonal music and the *tonality problem* was present when GI algorithms tried to compose in each of them. This problem can be described as follows. Since training samples have different tonalities and even can "modulate" (change the tonality) several times within the same sample, this increases the diversity of the samples and tends to "confuse" GI algorithms. So, in many compositions, modulations were observed to be incoherent in the musical style they should belong to. One of the solutions we proposed and adopted was changing the coding scheme in this way: instead of coding the absolute pitch of the notes, the *intervals* (distance in halftones) between notes, that is, the relative pitch from one note to the previous one, are coded. This coding scheme was called *differential notation* and allows us to abstract from the pitch of the notes and so, from the tonality. Given the nature of the tempered scale, the way to obtain this notation is just by subtracting the number that represents the previous note's pitch in implicit notation from the number that represents present note's pitch, leaving the length of notes coded as in implicit notation. Obviously, the first note from a score has not any note for subtracting its pitch, so its pitch will be coded with a special character 'S'. If a note to code is a *rest*, the pitch will be coded with number 99 because being a note without pitch (no sound) it can not be subtracted to next note. We do not maintain the number 0 for rests because in this notation it means that there are two notes with the same pitch. An example of this notation can be seen in Fig. 2. This idea of employing a differential representation of the pitches was also used by Todd [17].

In the two previous coding schemes, each note from the score is represented by only one symbol, that is to say, a string of characters or 'word' that globally represents both the pitch and length from the note. After performing the experiments with differential notation, it was considered that these coding schemes established a relationship between the pitch and length stronger than the one generally existing in music. So, it was decided to modify the previous notations by coding the pitch and length as separated words. These two new coding schemes were called *splitted implicit notation* and *splitted differential notation*. Results with them improved noticeably as can be seen in next section. An example of these coding schemes is shown in Fig. 2.



Implicit Notation:	22c	26c	24c	24c	24n	0c	24c	27c	24c	26c	27c	24c	22c	26c	22c	24c	26c	24n
Differential Notation:	Sc	4c	-2c	0c	0n	99c	0c	3c	-3c	2c	1c	-3c	-2c	4c	-4c	2c	2c	-2n
Splitted Implicit Notation:	22 c	26 c	24 c	24 c	24 n	0 c	24 c	27 c	24 c	26 c	27 c	24 c	22 c	26 c	22 c	24 c	26 c	24 n
Splitted Differential Notation:	S c	4 c	-2 c	0 c	0 n	99 c	0 c	3 c	-3 c	2 c	1 c	-3 c	-2 c	4 c	-4 c	2 c	2 c	-2 n

Fig. 2. A Gregorian style score coded with the four notations presented.

4 Experiments

For the present study, we chose 3 occidental musical styles from different epochs and we took 100 sample melodies from each one. These samples were 10 to 40 seconds long. The first style was *Gregorian* (Middle Ages). As a second style, we used passages from the sacred music of *J. S. Bach* (Baroque). The third style consisted of passages from *Scott Joplin* Ragtimes for piano (beginning of 20th cent.). The results in Musical Style Recognition are expected to be an indicative of the goodness of the musical style's models. So, if a model has been good for *Automatic Musical Style Recognition* it can be supposed that it will be good for *Automatic Composition*. This goodness will be further evaluated through more direct subjective listening tests with melodies automatically composed by the learned models.

4.1 Automatic Musical Style Recognition Experiments

We inferred three automata (one per style) with each GI algorithm, trying different values of *k* (with k-TSI) and *a* (with ALERGIA). Test melodies are analyzed to see which of the learned automaton can generate them with the greatest probability. Given the small size of the available corpus, *Cross-Validation* was used to measure the recognition accuracy of the different techniques. In each experiment, we took 90 melodies for training and 10 for test. Then the role of the melodies was shifted in such a way that, at the end, every melody was used for both training and testing. A *Confusion Matrix* was built with the results of each cross-validation partition. Its rows and columns denote the style of test melodies and the style in which they were classified, respectively. Then we obtained an *Average Confusion Matrix* that summarized the overall cross-validation results. The diagonal of the matrix represents the *Average Success Rate* in recognition. We also obtained the *Average Classifying Error* [4]. For the sake of simplicity, results will be summarized in tables representing the *average success rate* and *classification error*.

