CHAINS: Competition and Cooperation between Fragments Event Predictors in a Model of Auditory Scene Analysis

Robert Mill, Tamás Böhm, Alexandra Bendixen, István Winkler, and Susan L. Denham

Abstract—This paper presents an algorithm called CHAINS for separating temporal patterns of events that are mixed together. The algorithm is motivated by the task the auditory system faces when it attempts to analyse an acoustic mixture to determine the sources that contribute to it, and in particular, sources that emit regular sequences. The task is complicated by the fact that a mixture can be interpreted in several ways. For example, a complex pattern may issue from a complex source; or, alternatively, it may arise from the interaction of many simple sources. The idea pursued here is that the brain attempts to account for an incoming sequence in terms of short, fragmentary sources, called chains. Chains are built as the input arrives and, once built, are used to predict inputs. A group of chains can coalesce to form an organisation, in which the member chains alternately generate predictions. A chain fails upon making an incorrect prediction, and any organisation it belongs to collapses. Several incompatible organisations can exist in parallel. The CHAINS algorithm thus remains open to multiple interpretations of a sequence. Perceptual multistability, in which the perceptual experience of an ambiguous stimulus switches spontaneously from one interpretation to another, seems to require a similar flexibility of representation.

I. INTRODUCTION

The physical laws that govern the production of sound and its transmission through the air constrain to a great extent the acoustic signals that can arise in natural environments. In particular, the mechanical properties of an object dictate the range of sounds that it can generate, e.g., when struck. The auditory system exploits these ecological principles to associate sounds with sources [1]. When multiple acoustic sources are present, their sounds overlap and interleave, and the perceptual system must disentangle the acoustic mixture in order to meaningfully interpret the environment. Bregman [2] introduced the term auditory scene analysis to describe this process. Complex sound mixtures are first analysed into elements and then regrouped to form perceptual streams on the basis of grouping cues. Simultaneous cues promote the fusion of elements that occur at the same time, e.g., common onset or harmony. Sequential cues are those that provide evidence that sounds should be linked across time, e.g., common pitch, intensity or timbre. These cues cannot unambiguously resolve every stimulus to establish a single physical scenario. To take a well-known example [3], sequences of isochronous tones in the format \(ABA-\), where \(A\) and \(B\) denote frequencies and \(-\) denotes a silent gap equal in duration to a \(B\) tone, yield multiple interpretations, which, in line with the ecological principles set out above, reflect multiple physical possibilities (see Fig. 1).

In this article, we take the view that auditory input events are encoded as a valued point process, and that the perceptual system seeks to extract a description of the process from the events as they unfold. Ideally, such a description would take on the 'shape' of the cause (or causes) in the real world and provide a basis for predicting future events. For instance, a verbal description might run as follows: “\(A\) is repeating with a period of 500 ms.” The system should also satisfy four other criteria. (1) It should be able to generate multiple alternative descriptions of the same input process; e.g., a single sequence versus two interleaved sequences (see Fig. 1). (2) It must accomplish this in an online way; i.e., descriptions should form as the input events occur, as opposed to the algorithm having access to the entire sequence at all times. (3) It should gracefully handle jitter or drift in the point values or their temporal pattern. (4) It should be conservative and the descriptions nonmonolithic; i.e., if one aspect of a description fails, then other aspects that do not fail should be preserved. This paper presents an algorithm called CHAINS, which meets these criteria. Perceptual coding is the motivation for, rather than the subject of, this paper. The key aim is to address general coding principles suitable for the kind of task outlined in this paragraph. In fact, in reviewing the list of requirements above, it is clear that they could apply to any system that responds to discrete events generated by multiple sources in the environment (e.g., events in a video stream). For this reason, the work is also likely to be of interest to those working in the information-theoretic and engineering disciplines.

II. MODELLING SEQUENCES

Many mathematical frameworks have been developed for the purpose of characterising sequences. In this section, we contrast the CHAINS algorithm with two such formalisms: formal languages and stochastic processes. We find that the
generates definite predictions. This determinism allows (i) a
in that it stores the recent history of the input itself, and
HAINS
about upcoming events, the C
(CTMPs, [9]), partly resolve this issue, but, at the same time,
Stochastic models such as continuous-time Markov processes
which renders them unsuitable for processing auditory signals.

