PAC Learning of Interleaved Melodies

(revised)

Brian J. Ross*
Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
email: bross@sandbanks.cosc.brocku.ca

Abstract

A number of algebraic models of music that use the interleaving or shuffle operator have been suggested in the literature. This paper shows how interleaving expressions are conducive to PAC identification. PAC learning theory states that a "probably approximately correct" hypothesis that falls within a desired level of accuracy can be derived with a high probability from randomly selected training examples. The relevance of this result is that it is possible to construct very efficient melody recognition and synthesis modules for interleaving music languages.

1 Introduction

This research investigates the application of formal machine learning techniques towards the automated recognition of music. A summary of the problem is as follows. We are given a well-defined formal language describing expressions that (for our purposes) generate musical events. Each expression can be considered to be an automata that recognizes or generates particular musical melodies. The formal language used in this research is one of a family of algebraic languages for modelling concurrency, and uses the interleaving or shuffle operator to model the nondeterministic interaction of concurrent musical agents. The learning problem is then: given a finite set of example behaviours, can we determine the interleaving expression that generates them? Previous research has shown that interleaving expressions can be exactly identified from positive examples in $O(n^2)$ polynomial time. This paper extends this result by showing that, if the interleaving languages are constrained by a finite hypothesis space, then "probably approximately correct" or PAC learning can be applied. The significance of a PAC result is that we can determine a "good enough" solution hypothesis from a set of randomly selected examples very efficiently.

Section 2 gives an overview of interleaving models of music, as well as some basic results from computational learning theory. Section 3 presents the main PAC result for a simple representative interleaving language for musical melodies. A discussion concludes the paper in section 4.

2 Background

2.1 Interleaving, Concurrency, and Music

Recently, with the growing interest in parallel computers, considerable research effort has been applied towards the formal modelling of concurrent computation. Mathematical models of concurrency include process algebra [Hoare, 1985; Milner, 1989], Petri nets [Peterson, 1977], and trace theory [Aalbersberg and Rozenberg, 1988]. All of these algebra use the interleaving or shuffle operator. Consider the interleaving expression,

$$abc \ || \ de \ || \ f.$$

Here, the terms *abc*, *de*, and *f* are considered to be concurrent agents that nondeterministically contribute their contents to the overall trace of the expression. Example generated streams for this expression include *abcdef*, *afbdec*, *fdeabc*, and so on. Note that the relative order of elements in each term is preserved in the traces; for example, the letters in *abc* retain their order in all traces produced. The algebraic power of interleaving arises from its ability to denote the combinatorially large number of traces resulting from the nondeterminism occurring amongst component terms [Mayer and Stockmeyer, 1992]. The above expression generates

$$\frac{6!}{3!2!1!} = 60$$

possible streams. The expression $\alpha_1 \ || \cdots || \ \alpha_k$ with $|\alpha_i| = 1$ generates $k!$ streams, which is greater than 2^k.

More complex interleaving languages use various types of communication, synchronization, nondeterminism, and iterative constructs, all of which enhance the descriptive power of the language. For example, in [Olderog and Hoare, 1986], the interleaving expression

$$abcd \ ||_b \ ebfg \ ||_b \ bhi$$

takes the event b to be a synchronizing event which must synchronize or handshake between all the terms in order to be generated. The net effect is that terms will effectively delay until they synchronize with one another.

*Support by NSERC Operating Grant 0138467 is gratefully acknowledged.

Another effect is that the number of possible traces is significantly reduced.

Music can be considered to be a concurrent activity, in which autonomous musical agents (eg. musicians) contribute to the overall musical production. As with the synchronizing expression above, such concurrent behaviour may require different sets of agents to synchronize at particular moments as well. This abstraction of music has been modelled with formal mathematical languages for concurrency, for example, Petri nets [Haus and Rodriguez, 1988; Haus and Sametti, 1992], trace algebra [Chemillier, 1987; Chemillier and Timis, 1988; Chemillier, 1990], and process algebra [Ross, 1995]. These various models have been applied towards musical analysis and composition. In all these models, letting the alphabet denote musical notes, interleaving is particularly adept at modelling a variety of styles of melody. These interleaving expressions then act as melody recognition or generating automata. Chemillier's model illustrates many elegant algebraic properties of basic interleaving when applied to music. Ross's interleaving algebra, enhanced with operators for nondeterministic probabilistic choice, iteration, and parameter passing, is used in stochastic composition.

