University of Plymouth

PEARL **https://pearl.plymouth.ac.uk**

04 University of Plymouth Research Theses 01 Research Theses Main Collection

2009

Music as complex emergent behaviour : an approach to interactive music systems

Beyls, Peter F. E.

http://hdl.handle.net/10026.1/872

http://dx.doi.org/10.24382/4218
University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

University of Plymouth

PEARL **https://pearl.plymouth.ac.uk**

04 University of Plymouth Research Theses 01 Research Theses Main Collection

2009

Music as complex emergent behaviour : an approach to interactive music systems

Beyls, Peter F. E.

http://hdl.handle.net/10026.1/872

http://dx.doi.org/10.24382/4218
University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

MUSIC AS COMPLEX EMERGENT BEHAVIOUR: AN APPROACH TO INTERACTIVE MUSIC SYSTEMS

Volume 2 of 2

by

PETER F. E. BEYLS

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

9008801611
781.34 BEY

School of Computing, Communications and Electronics
Faculty of Technology

November 2009

90 0880161 1

Reference Only

LIBRARY STORE

Chapter 8: Machine Motivations and the *Drive* Object

8.1 Introduction

Interactive composing implies the exchange of musical information between man and machine. A musical discourse develops form the approval of material contributed by both parties. Improvisation becomes an abstract conversation that features variable levels of agreement between the human player (HP) and the machine player (MP). As identified in the previous chapter, this may be viewed as a process of attraction and repulsion. The confrontation is a result of the coupling that exists between two musical identities leading to complex temporal forms of conflict and agreement.

Provided that Oscar aspires autonomous behaviour one cannot accept its identity to be designed completely by an external human designer. In fact, explicit design is strongly discouraged throughout the whole thesis. The musical initiative of the MP should be unpredictable yet coherent: we are interested in the on-line development of behavioural profiles that document musically interesting interactions between MP and HP. In other words, one expects order to emerge from initial disorder. *Order* implies the detection of musical patterns that provide an apparent indication of either relative agreement or conflict. *Disorder* refers to the absence of such a clear opinion. A dynamic mechanism is needed that can make up criteria to interpret external agitation in terms of positive (agreement) or negative (conflict) impact. It must be robust and create an opinion by itself according to demands generated by its own internal dynamics. The *drive object* aims to provide such a structure.

A *drive* is a computational object that specifies a simple psychological orientation for a machine player. It can be considered an abstract suggestive *speculation* issued by that object. It has two options: integration or expression. Integration means that the

MP aims to produce music that integrates well with the last sequence played by the human player. In contrast, expression implies that the drive prefers to move away from the musical style suggested by the HP. The options are not mutually exclusive. They are viewed as two competing alternatives represented by two fluctuating quantities on a scale of 0 to 100. The drive object helps towards the materialisation of a first principle: the appreciation and accommodation of change (chapter 1, section 1.3.8).

Important information becomes available when either the MP or HP just finished playing their current melody. (For a discussion of the prediction algorithm used to make out if the HP is considered *just finished playing*, please refer to chapter 9). A first order quantity in Oscar is the current melodic distance between the last sequences produced by man and machine. More pertinent, the intention is to find out if that distance increases, decreases or remains the same over time. For instance, when the MP just finished, its effect on the situation can be computed. If the new distance is higher than the previous distance, one knows that the MP has contributed to the increase of the musical contrast between MP and HP. If the new distance is lower, then we know that both parties are musically getting closer together. Consecutive differences (*delta-similarities*) are tracked in time. If many such consecutive delta-similarities have the same sign, it is inferred that the MP and HP are either engaged in an escalating process of incremental contrast (negative sign) or apparent mutual understanding (positive sign).

Given a specific perception of behavioural changes in the HP, the MP must learn which behavioural motivation (integration or expression) should dominate. From this knowledge, an appropriate musical processing function can be selected to fulfil that specific orientation.

As explained in chapter 5, Oscar uses the notion of a *relationship* to specify a qualitative link between observed external changes and an internal quantity. This quantity represents the strength of an internal motivation. Internal motivations and external pressures are thus operationally connected as a complex dynamical system.

Recall that four types of relationships are employed; each specifies a different coupling between changes in input level *Qi* and changes in output level *Qo*. Both increments and decrements of input and output levels are considered yielding four relationships: (1) + +, (2) + -, (3) - +, (4) - -. Values evolve by way of multiplicative operators. The *activation-multiplier* used here are between 1.0 and 2.0 while the *inhibition-multiplier* is between 0.5 and 1.0.

8.2 Implementation of the Drive Object

A drive has three sensors, it is sensitive for three kinds of changes: (1) the first derivative of the similarity between the most recent melody produced by man and machine and (2) the first derivative of the quality and (3) quantity of the contents of the most recent man produced melody. All input changes are computed and normalized in a range -100 to +100.

A single drive is characterized by the following instance variables:

```
drive-ID            : 4
nr-runs             : 8
current-orientation : EXPRESSION
relationships Exp   : 1  0  1  0  1  1  0  1  0  0  1  0
relationships Int   : 1  0  1  1  1  0  0  0  1  0  0  0
firing-slot         :  0
expression-level    : 80.86634
integration-level   : 31.14185
efficiency-value    :  2.24824
efficiency-sum      :  8.50325
```

```
understanding-level : 45.21349
```

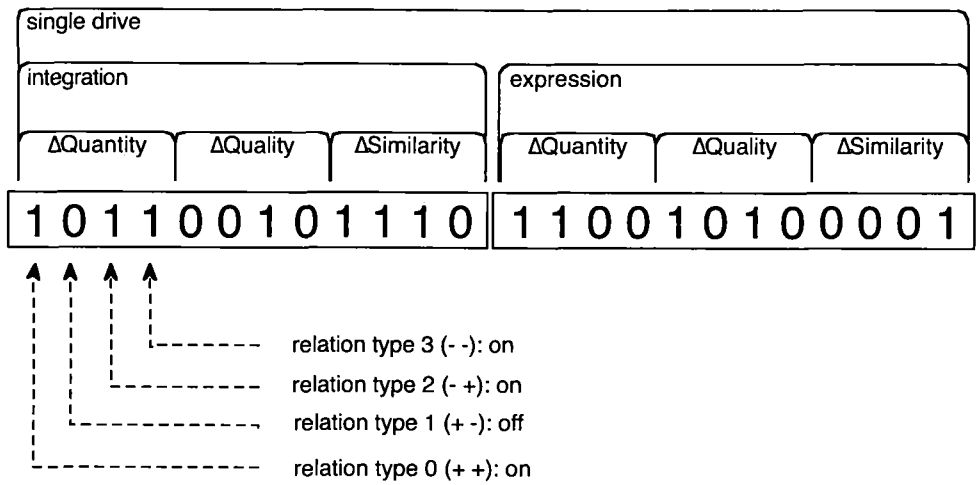


Figure 8.1: Topology of a single drive object showing three groups of four bits per motivation.