X
an event
x
followed by a pause of
w
seconds. The entire list
Q
is a compact description of a sequence of
X
and a single chain can participate in multiple organisations.

Fig. 2. Diagrams of a chain. The top row shows a transition diagram of
A) the chain \( c_{eg} \), and B) the chain \( c_{eg} \) with the loop flag raised. The bottom
row plots the chain \([(200, 0.2), (600, 0.1), 0.05, f] \) in time-event space,
with C) \( f = \emptyset \) and D) \( f = \{ \text{loop} \} \). The phase is marked with an open circle.

III. Chains

A chain is a compact description of a sequence of
X
and their temporal pattern. Chains are constructed dynamically as
events occur and, once there is evidence of repetition, they are
closed into loops that generate predictions. Chains that predict
disjoint sets of events are capable of forming an organisation,
and a single chain can participate in multiple organisations.

A. Data Types

The inputs to the model are drawn from the set of events \( \mathcal{X} \),
which can be finite or infinite, countable or uncountable. For
instance, if the input events are tones, then we may wish to
categorise them by a label (e.g., \( \mathcal{X} = \{ \text{A, B} \} \)), a frequency
(\( \mathcal{X} = \mathbb{R} \)), or a multi-dimensional quantity that mixes features
(e.g., \( \mathcal{X} = \mathbb{R} \times \{ \text{trumpet, flute} \} \)). The only requirement is
that \( \mathcal{X} \) be a metric space. That is, we define a distance metric
\( d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \)
such that \( \forall x, y, z \in \mathcal{X}, d(x, z) = 0, d(x, y) = d(y, x) \) and
\( d(x, y) + d(y, z) \geq d(x, z) \).

A chain is a compact and straight-forward representation of the repetitive
patterns in the input, whilst CTMPs require sparse, high-order
transition distributions, or advanced variable-order Markov
models [10] to fully generate the input; and (ii) to assess the
correctness of predictions in a straight-forward way. Note that
the training of Hidden Markov Models, popular in automatic
speech recognition [10], [11], is usually based on predefined
automata structures, thus rendering it inapplicable for our
purposes.

In summary, it seems that there are a number of algorithms
in the literature that exhibit a superficial similarity to CHAINS.
However, the smaller or larger differences in their specifics
mean that they cannot address the problem expressed in the
introduction.
The phase \( \phi \) corresponds to the current time in seconds in relation to the first event in the chain. The flag set \( f \) stores properties of the chain.

In what follows it is useful to use \( x_i(c) \) and \( w_i(c) \) to abbreviate the \( i \)th event and pause of chain \( c \), respectively. In addition to the basic properties, outlined above, a chain has a number of derived properties. The number of events in chain \( c \) is denoted \( n(c) \equiv |Q(c)| \). The total duration of the chain, \( l(c) \), is found by summing over the pauses: \( l(c) \equiv \sum_{i=1}^{n(c)} w_i(c) \). Note that events themselves are represented as instants. (Event duration can be encoded as an additional feature, if necessary.) Similarly, the time of the \( i \)th event relative to the current time is \( \tau_i(c) = \sum_{k=1}^{i-1} w_k(c) - \phi(c) \).

We now consider an example. Let \( \mathcal{X} = \{A, B\} \) and define the chain

\[
{c_{eg} = \langle \{(A, 0.2), (B, 0.1)\}, 0.05, \emptyset \rangle}.
\]

This chain consists of the event \( A \), a pause of 200 ms, the event \( B \), and another pause of 100 ms. The phase of the chain is 50 ms, which means that \( A \) occurred 50 ms ago, and \( B \) will occur in 150 ms. For derived quantities, we have \( n(c_{eg}) = 2 \) and \( l(c_{eg}) = 0.3 \).