Results. In our previous work [4], experiments were performed using only *implicit notation* and tables 1-3 present the best results for each GI algorithm. Results using ECGI were good for Gregorian, but not for Bach and Joplin's styles, giving a high *average classifying error* of the 35.9 %. With k-TSI, the results were better and the best were obtained with *k*=3. K-TSI was the algorithm that best worked in style recognition with this coding scheme, achieving an average classification error of the 9.96 %. The ALERGIA results were similar to k-TSI. We tried different *a* values covering its entire range ([0,1]) and the best results were achieved with *a*=0.01, obtaining an average classification error of the 13.3 %. It is worth noting that every algorithm obtained a 100% success rate when classifying Gregorian melodies. This will be a constant in almost every performed experiment. Clearly, Gregorian style is less complex than the others, and it is easy for the algorithms to discriminate Gregorian melodies from the rest.

Tables 1-3 also show the results for each GI algorithm using the different here proposed coding schemes. In order to summarize, only the best will be presented. Table 1 presents the results obtained with the ECGI algorithm. It can be seen that the use of *differential notation* leads to a great reduction in the classifying error (more than 50 % as compared with *implicit notation*). The use of *splitted differential notation* reduces error more than 75 %. Using *splitted implicit notation*, the classifying error is reduced drastically too, but not as much as with *splitted differential notation*. The great reduction of the classifying error using *differential notation* is specific for the ECGI algorithm as can be seen in table 1.

The k-TSI algorithm, results were much better than with ECGI for every coding scheme used. The best results were obtained with different *k* values depending on the coding scheme employed (Table 2). The “splitted” notations needed a greater value for k because now the number of symbols needed for coding a music sample is doubled. The most important error reduction is obtained with the *splitted notations*, being over 50% with respect to *implicit notation*. The best result is obtained with the *splitted differential notation* and it is the best one achieved in this work for Musical Style Recognition. It was reached a 5% classification error and close to 100% average success rate for melodies form Bach’s style. ALERGIA was the only algorithm whose results got slightly worse using *differential notation*. However, as can be noticed in Table 3, these were improved using the *splitted* notations, but not as much as with the previous algorithms. The best results were obtained with different *accuracy* (*a*) values depending on the coding scheme employed and, as with the other algorithms, the best one for our purpose was the *splitted differential notation*.

Clearly, with every algorithm the fact of separating the pitch and length from notes has been more important than changing the way of coding pitches to a *differential code*. This confirms our assumption that *implicit notation* established a relationship between the pitch and length stronger than the one generally existing in music. The best result was obtained with k-TSI algorithm, and it is worth noting that results with this coding scheme are quite good considering the small number of samples available.

	Implicit Notation	*Differential Notation*	*Splitted Implicit Notation*	*Splitted Differential Notation*
GREGORIAN	100 %	100 %	100 %	**100 %**
BACH	40 %	75 %	82 %	**89 %**
JOPLIN	52 %	79 %	**88 %**	86 %
Classif. Error	35.9 %	15.3 %	11.3 %	**8.3 %**

Table 1. Results for ECGI algorithm with the different coding schemes.

	Implicit Notation (k=3)	*Differential Notation (k=3)*	*Splitted Implicit Notation (k=5)*	*Splitted Differential Notation (k=4)*
GREGORIAN	100%	100%	100 %	**100%**
BACH	85%	88 %	94 %	**96 %**
JOPLIN	85%	86 %	89 %	**89 %**
Classif. Error	9.96%	8.6 %	5.6 %	**5 %**

Table 2. Results for k-TSI algorithm with the different coding schemes.

	Implicit Notation (a=0.01)	*Differential Notation (a=0.2)*	*Splitted Implicit Notation (a=0.009)*	*Splitted Differential Notation (a=0.009)*
GREGORIAN	100 %	100 %	100 %	**100 %**
BACH	80 %	85 %	82 %	**86 %**
JOPLIN	80 %	69 %	**86 %**	83 %
Classif. Error	13.3 %	15.3 %	10.6 %	**10.3 %**

Table 3. Results for ALERGIA algorithm with the different coding schemes.

4.2 Automatic Composition Experiments

Here, the procedure is similar to that used in Automatic Musical Style Recognition. The difference is that, once the automata are obtained, new melodies are randomly synthesized with the aim of obtaining good synthesis results. For a preliminary evaluation, the following criteria were adopted:

1.- The synthesis must make musical sense and must not be a random series of sounds.
2.- It must sound similar to the Musical Style that it is meant to (Gregorian, Bach, etc.).
3.- It must have the characteristics of a proper melody (beginning, development, and end).
4.- It must be different from any of the training melodies.