Relationships are specified as two 12-bit vectors. Since there are three types of input sensors each feeding four types of relationships, one must accommodate twelve potential effects of external change – three blocks of four bits. For instance, bits 0~3 account for delta-similarities, bits 4~7 account for delta-quality and bits 8~11 account for delta-quantity. Both primitive motivations must be included: integration and expression resulting in a total of 24 bits. If a bit equals to 1, it means that its relationship is active, if the bit equals to zero, its relationship is not accounted for. Note that many relationships can be active (bit on) in a single block. The output will reflect the contributions of all active relationships. When simultaneous relationships contribute opposite pressure, they may neutralise their mutual effect. This phenomenon contributes to non-linearity in the network.

Intuitively, it is understood that the density of "on" bits in the vector will condition the global responsiveness of the drive. Too many "on" bits may potentially produce over-stimulation leading to erratic output. In contrast, too few "on" bits lead

to under-stimulation, in this case, significant changes in input may get lost. We turn to a learning algorithm that learns to the appropriate couplings between input changes and internal motivations.

The behavioural motivation of a drive – its current orientation – depends on the strength of the two competing levels (0~100) for integration and expression. A minimum contrast should exist between both values; a threshold of 10 is introduced. If the difference between the levels for integration and expression is higher than 10, then the higher value decides on the orientation else the current orientation is decided at random. Therefore, the current orientation in the example above is *expression.* This has a double impact on further computations. First, the expression-vector becomes the source of temporary relationships and second, the output value affected by these relationships is the expression-level.

As an example, consider the first block of 4 bits of the Integration relationships: (1 0 1 1). Since the first bit is "on", relationship type 1 (+ +) takes effect. Thus when the input level increases the output level follows. Relationships type 2 (+ -) is not considered since the second bit if zero. The third bit is "on" meaning that the contribution of a relationship type 3 (- +) is added to the previous. In other words, when delta-similarity is either positive or negative, the output level will increase. In addition, the relationship type 4 (- -) says that if input level decreases the output level will follow in the same direction.

It is important to know how efficient a given drive actually is. When the external changes are processed by the relationships, they receive a qualitative interpretation because of non-linear couplings take place between the dynamics of external higher level quantities (similarity, quality and quantity) and competing internal behavioural motivations (integration and expression). Given the current orientation, we analyse if

MP and HP are coming together or drifting apart – according to their melodic similarity. For example:

```
activation-weight = 1.15
inhibition-weight = 0.85
penalise-weight   = 0.98


when
    current-orientation equals integration
    and
       (current-distance - previous-distance) > 0
    then
        efficiency-value =
             (min 100 (max 2 (efficiency-value * inhibition-weight)))
```

If the current orientation equals integration and the distance decreases, the *efficiency-value* is updated by the *activation-weight* (1.0 < weight < 2.0) and clipped to a maximum value of 100. The drive becomes more efficient because the sensed data confirms the present orientation.

```
when
    current-orientation equals integration
    and
       (current-distance - previous-distance) < 0
    then
        efficiency-value =
             (min 100 (max 2 (efficiency-value * activation-weight)))
```

If the current orientation equals integration and the distance increases, the *efficiency-value* is updated by the *inhibition-weight* (0.0 < weight < 1.0). In addition, when current and previous distances do not change, the drive is slightly penalized – the *efficiency-value* is multiplied by factor 0.98.

Every time a drive is run, the *efficiency-sum* is incremented by the current *efficiency-value*. At any point in time, the actual efficiency equals the *efficiency-sum* divided by the *nr-runs*.

8.3 Detection of Changes in Human-Machine Similarity

Two methods to compute melodic similarity were implemented. The first method is based on a series of transition tables that trace interval changes (not absolute values) for the dimensions of pitch, velocity and duration. Melodic similarity is inferred by summing the global differences for every cell in every transition table. The results are averaged and normalised to 0~100. Note that the Markov type transition tables are adaptive. When the contents of any array location hits a value of 100, all cells of the array are downscaled proportionally so that the maximum value in the array becomes 50 rather than 100. This creates a different yet still accurate content of the array while there is room to receive new input.

The matrix-comparator-element class holds three matrixes of different size; the pitch-matrix is 25 by 25, velocity-matrix is 16 by 16 and the duration matrix is 20 by 20 elements. It also holds a pointer to the melody that is currently subject to analysis. The matrix-comparator class keeps two instances of the matrix-comparator-element and keeps track of the previous and current total similarity so that amplitude and direction of change may be inferred. Pitch intervals are clipped to a range of -12 to +12 semitones, velocities (ranging 0 to 127) are normalised to 16 values and durations are clipped and scaled as to reside in a range of 20 units.

The difference-matrixes visualised below shows the *difference* between the contents of both matrix-comparator-elements – for the dimensions of pitch-intervals, velocity-intervals and duration-intervals.

Three options exist:

A) One of the matrix locations holds a positive value, but not both. When the first matrix value is positive, a -1 is shown.

B) Same situation as above, however, the second matrix holds a positive value and the first matrix holds zero, then a -2 is shown. The values -1 and -2 are only used to highlight critical locations in the matrix; they add no weight to the final compute-similarity method.

C) In case the respective matrix locations both provide positive values, the effective ratio is shown, normalised to a values between 0 and 100. This value is taken into account to compute global similarity.

The example below reflects the difference of the contents of working memory and the first function in the compound-function-pool:

```
Pitch-intervals transition matrix
. . . . . . . . . . . . . . . . . . . . . -1 . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . -1
  . . . . . . . . . . . -1 . . . -2 . . . . . . . . . .
  . . . . . . . . . . -1 . . . . . . . . . . . . . . .
  . . . . . . . . . -1 . -1 -1 . -2 . -1 . . . . . . . .
  . . . . . . . . . -2 . 50 . . . . . . . . . . . . . .
  . . . . . . . . . . . -2 . . . . . . . . . . . . . .
  . . . . . . . . . . . . -2 . -1 . -1 . . . . . . . .
  . . . . . . . . . -1 . . . . -1 -1 . . -1 . . . . . .
  . . . . . . . . . . . . . . . -1 . . . . . . . . . .
  . . . . . . . . . . . . . . . -1 . . . . . . . . . .
  . . . . . . . . . . . . . . . -1 . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . -1 . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . .
 -1 . . . . . . . . . . . . . . . . . . . . . . . . . .

Velocity-intervals transition matrix

 -1 . . . . -1 . . . . . . . . . .
  . . -1 . . . . . . . . . . . .
  . . -1 . . . . . . -1 . . . . .
  . . . . . . . . . . . . . . . .
  . . . . . -1 . . . . . . . . .
  . -1 . . . -1 34100 . . . . . .
 -1 . . . . -1 -1 50 . . . . . -2 . .
  . . . . . . 50 . -1 -1 . . . . . .
  . . . . . -1 . . . . . . . . . .
```

```
. . . . . -1 . . . -1 . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . -2 . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
```