2.2 Exact Identification of Interleaved Expressions

Given a particular interleaving language, can an interleaving expression be automatically derived from examples of its output behavior? In other words, if we have a representative subset of different traces we would like an expression to produce, can we automatically identify the expression that generates them? This problem is typical of those in computational learning theory or CLT [Angluin and Smith, 1983]. CLT is an active research area concerned with the automatic or semi-automatic identification of formal automata from examples of their behaviour. Generally speaking, many such problems are known to be NP-complete, while others become polynomial if the language is constrained, or additional information about the behavior (positive membership, negative membership, counterexamples,...) is given.

Imagine a learning scenario in which we have a universe of possible hypothesis expressions over some domain of interest. As we give our hypothesis selection algorithm more examples, the space of candidate hypotheses consistent with these examples reduces in size. Three situations eventually arise:

1. We have more than one consistent expressions left. This means that the hypothesis selection algorithm chooses one of them, which may or may not be consistent with our target behaviour.

2. We have exactly one expression left. It may or may not be equivalent to the target behaviour.

3. The set of consistent expressions is empty. There are no expressions that cover the example set.

When constructing a hypothesis in steps 1 and 2, it cannot be ensured that the hypothesis is equivalent to the desired target expression. Only an exhaustive supply of examples can guarantee that, which defeats the purpose in using the generalization that occurs with machine learning technology. Hopefully, the algorithm should generalize from the examples to determine an adequate hypothesis. Step 3 implies that there are no candidate solutions for this example set (which, by the way, does not occur with interleaving, since the degenerate expression can always be supplied if necessary).

The exact identification of some classes of interleaving expressions is tractable (an example is outlined in section 3.2). A subset of [Chemillier, 1987]'s interleaved music expressions can be exactly identified using a classical graph theoretic procedure [Kaufmann, 1968]. Similar ideas were used by Ross for finite and cyclic synchronized interleaving expressions [Ross, 1994a; 1994b]. Successful identification of interleaving expressions presumes that an ordering criteria over hypothesis expressions is used. For example, expressions with larger terms may be preferred over those with smaller ones. Without such an ordering, the interleaving expression $\alpha_1 \; || \cdots || \; \alpha_k$ with $\Sigma = \cup \alpha_i$ can always be chosen, as it covers all possible traces. The complexity of these identification algorithms is $O(n^2)$ for expressions using a fixed $|\Sigma| = n$. Identification is even more efficient if synchronization is included, as synchronization events greatly prune the trace space. When such interleaving expressions denote musical events, then these results mean that interleaving music "programs" can be automatically synthesized from positive training examples of desired melodies.

2.3 PAC Learning Theory

This paper considers the application of PAC learning of interleaved music expressions [Valiant, 1984; Haussler, 1988]. We first presume that the cardinality of all the possible events Σ is finite. This is not an unreasonable restriction, if one considers that members of this set are essentially musical events, and there are only a finite number of musical events of interest in a given composition. The purpose behind restricting the size of Σ is that we can then more easily restrict the number of hypothesis expressions H: a finite $|H|$ is necessary in order for PAC learning to be applicable. This restriction is not as constraining as it might appear, since many nontrivial musical behaviours can be generated from such languages. We treat the examples as being representative behaviours (tunes) exhibited by some desired target expression whose exact form is unknown.

The idea behind PAC learning is that, if the universe of candidate expressions is finite, then we are able to converge to an "acceptable" solution with a high probability using randomly selected examples. In addition, PAC identification typically occurs very efficiently - relatively few examples are necessary. While the candidate hypothesis selected might not be exactly equivalent to the target behavior, PAC theory gives a probabilistic guarantee that it will conform to a desired level of accuracy when a quantified number of randomly selected examples are used.

The following definitions and theorems from [Haussler, 1988] define PAC identification more rigorously.

Definition 1 *Consider the universe U of hypotheses* *derivable in some domain D, and the set of examples S of some target concept in D. The* version space *of S with respect to E is subset of hypotheses in U that are consistent with all the examples in S.* $\square$

Definition 2 *Consider the selection of examples from some example space that conforms to some probabilistic distribution. The* error *of a hypothesis with respect to a target concept relative to this distribution is the probability that the hypothesis and target concept disagree.* $\square$

Definition 3 *Given a hypothesis space H, a target concept E, a sequence of examples S of E, and an* error tolerance ϵ*, where $0 \leq \epsilon \leq 1$, the version space of S is ϵ-*exhausted *if it does not contain any hypothesis that has error more than ϵ with respect to E.* $\square$