A chain, as a sequence of events, can be drawn in graphical form as a transition diagram, in which the nodes and arcs are labelled with events and intervals of time, respectively. If relevant, the phase of the chain can be indicated by placing a marker on an arc. Fig. 2A shows the chain \( c_{eg} \). If the loop flag appears in the set \( f \), then a chain represents an endless cycle of events. These are represented as a cycle graph. For example, setting \( f(c_{eg}) = \{loop\} \) results in the diagram shown in Fig. 2B. Alternatively, events in a chain can be plotted in a time-event space, as shown in Fig. 2C.D. This is particularly useful if \( d \) is a continuous function of its arguments, in which case the case between events can be conveyed diagramatically.

B. Algorithm Overview

The CHAINS algorithm processes three types of chain: build chains, which are always linear, prediction chains, which are always loops, and input chains, which may be linear or loops. We use \( C \) to denote the current set of build and prediction chains, and \( c_{in} \) to denote a single input chain. The basic structure of the algorithm is simple: actions are triggered by events in either input chains or prediction chains. Input events cause chains to be built; prediction events cause chains to be retained (if they succeed) or removed (if they fail).

A pseudocode listing for the CHAINS algorithm is supplied in Fig. 3. After the set of chains is initialised to the empty set (line number #2), the algorithm iterates until there is no more input (#3 – 13). If the input is a loop chain, then the algorithm runs continually. Four operations are carried out on each iteration. The first creates and extends build chains according to incoming input events (#4 – 7). The second removes chains that fail in their predictions (#8). The third advances all the chains to the next event, which is either an advance phase (#10). The fourth changes those chains that succeed to predict (if they are build chains) (#11). The pseudocode is as follows:

```plaintext
1: procedure CHAINS(c_{in})
2: \( C \leftarrow \emptyset \) initialise
3: repeat
4: for all \( x \in \text{NOW}(c_{in}) \) do
5: \( C \leftarrow \bigcup_{c \in C} \text{ADD}(c, x) \) add / close
6: \( C \leftarrow C \cup \{[(x, 0)], \emptyset, \{\text{build}\}\} \) new singleton
7: end for
8: \( c_{in} \leftarrow \text{ADVANCE}(c_{in}) \) remove failed
9: \( t' \leftarrow \min(\text{NEXT}(c_{in}) \cup \bigcup_{c \in C} \text{NEXT}(c)) \) advance input
10: \( C \leftarrow \{\text{ADVANCE}(c, t) : c \in C\} \) advance chains
11: \( C \leftarrow \{c \in C : \text{PKEEP}(c)\} \) vet chains
12: until \( \text{NEXT}(c_{in}) = \emptyset \)
13: end procedure
```

Next, we consider an example. Let \( x_{i}(c) \) and \( w_{i}(c) \) to abbreviate the \( i \)th event and pause of chain \( c \), respectively. If the phase \( \phi \) appears in the set \( f \), then a chain represents an endless cycle of events. These are represented as a cycle graph. For example, setting \( f(c_{eg}) = \{loop\} \) results in the diagram shown in Fig. 2B. Alternatively, events in a chain can be plotted in a time-event space, as shown in Fig. 2C.D. This is particularly useful if \( d \) is a continuous function of its arguments, in which case the case between events can be conveyied diagramatically.

```plaintext
15: function ADD(c, x)
16: if \( \text{build} \in f(c) \) then
17: \( c' \leftarrow c \) copy
18: \( Q(c') \leftarrow Q(c') + [(x, 0)] \) append event
19: \( C_{add} \leftarrow \{c'\} \) add
20: if \( d(x_1(c'), x_n(c')(c')) \leq \delta \) then
21: \( c'' \leftarrow c \) copy (original)
22: \( f(c'') \leftarrow \{\text{loop, predict}\} \) set to predict chain
23: \( \phi(c'') \leftarrow 0 \) reset phase
24: \( C_{add} \leftarrow C_{add} \cup \{c''\} \)
25: end if
26: end if
27: return \( \{c\} \cup C_{add} \)
28: end function
```