In order to assess the quality of the synthesis, a secondary evaluation was carried out by the first author following his own (subjective) musical criteria. This quality evaluation was made according to the following classification: very good, good, not very good, bad. A synthesis was considered *bad*, if it did not satisfy all of the above four criteria. If it did satisfy them all, then it was classified according to quality level of the composition as *very good*, *good* or *not very good*. We consider a synthesis as *very good* when it can be taken as an original piece from the current style without being a copy or containing evident fragments from samples, and the remaining qualifying adjectives explain themselves.

Results. In our previous work [4], experiments were performed using only the *implicit notation* coding scheme and the best results were obtained with ECGI and K-TSI (with *k*=3,4) algorithms. These results ranged from *good* to *very good* for Gregorian and from *bad* to *not very* good for the styles of Bach and Joplin. ALERGIA's results were not so good, so it was decided not to use it in future studies unless a larger number of samples were available. Many compositions synthesized using *implicit notation* showed what we called the *"tonality" problem*, which was explained in section 3. For solving this problem alternative coding schemes were proposed. To this end, only *differential notation* has been tested so far, leaving the rest of coding schemes for future studies.

With the ECGI algorithm, *differential notation* results in Bach's and Joplin's styles are improved, increasing the number of *good* syntheses, but not obtaining still any *very good* one. An example of ECGI compositions in Gregorian style using *differential notation* can be seen in Fig. 3. With the K-TSI algorithm, results have been enhanced in every style, specially in Gregorian style, but still *very good* melodies are not obtained for the other styles (in the opinion of the authors). Best *k* values were, as with *implicit notation*, *k*= 3 and *k*=4. . The best results in *Automatic Composition* have been obtained with this algorithm, like in *Automatic Musical Style Recognition.*

As a whole, results have been improved in every experiment, but not as much as we expected. The *tonality problem* has been partially solved, because now the synthesized melodies have more continuity (tonally speaking) but another not desired effect derived from the coding has appeared: if two or more musical fragments that follow a descending/ascending scale are concatenated while creating a synthesis, as the pitch encoding is always relative to the previous note, the synthesis tends to exceed the common *tessitura* (range of pitches) in the corresponding style. This effect was observed also by Todd [17], but in his experiments it was more negative and he discarded this notation. In ours, the improvement of the results has been better than the occasional occurrences of this effect.

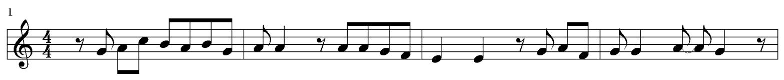


Fig. 3. Example of ECGI composition in *Gregorian* style using *differential notation.*

5 Conclusions and Future Trends

Two applications of Grammatical Inference to Music Processing have been presented. These *are Automatic Composition* and *Automatic Musical Style Recognition*. Different coding schemes for music have been proposed and compared according to results in Style Recognition. Preliminary Composition results with one of the newer coding schemes are also presented. Results from our previous studies in this areas have been improved, showing the need of proper music coding schemes in order to take the greatest advantage of the GI algorithms. The best results have been obtained with the k-TSI algorithm both in *Automatic Musical Style Recognition* and in *Automatic Composition.* Limitations of space and time in this paper have prevented us from discussing many aspects in a deeper way, and future trends for this work are presented next.

Several lines of study can be followed to attempt improving results. Obviously, experiments must be done with *splitted* notations in Automatic Composition, and a better protocol for the analysis of automatic compositions should be found (establishing more rules, analysis by a set of musicians, etc.). Additionally, the following lines of study are proposed. First, the amount of data used so far is insufficient and better performance is expected by increasing the number of training samples. Second, since training samples have different tonalities, this increases the diversity of the samples and tends to "confuse" GI algorithms. This was called the *tonality problem* in our previous work and it became quite apparent in Automatic Composition when using *implicit notation*. This problem has been partially solved using *differential notation*, but still has to be fixed. For this purpose one can take advantage of the flexibility offered by the MGGI methodology [10] [19] to explicitly impose tonality constraints in the learning procedure, by adequately relabeling of symbols in the training strings. Once these problems are solved, we could employ entire musical pieces as samples, and not just small fragments as was done in this study. Future studies could also be expanded to include *polyphony*. So far, GI algorithms compose *monodies*; that is, melodies without accompaniment. By changing the GI algorithms in such a way that they could accept samples formed by several strings, it might be possible to deal with samples with more than one voice. This is not a trivial task, and an easier way is to apply automatic harmonizing algorithms to the GI compositions.