Theorem 1 *Let H be a finite hypothesis space. Let Q be a sequence of m independent random examples of some target concept E in H drawn under some probability distribution on the example space. Then for any δ, $0 < \delta < 1$, if*

$$m \geq \epsilon^{-1}(ln(1/\delta) + ln|H|) \qquad [\mathbf{1}]$$

then the version space of Q is ϵ-exhausted with probability at least $1 - \delta$.
Proof: *See [Valiant, 1984; Haussler, 1988].* $\square$

The main result in the above is equation [**1**]. If we have a language with a finite hypothesis space of size $|H|$, and we have a mechanism for determining expressions from H that are consistent with supplied examples, then equation [**1**] determines the PAC identification characteristics for that language. Pragmatically speaking, given parameters ϵ and δ, we simply apply a natural log over the size of H to determine, in concert with the rest of [**1**], the number of randomly-selected examples to use. This logarithmic reduction typically means that very few examples are required in order to determine a good solution.

3 PAC Learning of a Simple Interleaving Language

We now show that an interleaved language $\mathcal{I}$ is PAC identifiable. The language $\mathcal{I}$ is simple, in order to simplify the mathematics. However, it contains the essence of the interleaving used in more complex music algebra.

3.1 Language Definition

The interleaving language $\mathcal{I}$ has the following syntax. We first presume a finite alphabet defined by a set Σ. A set is used in order to prevent repeating elements.

Definition 4

$$\mathcal{I} \;::=\; \alpha_1\alpha_2\cdots\alpha_k \quad | \quad E_1 \,||\, E_2$$

where $\alpha_i \in \Sigma$, $k \geq 0$, and $E_1, E_2 \in \mathcal{I}$. If $k = 0$ then the term is the empty string ϵ. $\square$

Example $\mathcal{I}$ expressions are abc, $ab \,||\, cd \,||\, e$, and ϵ, and their respective Σ are $\{a, b, c\}$, $\{a, b, c, d, e\}$, and $\emptyset$.

A trace is a string from Σ^* composed entirely of actions. The trace model for a given I expression is usually a set of strings, since its inherent nondeterminism permits the generation of multiple strings. In the following, the semantic function $\mathcal{T}$ generates a trace for the argument expression.

Definition 5 Trace semantics for $\mathcal{I}$:

$$\begin{array}{ll} \mathcal{T}(\epsilon) & = \{\epsilon\} \\ \mathcal{T}(\alpha E) & = \{\alpha t \mid t \in \mathcal{T}(E)\} \\ \mathcal{T}(E_1 \,||\, E_2) & = \{t \mid \exists t_1 \in E_1, t2 \in E_2 : t \in (t_1 \,||^*\, t_2)\} \\ where: & \\ \epsilon \,||^*\, \epsilon & = \{\epsilon\} \\ \alpha t \,||^*\, \epsilon = \epsilon \,||^*\, \alpha t & = \mathcal{T}(\alpha t) \\ \alpha s \,||^*\, \beta t = \beta t \,||^*\, \alpha s & = \alpha.(s \,||\, \beta t) \cup \beta.(\alpha s \,||\, t) \end{array}$$

$\square$

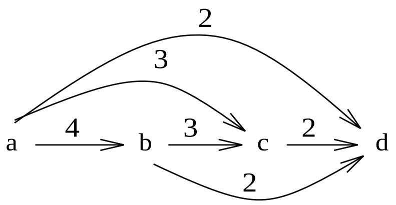
The expression abc has the trace set $\{abc\}$, while $ab \,||\, d$ generates the traces $\{abd, adb, dab\}$.

3.2 Exact identification of $\mathcal{I}$

$\mathcal{I}$ can be exactly identified from positive examples. Exact identification of interleaving only makes sense, however, if an ordering criterion is defined over hypothesis expressions. Otherwise, the degenerate expression $a \,||\, b \,||\, \cdots$ is trivially valid. A number of criteria are possible. The following presumes an ordering in which hypotheses are selected that contain *at least one maximally-sized term.*

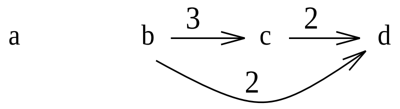
The key to the identification algorithm is the denotation of consistent terms via an acyclic digraph or DAG. Nodes in the graph denote elements from Σ, and arcs denote ordering of elements consistent with the example set. Arcs are also labelled with an integer indicating the maximal-sized path traversable from that arc.

The graph is initialized after the first example E_1 is read. The initial graph denotes all possible subtraces embedded within E_1. For example, if E_1 is $abcd$, then the initial graph is:



Various paths in this graph denote terms consistent with E_1. Arc lengths are initialized at this time as well, which is done directly from the position of elements within E_1.