This is particularly useful if \( d \) is a continuous function of its arguments, in which case the case between events can be conveyied diagramatically.

```plaintext
29: function NOW(c)
30: return \( \{x_i(c) : i \in 1 \ldots n(c), \tau_i = 0\} \)
31: end function
```

```plaintext
32: function NEXT(c)
33: \( t \leftarrow \min(\{\tau_i(c) : i \in 1 \ldots n(c), \tau_i(c) > 0\} \) soonest time
34: if \( t = \emptyset \) and \( \text{loop} \in f(c) \) then
35: \( t \leftarrow \{l(c) - \phi(c)\} \) event in next iteration
36: end if
37: return \( t \)
38: end function
```

```plaintext
39: function ADVANCE(c, t)
40: \( \phi(c) \leftarrow \phi(c) + t \) advance phase
41: if \( \text{loop} \in f(c) \) then
42: \( \phi(c) \leftarrow \phi(c) - l(c) \) \( \frac{\phi(c)}{l(c)} \) wrap phase
43: end if
44: if \( \text{build} \in f(c) \) then
45: \( w_{n(c)}(c) \leftarrow w_{n(c)}(c) + t \) extend final arc
46: end if
47: return \( c \)
48: end function
```

Fig. 3. A pseudocode listing for the CHAINS algorithm. The functions \( \text{PFFAIL} \) and \( \text{PKEEP} \) are listed in Fig. 5. Note that \( \lceil x \rceil \) gives the largest integer less than or equal to \( x \), and \( \min(S) \equiv \{s \in S : \exists s' \in S : s' < s\} \). The + operator in line #18 performs list concatenation.
input or a prediction (#9 – 11). The fourth removes any chains that grow too long in the process (#12).

C. Building Chains

The first step of an iteration tests whether any input events occur in the chain $c_{in}$ at the current time (#4). An input event $x$ has three consequences: a build chain, consisting solely of that event, is created de novo (#6); existing build chains are duplicated, and the event $x$ is appended to each copy; and newly-extended chains that evince repetition are duplicated again and converted to predict chains. The last two of these are performed in the ADD subroutine (#15 – 28).

The ADD subroutine takes a chain $c$ and returns a set of up to three chains: $c$ itself (always returned), an extended version of $c$ (if $c$ is a build chain), and a predicting version of $c$ (if $c$ can close). A linear build chain is able to “close” into a looping prediction chain, if it satisfies some closure criterion, which, in this case, is simply that the first and last events in the chain are sufficiently similar, according to a distance threshold $\delta$ (#20). If the closure criterion is met, then a corresponding closure action is carried out. Here the closure action simply removes the last event and converts the chain to a prediction chain by modifying its flags and phase (#21 – 24).

It should be noted that the particular closure criterion and action given (#20 – 25) do not form part of the CHAINS specification, which invites the experimenter to specify their own functions to perform these operations.

Figure 4 provides an example of chains being built in response to the input $[(100, 0.1), (200, 0.1), (102, 0.1)]$. In this case, the closure criterion and action are those given in Fig. 3, using $d(x_1, x_2) = |x_1 - x_2|$ and $\delta = 5$. The events 100 and 200 are too distant to allow the chains to close, but the distance between the first and third events is small enough to allow the chain to close.

D. Passage of Time

The current time in the model is updated in discrete jumps. There is no global clock, to which all chains have access. Instead, the phase of a chain, $\phi$, keeps time in relation to its own events. Timing control is delegated to three subroutines, NOW, NEXT and ADVANCE. The NOW subroutine (#29 – 31) returns the set of events that occur in a chain at the current time. The NEXT subroutine (#32 – 38) returns a singleton set containing the amount of time until the next event in a chain, excluding current events, or the empty set if there are no more events. The next event in a loop chain may occur on the next iteration of the chain (#41 – 43).