Duration-intervals transition matrix

```
  6 25 . . . . -2 . . . . . . . . . . . . . . . .
100 -1 -1 . . . . . . . . . . . . . . . . . . . .
 -1 . . . . . . . . . -2 . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
-2 . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
-2 . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
```

The total similarity for a given parameter is computed according to the following algorithm:

```
(defmethod compute-similarity ((self matrix-comparator) parameter)
  (let ((s (active-locations-sum self parameter)) ;; sum of abs values
        (na (nr-active-locations self parameter)))
    (if (plusp s)
      (round (/ (loop with a1 = (funcall parameter (car (comparators self)))
                      with a2 = (funcall parameter (second (comparators self)))
                      with n = (- (car (array-dimensions a1)) 1)
                      initially (format t "~% similarity for parameter: ~a, Sum active-locations-
                sum: ~d Nr-active-loc: ~d."
                                        parameter s na)
                      for i from 0 to n sum
                      (loop with v
                            for j from 0 to n do
                            (if (zerop (+ (aref a1 i j) (aref a2 i j)))

                              ;; no contribution
                              (setq v 0)

                              (if (or (zerop (aref a1 i j))
                                      (zerop (aref a2 i j)))

                                ;; only one location non-zero, negative value
```

```
                    (setq v (round (* -1 (* 100 (/ (max (aref a1 i j)
                                    (aref a2 i j))
                                                    s)))))

                    ;; both non-zero
                    (setq v (* 100 (/ (- (max (aref a1 i j) (aref a2 i j))
                                         (abs (- (aref a1 i j) (aref a2 i j))))
                                      (max (aref a1 i j) (aref a2 i j)))))))
                sum v))
        na))
100)))
```

The algorithm *active-locations-sum* first computes *S*, the sum of the absolute values of the pair wise differences of all matrix locations for a given parameter - for instance, pitch. In other words, the sum S represents the *total contrast* of the current comparator contents. The nr-active-locations simply counts the number of matrix locations, again pair wise, where either one of the two matrixes holds a non-zero value. Next, the *compute-similarity* algorithm loops through both matrixes and sums the value V.

In case only one location is non-zero, a negative value is added to reflect the problem that both matrix locations are produce conflicting information. The amount of negative weight is computed proportional to the amplitude of the absolute value of single non-zero values. The result is normalised on a scale from 0 to 100 percent and multiplied by -1.

When the respective matrix locations in both matrixes hold non-zero values, we are in a position to evaluate exactly how they contribute to global similarity. The value v is computed as a ratio: the maximum-value minus the (abs difference) divided by the maximum-value. Again the result is normalised. Finally, the similarity is obtained by dividing the sum by the number of active locations.

Note that this method may return a negative value proportional to the amplitude of values in pairs of matrix locations where only one of these locations holds a positive number. The average of the contributions of the three parameters represents a global degree of *relative* similarity, the exact value of the similarity levels are not crucial in this particular case since we are only interested in *changes* between successive similarities.

The similarities for the matrixes in the example above are as follows:

```
Similarity PITCH: active-sum: 34, Nr-active-loc: 25. Similarity = -1.9600
Similarity VELOC: active-sum: 30, Nr-active-loc: 22. Similarity =  6.8333
Similarity DURAT, active-sum: 32, Nr-active-loc: 10. Similarity =  9.7263
```

After input of four new MIDI events to the contents of working-memory (number of event remains unchanged because of the FIFO-type memory structure) still with the first function in the compound-function-pool offers the following results:

```
Similarity PITCH: active-sum:  9, Nr-active-loc: 11. Similarity = -1.7500
Similarity VELOC: active-sum: 15, Nr-active-loc: 14. Similarity =  6.5600
Similarity DURAT, active-sum: 11, Nr-active-loc: 12. Similarity =  7.1349
```

This new input seemingly contained overlapping pitch-intervals with melodic material in the compound-function; a less negative similarity signals an increase in pitch similarity, similarities for the parameters velocity and duration become lower. The normalised (0~100) total change in similarity is obtained from:

```
(/ (+ (* -1 (* 100 (/ (- 1.9600 1.7500) 1.9600)))
      (* 100 (/ (- 6.8333  6.5600) 6.8333))
      (* 100 (/ ( - 9.7263  7.1349) 9.7263)))
   3)
```

The resulting increase in global similarity is 6.6428 percent.

An alternative method is to compare the contents of the histograms for the same four dimensions of the two melodies under observation. The more the histograms overlap, the lower the melodic distance between the two source melodies. This method is slightly less computationally expensive as it does not need adaptation.

8.4 Learning in the Drive Object

Our learning method is very similar to Reinforcement Learning (RL) (Sutton and Barto 1998). In RL, the learning agent receives feedback about how appropriate its actions are in order to achieve a given goal. RL provides only information that the previous action was not appropriate but does not offer instructions of what should be done in order to learn. Therefore, RL is a form of unsupervised learning.

RL is most frequently encountered in nature. An organism aims to maximize its rewards by undertaking the right action in response to the perception of particular states of the world. The organism (agent) learns a *policy* to map states to actions. However, the agent does not have access to a complete model of the environment. In addition, the agent has no initial knowledge of the kind of rewards to expect from interactions with the environment. RL is typically applied in real-world problems characterised by a huge state space. All of this seems to fit the essence of interactive composing: man and machine must learn to behave successfully without any *a priori* information about their mutual personalities.

Oscar's two-stage learning algorithm is depicted in figure 8.2. Stage one updates the levels of integration and expression from the evaluation of the current relationships. Stage two updates the efficiency according to the current orientation.

Let us tackle stage one in detail. For every delta value, the respective slot of the relationships-vector is evaluated.

Stage one encloses two nested loops; the arguments are the current gradients (*delta-values*) in human-machine melodic similarity and the changes in quality and quantity of the contents of working memory. All 24 bits of the *relationships-vector* are addressed (refer to figure 8.1). The levels are scaled up (activation) or down (inhibition) according to the type of relationship and the delta value. In the end, the integration and expression levels will reflect the accumulated impact of the combination of vector on-bits and the sign of the respective delta-values. Please refer to chapter 5, section 5.2 for a detailed account of the notion of relationships.