When the next example E_2 is read, the inconsistent subtraces introduced by E_2 are determined. All the ordered pairs of elements within E_2 are extracted, and the pairs inconsistent with the ordering in the DAG are determined. When an inconsistency is found, that arc is deleted from the DAG. Deleting an inconsistent arc means that the elements joined by that arc cannot reside in the same term, since the those elements have inconsistent orderings within the examples. For example, if $E_2 = bcda$, then its pairs are $\{bc, bd, ba, cd, ca, da\}$, and the pairs inconsistent with the above DAG are $\{ba, ca, da\}$. These arcs are then removed, and arc path lengths are altered appropriately:



Updating arc lengths involves a simple recursive check of paths leading to a node α whose outgoing arc is being deleted. If this deleted arc represented a maximum-length path for any paths that lead to α, then arcs in those paths have their lengths updated with the next largest path that exists for them in the DAG. This is directly determined from looking at path lengths in the DAG – no graph searching is necessary. Note that the deletion of a single arc affects a possibly exponential number of potential terms, due to the fact that there can be an exponential number of traversable paths leading to the two nodes whose connecting arc is removed. Also note that the whole graph might eventually disintegrate. This would indicate that the example set only permits a degenerate solution, eg. $a \mid\mid b \mid\mid c \mid\mid d$.

After all the examples are processed, the final step is to determine an appropriate interleaving hypothesis. This is easily done from the DAG using greedy term extraction. Since the ordering criterion stipulates that a hypothesis with a maximally-sized term is preferred over an expression with lesser-sized terms, extracting a maximal-sized term simply requires finding an arc in the DAG with a maximal-sized length value. Once we find this, we traverse the DAG from there, recording the node elements composing that term. Those nodes and their respective arcs are deleted. This is recursively repeated until all the nodes in the DAG have been visited.

The above algorithm is $O(n^2)$ in complexity. This is because there are $\left(\begin{array}{c} n \\ 2 \end{array}\right) = (n^2 - n)/2$ pairs of elements to process from every example. Hypothesis construction, however, is $O(n)$.

3.3 PAC identification of $\mathcal{I}$

The derivation of a PAC model for these languages relies upon quantifying the size of the hypothesis space for the language in question. In fact, both [Chemillier, 1987; Ross, 1994b] show that the domain space for similar interleaving languages is related to the combinatorial problem of counting the number of partitions of finite sets of elements, which is given by the Bell number B_n for sets of size n [Andrews, 1976].

Theorem 2

$$B_0 = B_1 = 1$$
$$B_{n+1} = \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array}\right) B_k$$

Proof: *Informally, given an item indexed* $n+1$*, there are* $\left(\begin{array}{c} n \\ k \end{array}\right)$ *partitions of it into sets of size* k*. The remaining* $n-k$ *items can be partitioned into* B_{n-k} *sets, giving a total of* $\left(\begin{array}{c} n \\ k \end{array}\right) B_{n-k}$ *partitions. Summing over all possible* k*'s yields the Bell equation.* □

Theorem 3 *The* n^{th} *Bell number is asymptotically equivalent to*

$$B_n = m(n)^n e^{m(n)-n-1/2} / \sqrt{ln(n)} \qquad [\mathbf{2}]$$

where $m(n)ln(m(n) = n - 1/2$.
Proof: *See [Graham* et al.*, 1989].* □

Theorem 4 *Let* Q *be a sequence of* $m \geq 1$ *independent random traces from the trace of a target* $\mathcal{L}$ *expression* E *drawn under some probability distribution on the example space. Then, for large* $n = |\Sigma|$*, and for any* $0 < \epsilon < 1$*, the version space for* Q *will be* ϵ*-exhausted with probability* $1 - \delta$ *after approximately*

$$\epsilon^{-1}(ln(1/\delta) + n\ ln(m(n)) + m(n) - n - 1/2 - ln(\sqrt{ln(n)}))$$

independent random examples using the given probability distribution, where $m(n)ln(m(n)) = n - 1/2$.
Proof: *Take the approximate size of the universe of* $\mathcal{I}$ *expressions given in equation* [**2**] *in theorem 3, and apply it within the PAC equation* [**1**] *of theorem 1.* □

The ramification of theorem 4 is that $\mathcal{I}$ expressions are PAC identifiable.