The ADVANCE subroutine (#39 – 48) advances a chain in time by adding to its phase. The phase of a loop chain $c$ is wrapped onto the interval $[0, l(c)]$ following addition (#42). For build chains, the pause after the most recent event extends as time passes (#45). The main algorithm loop first computes the time interval until the next event and then advances all chains by that amount.

```plaintext
49: function PFAIL(c, c_{in})
50: \quad X ← \text{NOW}(c) \quad \text{predictions now}
51: \quad return \text{predict} ∈ f(c) ∧ ∃x ∈ X \rightarrow \text{PMATCH}(c_{in}, x, 0)
52: end function

53: function PKEEP(c, C)
54: \quad C ← \{c ∈ C : l(c) < ε\}
55: \quad C ← \{c ∈ C : \exists c' ∈ C : l(c') < l(c) ∧ \text{PEQV}_{T}(c, c')\}
56: \quad return C
57: end function

58: function PMATCH(c, x, t)
59: \quad for i ∈ 1 \ldots n(c) do
60: \quad \quad Δt ← τ_i - t \quad \text{time difference}
61: \quad \quad Δx ← d(x_i(c), x) \quad \text{event difference}
62: \quad \quad if loop ∈ f(c) then
63: \quad \quad \quad Δt ← Δt - l(c) \left(\frac{\Delta t}{l(c)} + \frac{1}{2}\right) \quad \text{wrap}
64: \quad \quad end if
65: \quad \quad if Δt^2/σ_t^2 + Δx^2/σ_x^2 ≤ 1 then
66: \quad \quad \quad return true \quad \text{match}
67: \quad \quad end if
68: \quad \quad end for
69: \quad \quad return false \quad \text{no match}
70: end function
```

Fig. 5. Continuation of the pseudocode listing for the subroutines that handle event matching, which are invoked in Fig. 3. The relation PEQV is described in the text of section III-F.
E. Predicting Events

Once build chains are closed, they start predicting inputs and remain in \( C \) as long as their predictions do not fail. The failure of a chain is tested by the predicate \( PFAIL \). A chain fails if it is a prediction chain and one of the events it predicts does not match the input chain \( c_{in} \) (#51). Because events unfold in continuous time, and the events themselves may be continuously-valued, it is necessary to allow some tolerance when matching them. \( PFAIL \) makes use of another predicate, \( PMATCH \), which takes a chain \( c \) and searches for an event \( x \) in \( c \) at time \( t \), where \( t > 0 \) refers to the future. The form that \( PMATCH \) takes is at the discretion of the experimenter, but a basic routine is suggested in #58 – 70, which is based on a scaled Euclidean metric (#65). The parameters \( \sigma_t \) and \( \sigma_x \) control the width and height of an elliptical matching region in a time-event space (see Fig. 6).

In keeping with the principle that a chain is only responsible for predicting part of the input, it is not removed when an input event has no corresponding prediction, only when a prediction has no corresponding input event. Fig. 6 illustrates the matching procedure in time-event format. The first three predictions of the chain succeed, the unpredicted input is ignored, and the fourth prediction finds no match, at which point \( PMATCH \) fails.

F. Removing Excess Chains

The number of build chains in \( C \) grows exponentially from just three input events. The removal of surplus chains is handled by the predicate \( PKEEP(c, C) \), which is satisfied only if a chain \( c \) should be retained. The details of its implementation are unspecified, but it should at least ensure that: (i) build chains do not grow indefinitely, and (ii) subsets of predict chains in \( C \) that are very similar do not emerge. To illustrate the second of these points, if the input sequence \( ABABABA \ldots \) is submitted, then multiple instances of the same predict chain—a repeating \( AB \) sequence—will be derived. The periodic formation of precisely identical chains does not pose a problem, because \( C \) is a set. However, if the A’s and B’s are jittered very slightly in either time or some event dimension, then sethood no longer counteracts this problem.

In order to detect and remove redundant chains, it is useful to define a tolerance relation, (loosely) termed prediction equivalence, which is denoted \( E\text{qv}_T \). \( E\text{qv}_T(c_1, c_2) \) is satisfied if and only if both \( c_1 \) and \( c_2 \) are prediction chains, and every event predicted by \( c_1 \) in the next \( T \) seconds matches an event in \( c_2 \), and vice versa. Fig. 7A shows an example of two equivalent chains in a time-event space. With this relation defined, a minimal version of the \( PKEEP \) predicate can be put forward (#53 – 57), which removes a chain if it exceeds a certain duration \( \epsilon \), or there is an equivalent, shorter chain.