Stage two evaluates the resulting (potentially changed) *drive-orientation*, decided on by taking the highest value of the two competing levels as the winning current orientation. Take note that we exploit *only* the *delta-similarity* at stage two. This delta value and its sign provide information as to whether the current relationships were helpful to steer the drive towards the optimal orientation. The *intended* optimal orientation (integration or expression) is the one that is consistent with the last change in human-machine similarity. For example, when the human-machine melodic distance *decreases* and the orientation is *integration*, we conclude that the drive is indeed resourceful towards the fulfilment of this drive's orientation – therefore, its efficiency-level is scaled up using the *activation-factor*. In similar vein, in case the melodic distance *increases* and the orientation is *expression*, the efficiency level is also scaled up. Efficiency-level is inhibited when the changes in distance are in conflict with the orientation; i.e., either a combination of integration and increasing distance or expression with decreasing distance.

Remember, a given orientation is considered a machine *suggestion* to temporarily approach musical interaction from a given perspective; i.e., either a wish for man and machine drifting apart (expression) or narrowing the human-machine melodic distance (integration). The rationale is that a suggestion is first generated at random and subsequently adjusted according to the evaluation of the data gathered during actual interaction.

In total, four situations are considered combining two orientation and two delta signs – the efficiency levels are activated or inhibited according to the motivation explained above. At last, when the orientation is NIL; i.e., when there is less than 10% contrast between integration and expression levels, the efficiency of the drive is slightly downscaled using a typical *inhibition-factor2* value of 0.96.

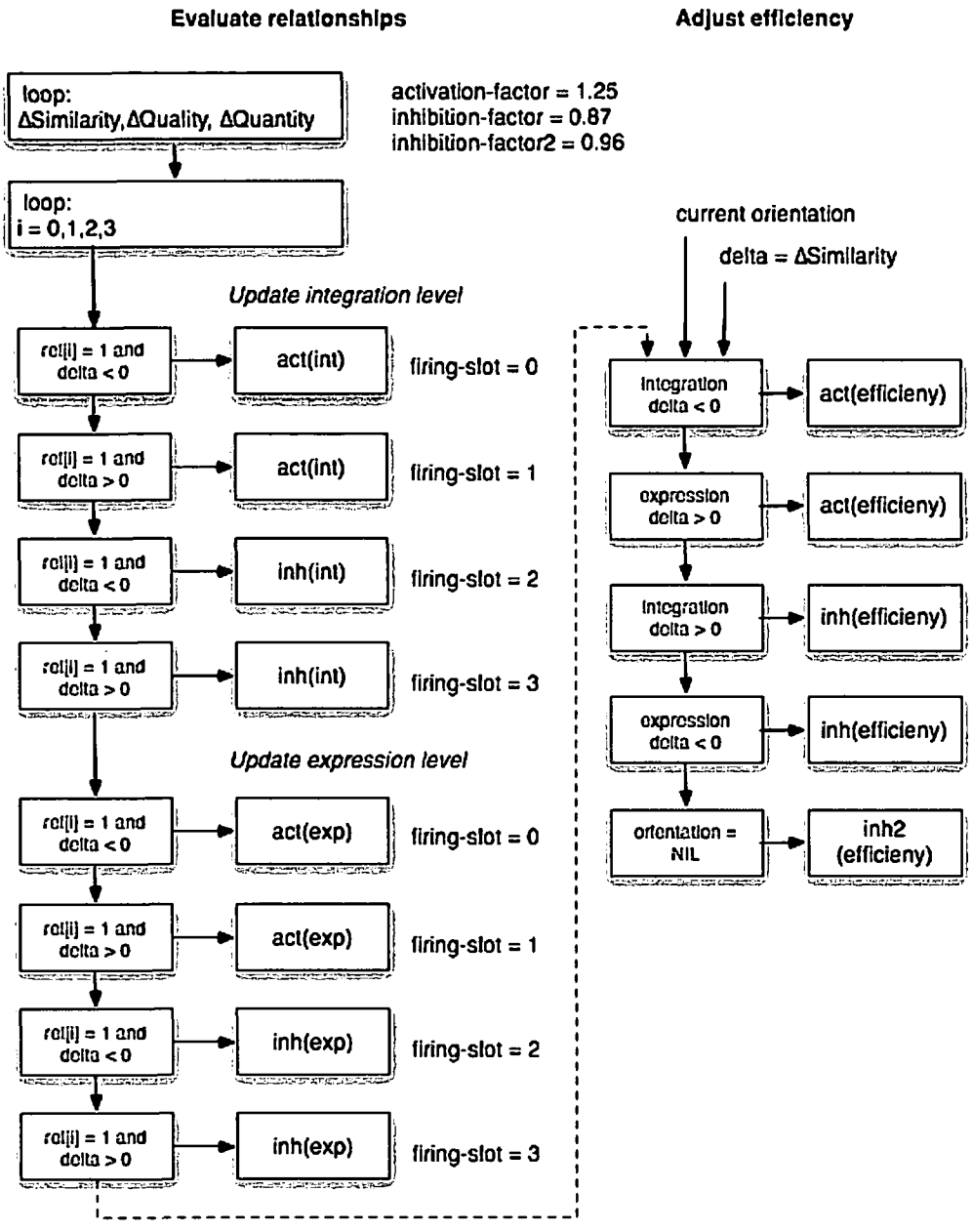


Figure 8.2: Manipulation of levels of integration and expression according to the evaluation of the drive's current relationships and adjusting the efficiency of the drive according to perceived changes; i.e., the delta value and its sign.

8.5 Definition of the Drives Pool

A drive is expected to develop relationships autonomously: the layout and strength of the couplings between changes in musical input by the human performer (HP) and internal changes influencing motivational levels ultimately guiding behaviour. The scope of the interplay of man and machine becomes virtually infinite because the HP behaves unpredictably and the non-linearity in the drive produces equally unpredictable orientations. However, we require a gradual acquisition of competence through the act of interaction itself. The drive learns the effect of external changes by adjusting its integration and expression levels. In a later phase, these levels can be addressed and exploited. For instance, if the integration-level dominates over the expression-level, the drive will instruct the compound-function-pool to exploit musical processing functions that help man and machine to integrate in the future.

A critical mass of drives is needed to guarantee the potential development of many different types of interactions. In the current implementation, the drives-pool contains between eight and 30 drives. The initial relationships are random with a density of 50 percent, and both orientation levels receive a value between 40~60. The rationale is to provide initial momentum for change in either positive or negative directions. Now, the idea is to select and run all drives using a random selection scheme. However, the chance for a drive to be selected is inverse proportional to the number of times it ran in the past. Thus, all drives get a chance to perform but not in any specific explicit order. For now, we only consider the learning process that takes place between any two points in time where the genetic operators are applied – typically several minutes. At the beginning of the learning period, any drive can be selected because none has developed an efficient behavioural orientation. Exploration takes place: the pool of drives is sampled at random and the orientation levels are pushed up or down. When a

clear contrast gradually emerges, one may decide to actually exploit the knowledge that was acquired online. So first, many options are given a chance to develop while later on, the promising ones are applied. A probabilistic ranking scheme is used that conditions efficient drives to be selected proportional to their actual efficiency level. Once the learning period is finished, the genetic operators are applied. The drive's efficiency-level is viewed as equivalent to fitness. The newly bred generation will thus reflect the knowledge gathered during the learning period.