References

1. Amengual J.C., Vidal E. and Benedí J.M. October 1996. Simplifying Language through Error-Correcting Decoding. Proceedings of the ICSLP96 (IV International Conference on Spoken Language Processing), pp. 841--844, Philadelphia, PA., USA, 3-6.
2. Amengual J.C. and Vidal E. 1996. Two Different Approaches for Cost-efficient Viterbi Parsing with Error Correction. Advances in Structural and Syntactic Pattern Recognition, pp. 30-39. P. Perner, P. Wang and A. Rosenfeld (eds.). LNCS 1121. Springer-Verlag.
3. Carrasco, R.C.; Oncina, J. 1994. Learning Stochastic Regular Grammars by means of a State Merging Method. "Grammatical Inference and Applications". Carrasco, R.C.; Oncina, J. eds. Springer-Verlag, (Lecture notes in Artificial Intelligence (862)).
4. Cruz P.P. 1996. Estudio de diversos algoritmos de Inferencia Gramatical para el Reconocimiento Automático de Estilos Musicales y la Composición de melodías en dichos estilos. PFC. Facultad de Informática. Universidad Politécnica de Valencia.
5. Cruz P.P. 1997. A study of Grammatical Inference Algorithms in Automatic Music

Composition. Proceedings of the SNRFAI97 (VII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes). Vol. 1, pp. 43-48. Sanfeliu A., Villanueva J.J. and Vitrià J. Eds. Centre de Visió per Computador, Universidad Autónoma de Barcelona.

6. Cruz P.P. 1997. A study of Grammatical Inference Algorithms in Automatic Music Composition and Musical Style Recognition. Proceedings from the 'Workshop on Automata Induction, Grammatical Inference, and Language Acquisition', celebrated during the ICML97 (The Fourteenth International Conference on Machine Learning) Nashville, Tennessee. Electronic publication in the Workshop Web page (http://www.cs.cmu.edu/~pdupont/mlworkshop.html).
7. Eberhart, R.C.; Dobbins, R.W. 1990. Neural Network PC Tools. Academic Press Inc, pp. 295-312.
8. Forney, G. D. 1973. The Viterbi algorithm. IEEE Proc. 3, pp. 268-278.
9. Fu, K.S. 1982. Syntactic Pattern Recognition and Applications. Prentice Hall.
10. García P., Vidal E., Casacuberta F. 1987. Local Languages, the Successor Method, and a step towards a general methodology for the Inference of Regular Grammars. IEEE Trans. on Pattern Analysis and Machine Intelligence. Vol.PAMI-9, No.6, pp.841-844.
11. García, P.; Vidal, E 1990. Inference of K-Testable Languages In the Strict Sense and Application to Syntactic Pattern Recognition. IEEE Trans. on PAMI, 12, 9, pp. 920-925.
12. Nuñez, A. 1992. Informática y Electrónica Musical. Ed. Paraninfo.
13. Rulot, H.; Vidal, E. 1987. Modelling (sub)string-length based constraints through a Gramatical Inference method. NATO ASI Series, Vol. F30 Pattern Recognition Theory and Applications, pp. 451-459. Springer-Verlag.
14. Rumsey, F. 1994. *MIDI Systems & Control*. Ed. Focal Press.
15. Schwanauer S.M.; Levitt D.A. 1993. Machine Models of Music. The MIT Press.
16. Segarra, E. 1993. Una Aproximación Inductiva a la Comprensión del Discurso Continuo. Facultad de Informática. Universidad Politécnica de Valencia.
17. Todd P. 1989. A sequential network design for musical applications. Proc. of the 1988 Connectionist Models Summer School. Morgan Kaufmann Publishers, pp. 76-84.
18. Vidal E., Casacuberta F., García P. 1995. Grammatical Inference and Automatic Speech Recognition. In "Speech Recognition and Coding: New Advances and Trends", A.Rubio y J.M.López, Eds., Springer Verlag.
19. Vidal E., Llorens D. 1996. Using knowledge to improve N-Gram Language Modelling through the MGGI methodology. In 'Grammatical Inference: Learning Syntax from Sentences'. Proc. of 3rd ICGI. L.Miclet, C. de la Higuera (Eds.). Springer-Verlag (Lect. Notes in Artificial Intelligence, Vol.1147).

The authors wish to thank the anonymous reviewers for their proposals for the improvement of this paper.