IV. Organisations

A. Definition

Organisations form when a subset of chains in \( C \) predict the input as a group. Formally, an organisation is any subset \( \Omega \subseteq C \) such that

\[
\forall c_1, c_2 \in \Omega, COM_T(c_1, c_2),
\]

where \( COM_T \) is the prediction complementariness between two chains on the interval \( T \). \( COM_T(c_1, c_2) \) is satisfied if and only if both \( c_1 \) and \( c_2 \) are prediction chains, and no event predicted by \( c_1 \) in the next \( T \) seconds matches by an event in \( c_2 \), and vice versa. Fig. 7B shows an example of two complementary chains in a time-event space.

From the definition above, it is clear that, like a chain, an organisation need not predict all the input events. However, those organisations that do predict all the events in a cycle are the most useful. These we refer to as complete. Figure 8 shows the set of complete organisations that can be derived from the repeating \( ABA \ldots \) sequence. Each diagram shows the set of chains in an organisation, arranged such that if the chains were to all rotate clockwise with the same circumferential speed, the events passing through a common centre (dotted circle) would reproduce the sequence.

B. Selecting an Organisation

This example demonstrates that even simple sequences can yield multiple, mutually exclusive interpretations, where there is nothing in the input sequence to promote one interpretation over another. On what basis, then, should an organism or mechanism select an organisation? The auditory and visual system in humans, when faced with such ambiguous stimuli, explore the range of possibilities by switching stochastically
amongst percepts (e.g., [12]–[14]). The preceding sections have shown that the process of extracting the organisations can be captured algorithmically. Presumably, an additional component that switches stochastically between organisations could be added to CHAINS. The question then becomes, how many processing resources should be devoted to considering each organisation?

We close by considering briefly which organisations should be favoured, leaving aside the dynamics of switching. One possibility is to prefer organisations that are less complex. A reasonable candidate expression to quantify the complexity of an organisation might be

\[ h(\Omega) = |\Omega| + \sum_{c \in \Omega} n(c), \]

where the first term accounts for the cost of storing a chain, and the second term accounts for the cost of listing their events. Using this measure, organisations A and B in Figure 8 would be least complex \((h = 4)\); C and D would be more complex \((h = 5)\); and E would be most complex \((h = 6)\). The auditory system would appear to use a rule similar to this: when \(ABA^-\) is presented to listeners, the dominant percepts correspond to organisations A and B.

V. Conclusion

When listening in everyday mode, we are aware not only of isolated sound events, but also the patterns into which those events are organised—rhythms, repeating sequences, etc.—and often it is the patterns themselves that we find most arresting. We have presented an algorithm, CHAINS, for extracting the rules that give rise to temporal patterns. It does this by generating many, parallel hypotheses, and testing their predictions online. Groups of compatible chains can cooperate to form organisations. The types of organisations that emerge reflect various configurations in the environment.

CHAINS was first developed to describe how listeners parse ambiguous tone sequences with rhythmic repetitions. The algorithm is, however, quite general in nature and can be applied to any data type, including multimodal events. This is particularly relevant to the SCANDLE project, where rhythmic patterns from multiple sources need to be integrated; e.g., the sonar return from an arm swing and the sound of a footfall might constitute an event.

Finally, there are good prospects for the CHAINS algorithm to be expanded in various directions. For instance, a chain could be weakened rather than eliminated when it makes an incorrect prediction. There is also scope to introduce other kinds of chain. A natural suggestion, given the restriction of the current formulation to cyclic patterns, is a trigger chain, which, upon observing one sequence, predicts another. The broader aim is to build CHAINS in a direction laid down by a particular understanding of perception: namely, that it is an active process, in which algorithmic blocks are assembled and dismantled, in an ongoing attempt to explain, in various ways, the nature of what is being perceived.

References