The procedure to select a drive depends on three elements: the *learning-mode*, the *performance-mode* and an *input parameter* (0 to 100). The selection procedures is summarised in figure 8.4. Whatever the learning-mode, the input parameter is interpreted as a threshold to compare random numbers between 0 and 100 (the *coin* function). In other words, the input parameter represents a probabilistic value to favour exploration or exploitation; meaning, respectively, selecting a random drive from the current drives-pool or selecting a specific drive from the current group of efficient drives, if any.

A subtle selection mechanism is at work when performance mode equals *evolution-and-learning*. A second random number decides whether a selection will be based on (1) the argument of efficiency or (2) on the argument of the accumulated understanding-level. This second, higher-level selection level thus takes learning into account. In this case, *exploitation* means to make good use of very short-term information; the understanding-level as it is being conditioned by the current degree of human-machine understanding.

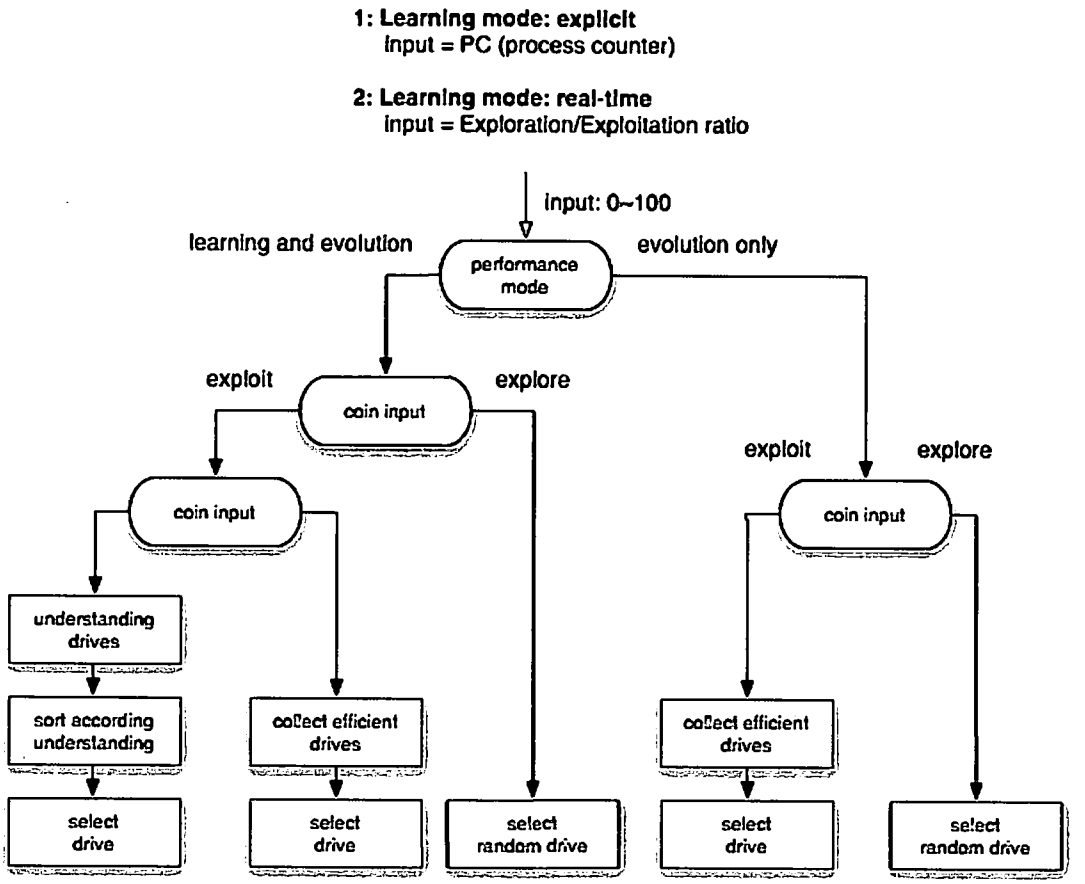


Figure 8.4: Drives are selected according to the current learning mode and an input value between 0 and 100. This value operates as a stochastic threshold to decide about the invocation of an exploration or exploitation oriented selection procedure.

When performance-mode equals *learning-and-evolution*, then:

```
if (random 100) < exploration-exploitation-ratio
    if (random 100) < exploration-exploitation-ratio
        select efficient drive
        else
        select understanding drive
    else
        select random drive
    then
```

When performance-mode equals *evolution-only*, then:

```
if (random 100) < exploration-exploitation-ratio
```

```
        select efficient drive
        else
        select random drive
        then
    then
```

8.6 Human-machine Common Understanding

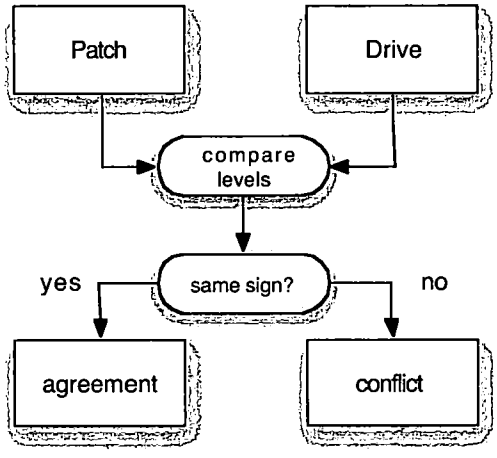


Figure 8.5: Top-level system-common-understanding is obtained by comparing the level and sign of the current output levels of patch and drive objects.