4 Discussion

The PAC result found in this research shows that musical melodies generated with interleaving expressions are conducive to efficient automatic learning. Interleaving is an important mathematical model of a class of musical melody. Chemillier has illustrated the mathematical elegance of his interleaving music algebra, and music composition languages based on Petri nets and process algebra illustrate the usefulness of treating interleaving as a generative operator. Earlier work by Chemillier and Ross show that interleaving expressions can be exactly identified in polynomial time. The derivation of a PAC result is important, as it shows that a target expression within a given error of accuracy can be inferred efficiently from randomly selected positive examples. In fact, for the example language $\mathcal{I}$, the number of examples required tends to be $n(ln(n))$ small over the size of Σ. In other words, a "good" target expression will be identified very quickly from random examples.

A practical application of this work is that it is possible to implement efficient learning-based tools for many types of concurrent music languages. Work is underway in creating a melody-generating module for the MWSCCS musical process algebra [Ross, 1995]. Such a utility can be trained to recognize music programs that generate desired "styles" of interleaved melodies given representative examples of the desired melody. This can be also viewed as a means for automatic synthesis of music. Hence this technology can be used for both music recognition and generative composition.

The interleaving language studied here is quite simple. More complex interleaving schemes employing synchronization and cyclic repetition have been studied [Ross, 1994a; 1994b], and similar PAC results should hold for them as well. Of course, comprehensive music programming languages employ additional constructs, such as nondeterministic or stochastic choice operators. The

PAC result will no longer hold if the cardinality of the hypothesis space is infinite, and so convergence cannot be ensured. In these more general languages, other learning techniques will be required.

Acknowledgements: Thanks to Tom Jenkyns for helpful discussions.

References

[Aalbersberg and Rozenberg, 1988] I.K. Aalbersberg and G. Rozenberg. Theory of Traces. *Theoretical Computer Science*, 60:1–82, 1988.

[Andrews, 1976] G.E. Andrews. *The Theory of Partitions*. Addison-Wesley, Massachusetts, 1976.

[Angluin and Smith, 1983] D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.

[Chemillier and Timis, 1988] M. Chemillier and D. Timis. Toward a theory of formal musical languages. In *Proceedings ICMC*, pages 175–183, Cologne, 1988.

[Chemillier, 1987] M. Chemillier. Monoide Libre et Musique: Deuxieme Partie. *Theoretical Informatics and Applications*, 21(4):379–418, 1987.

[Chemillier, 1990] M. Chemillier. Solfege, Commutation Partielle et Automates de Contrepoint. *Math. Inf. Sci. Hum.*, 28(110):5–25, 1990.

[Graham *et al.*, 1989] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, New York, 1989.

[Haus and Rodriguez, 1988] G. Haus and A. Rodriguez. Music Description and Processing by Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets (LNCS 340)*, pages 175–199. Springer-Verlag, 1988.

[Haus and Sametti, 1992] G. Haus and A. Sametti. ScoreSynth: A System for the Synthesis of Music Scores Based on Petri Nets and a Music Algebra. In D. Baggi, editor, *Computer-Generated Music*, pages 53–77. IEEE Computer Society Press, 1992.

[Haussler, 1988] D. Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221, 1988.

[Hoare, 1985] C.A.R. Hoare. Programs are Predicates. In C.A.R. Hoare and J.C. Shepherdson, editors, *Mathematical Logic and Programming Languages*, pages 141–155. Prentice-Hall, 1985.

[Kaufmann, 1968] A. Kaufmann. *Introduction a la combinatorique en vue des applications*. Dunod, 1968.

[Mayer and Stockmeyer, 1992] A.J. Mayer and L.J. Stockmeyer. The complexity of word problems - this time with interleaving. Technical Report RJ 8947, IBM Research Div., San Jose, CA, 1992.

[Milner, 1989] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Olderog and Hoare, 1986] E.-R. Olderog and C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informatica*, 23:9–66, 1986.

[Peterson, 1977] J.L. Peterson. Petri Nets. *Computing Surveys*, 9(3), September 1977.

[Ross, 1994a] B.J. Ross. The Inductive Inference of Cyclic Synchronized Interleaving. In *Proceedings of the 11th European Conference on Artificial Intelligence*, Amsterdam, 1994. John Wiley and Sons.

[Ross, 1994b] B.J. Ross. The Inductive Inference of Finite Interleaving with Synchronization. Technical Report CS-94-02, Brock University, Dept. of Computer Science, June 1994.

[Ross, 1995] B. Ross. A Process Algebra for Stochastic Music Composition. Technical Report CS-95-02, Brock University, Dept. of Computer Science, February 1995.

[Valiant, 1984] L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.