It is of vital interest to see whether the human and machine performers produce similar or contrasting information. To this purpose, the output of the current patch and the output of the current drive are compared. Both objects provide an output scalar between -100 and +100 for normalised evaluation. One needs a function to compute the global orientation of a drive or patch. It accepts contributions of the two competing forces, represented as levels for integration and expression. The *global-orientation* function also returns a value between -100 and 100. For example, the global-orientation for a drive is computed as follows:

```
(defmethod get-global-orientation ((self drive) &key (print nil))
  (when (and (plusp (expression-level self))
```

```
        (plusp (integration-level self)))
    ;; if both are zero then global-orientation = undecided
(let ((level-sum (+ (expression-level self)
                    (integration-level self)))
      (orient))
  (setq orient
        (if (> (integration-level self)
               (expression-level self))
          ;; return normalised value (strength) 0~100
          (round (* 100 (/ (integration-level self)
                           level-sum)))
           ;; integration = positive value
          (round (* -1 (* 100 (/ (expression-level self)
                                 level-sum))))))
           ;; expression = negative value
  (when print
    (format t "~%Drive-set global orientation: ~a." orient))
  orient)))    ;; return signed value
```

This function first checks whether both levels are positive, when either is zero, it is assumed there is no ground to compare the two competing levels. In the latter case, the function returns *nil*, denoting the orientation to remain undecided. The *level-sum* variable contains the sum of both levels. In case the *integration-level* supersedes the *expression-level*, this function returns a positive value; the integration-level as normalised (0~100) in relation to the level-sum value. In case the expression-level is higher that the integration-level, a negative normalised value is returned (-100~0), also respective to the level-sum. A positive global-orientation value thus denotes *integration* while a negative value denotes *expression*.

If one compares the global-orientation of both drive and patch, then a value designated as the temporary *system-common-understanding* is obtained. This value reflects the status of the complete system given the relationships articulated by the current user input (the patch) and the relationships articulated by the delta values

acting in the present drive. At this point we acquire an impression of system's behaviour as a whole: a weighted indication of agreement or conflict in the current human-machine interplay. The level of common understanding is computed as follows:

```
(defmethod get-common-understanding ((self improviser-class))
  ;; compute conflict or agreement plus strength
  (let* ((ear (interface self))
         (patch-orientation (get-global-orientation
                        (current-patch (patcher ear))))
         (drive-orientation (get-global-orientation
                        (current-drives-set (drives-pool ear))))
         (understanding nil)
         (diff nil))
    (if (and patch-orientation
             drive-orientation)
      (progn (setq

          understanding
               (if (eql(signum patch-orientation)
                               (signum drive-orientation))
                                   'agreement
                                   'conflict)

            diff
           (if (eql (signum patch-orientation)
                                (signum drive-orientation))
                       (round (/ (+ (abs patch-orientation)
                       (abs drive-orientation)) 2))
                       (round (/ (- patch-orientation drive-orientation) 2))))

          (format t "~% Improviser common understanding: ~d percent ~a."
                diff understanding))
      (format t "~%  Improviser common understanding is NIL."))
    diff))  ;; return signed value(minus = conflict, plus = agreement) or NIL
```

The following four combinations potentially exist:

Patch global orientation	Drive global orientation	System common understanding
integration	integration	agreement
integration	expression	conflict
expression	integration	conflict
expression	expression	agreement

Table 8.1: System common understanding as a function of the patch and drive global orientation.

When patch and drive global-orientations carry the same sign, system-common-understanding becomes the average of the absolute values of both objects, in case the signs are disagreeing, system-common-understanding is proportional to the difference of the signed orientations.

Conflict results when patch and drive present contrasting output. Otherwise, when patch and drive generate values with the same sign (both positive or negative), they aim to behave according to the same articulation; i.e., either integration or expression. Therefore, it is concluded that man and machine are in relative agreement of the current situation.

The common-understanding value is further used to adjust the level of appropriateness of the drive that contributed to the current level of common understanding as detailed next.

The drive object is the only object in the system that has the ability to learn. Drives feature an *understanding-level* instance variable, its level (0 to 100) is adjusted according to the following function:

```
(defmethod adjust-understanding-level ((self drive-set) cu)
      ;; adjust level according to common-understanding
```

```
        (let ((factor (if (plusp cu)
                          (remap cu 0  100 1 3)
                          (remap cu 0 -100 1 0.3))))
      (setf (understanding-level self)
            (max 2 (min 100 (* (understanding-level self) factor))))))
```

The level of system-common-understanding is remapped according to its sign. If the system-common-understanding argument is positive, its value (1 ~ 100) is scaled to the range 1 to 3. That value is used as a multiplier to scale the understanding-level. Likewise, negative common understanding values (0 ~ -100) proportionally remap to a multiplier of 1 to 0.3.

The *understanding-level* is a learned indication of how successful the drive contributes to an interaction climate characterized by human-machine *agreement.*

Finally, one may derive a global conclusion about the nature of the current social relationship between the machine and human performer and compute the *system-global-orientation.* The latter reflects the momentary human-machine correlation and is easily derived by averaging of the global orientation of current patch and current drive:

```
System-global-orientation = (patch-orientation + drive-orientation)/2
```

Two exemplary printouts of momentary human-machine interaction exposed in the improviser global orientation and common understanding:

```
patch-orientation             : 100 INTEGRATION.
drive-orientation             : -46 EXPRESSION.
system-global-orientation     :  27 INTEGRATION.
system-common-understanding   :  73 percent CONFLICT.
```

```
patch-orientation            : -37 EXPRESSION.
drive-orientation            : -45 EXPRESSION.
system-global-orientation    : -41 EXPRESSION.
system-common-understanding  :  41 percent AGREEMENT.
```

As an example, consider the data in figure 8.6, it shows a history trace of levels of global-orientation for the patch and drive objects. Both levels start with negative values; i.e., they both agree on the common objective of *expression*. At first, the two polynomials move in similar motion but end up moving in a phase difference of 180 degrees towards the end of the experiment. Note this profile is characteristic and extends over the duration of the experiment[2]. When zooming in on the data, interesting oscillatory behaviour is observed. Figure 8.7 shows global-orientations over a data block of 100 samples, about halfway through experiment e7. Both levels develop gradually variable phase relationships which results in zones of human-machine cooperation towards common ends or otherwise, the appearance of egocentric behaviour potentially resulting in conflicting objectives. The global emergent effect is a propagation of weighted zones of variable social pressure between both interacting parties.

Chapter 10, section 10.4.8 provides additional examples of correlation analysis between the drives global orientation and system common understanding.

[2] The total duration of experiment e7 is 38 minutes 24 seconds.

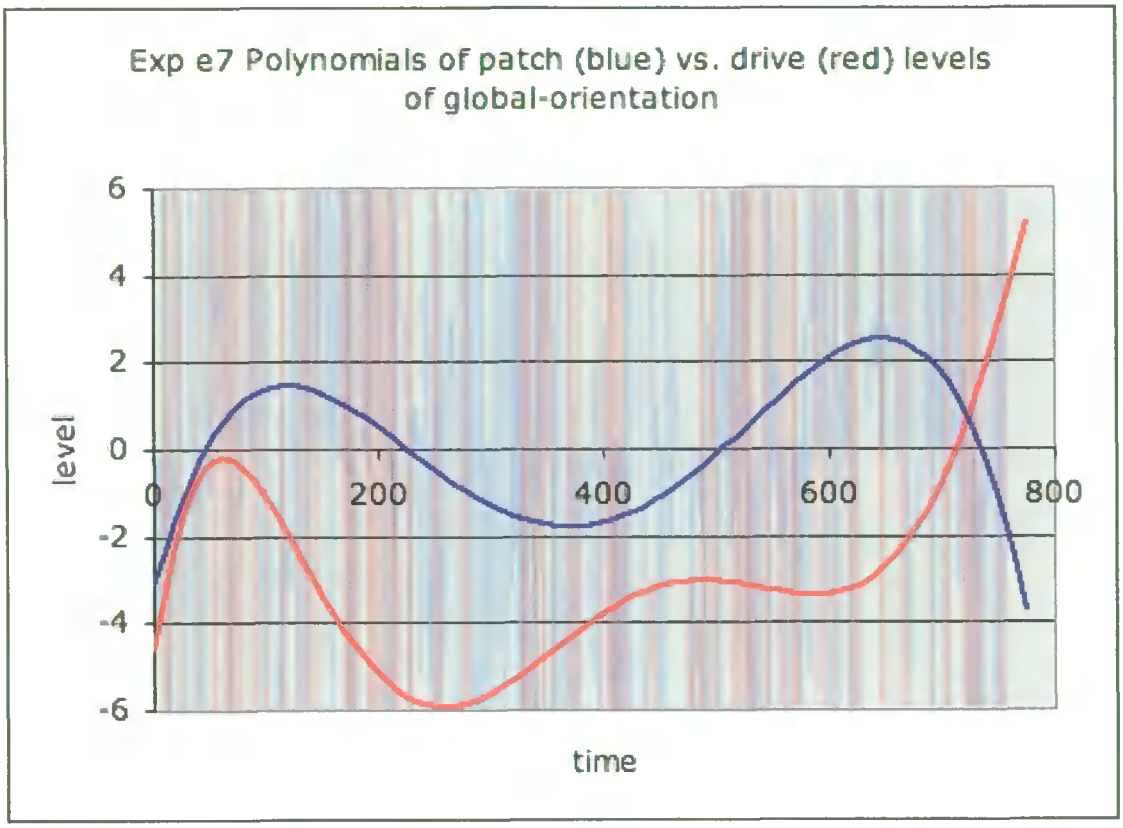


Figure 8.6: Experiment e7, patch vs. drive levels of global orientation.

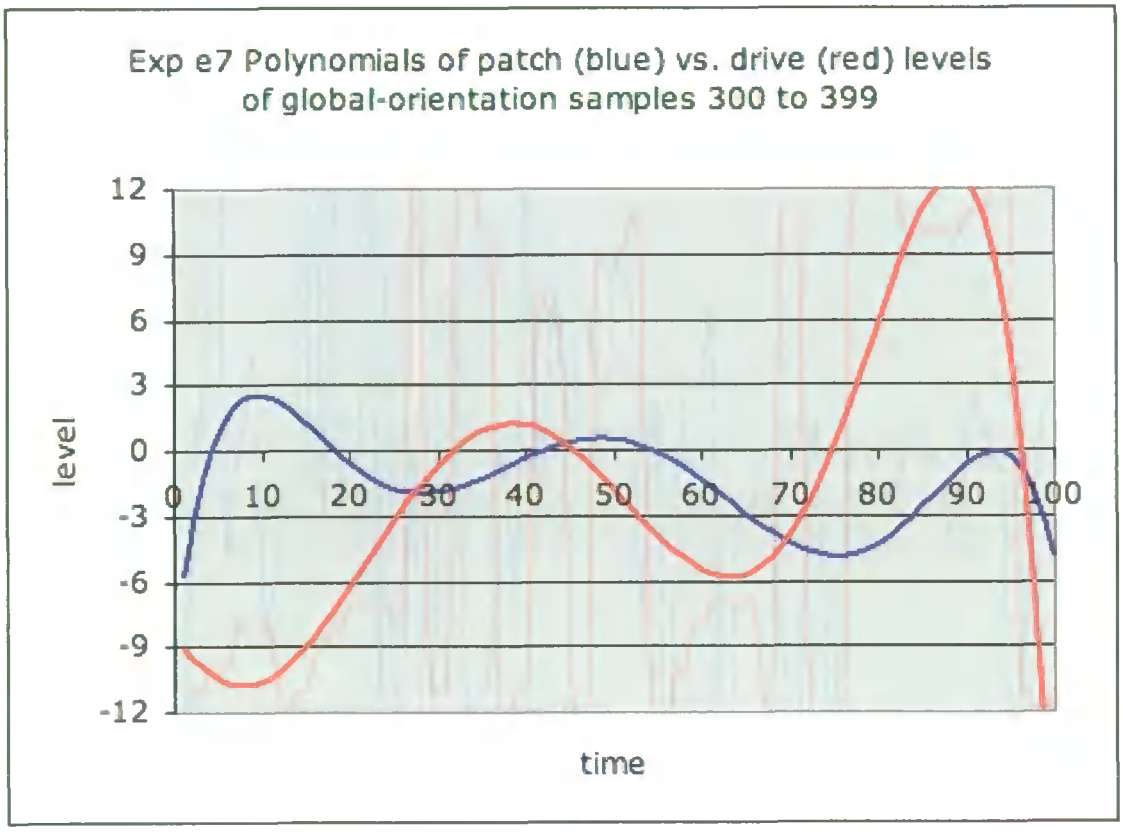


Figure 8.7: Experiment e7, patch vs. drive levels of global orientation – detail.

8.7 Optimisation of the Drives Pool

In terms of genetic evolution, the fitness of a drive is equivalent to its efficiency. The breeding procedure is similar to the procedures used with listening brains (SAN objects) and the patcher objects. Optimisation using genetic operators aims to modify the relationships inside the drives to make them better adapted to the variable external pressures; i.e., the *changes* in human-machine similarity and the *changes* in quality and quantity of the material provided by the human performer.

Breeding is organised as follows;

1) The drives population is sorted according to fitness
2) The two fittest drives are considered parents
3) A new population is created: the relationship-vectors of both parents are considered genotype and new vectors are computed using a single point crossover operator
4) A small amount of mutation is applied to all drives in the new population, mutation level is 5% in all experiments
5) All instance variables of every new drive are reset and the integration- and expression levels are set to a random centre value between 40 and 60.

8.8 Conclusion

The drive object exemplifies a dynamic structure that generates temporal machine-propositions that instruct the system to either (1) assimilate the current context suggested by the human interactor or (2) generate autonomous responses irrespective of context. The two competing orientations are identified as respectively *integration* and *expression*. A drive contains a network of relationships that provide a qualitative interpretation of impinging changes; the competing integration and expression levels

inside a drive adapt according to the nature of its relationships and the incoming signals. The efficiency of a given drive is defined as proportional to how well its relationships contribute to the achievement of its current orientation, that is, the orientation with the highest level. In addition, a drive has a learning component. Its understanding-level is updated proportional to how effective its current relationships actually are in serving its implied goal: integration or expression.

The maintenance of variable machine motivation differs sharply with explicit systems design that typically builds on conventional mapping procedures. In contrast, a drive is a flexible data structure that adapts its integration and expression levels according to its relationships and the accommodation of external changes.

Chapter 9: Interaction Tracking, Analysis and Coordination of Breeding and Learning

The present chapter details the complex chain of decision-making coordinated by Oscar's background analysis-process. The inference of *temporal interaction patterns* is described, that is, how they manage to identify pivotal moments in human-machine interplay. An adaptive algorithm is developed that aims to predict the starting and halting behaviour of a human interactor; it is instrumental in guiding the output scheduler to play according to the current machine orientation; i.e., integration or expression. Next, we address the functions that coordinate learning in the drive objects and breeding of fresh populations of all four system components subject to evolution: sensor-activator-networks, patches, compound-functions and drives.

9.1 Introduction

The analysis-process is one of the six concurrent LISP processes running in the background (please refer to chapter 3, section 3.4.2). The sampling rate of the analysis process is set between 500 and 2000 milliseconds. The analysis-process schedules the *analyse-function* consisting of a sequence of the six procedures listed next. A detailed description follows in the subsequent sections.

1) Update the info string in the GUI reflecting information on the last event performed by the human interactor: pitch, velocity, duration, pitch-interval, event counter and the result of the zones-sensor. See chapter 4, section 4.4.2.

2) The handle-ear-analysis (ear method) runs the listening network incorporating the current sensor-activator-network and the current patch – see this chapter, section 9.2.

3) Update higher-level context; four continuous values: quantity, quality, human-responsiveness and exploration-exploitation ratio. Quantity and quality levels and human-responsiveness are visualised in the GUI. See chapter 4 section 4.5.8.

4) Update the interaction-trail; the interaction-trail variable is a 16-element vector and holds the recent history documenting temporal synchronisation within the interplay of man and machine. This information is addressed to compute a number of significant temporal interaction patterns (TIP) – see this chapter, section 9.3.

5) Evaluate interaction; the program guides further decision making according to the detection of four specific types of temporal interaction patterns. The *evaluate-interaction-function* is itself a sequence of seven sub-functions as depicted in figure 9.1. See section 9.4.

6) Run predictor; the predictor attempts to predict the future start-time of the human improviser. See section 9.5.

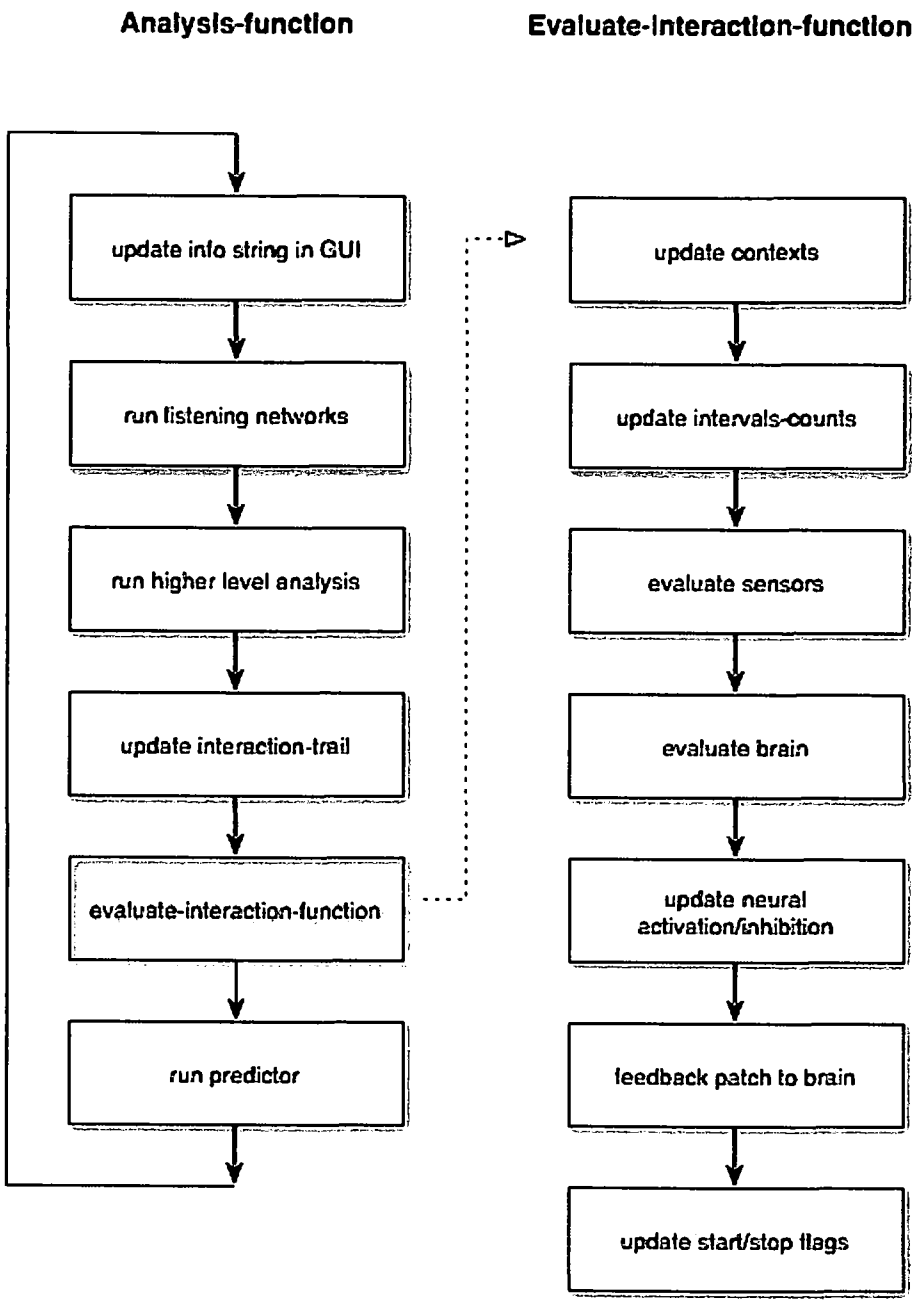


Figure 9.1: The analysis-function (on the left) is scheduled by the background analysis-process, typically every 1000 milliseconds. The evaluate-interaction-function itself evaluates a sequence of sub-functions as depicted on the right hand side.

9.2 The Top-level Analysis Function

The *evaluate-interaction-function* (ear method) procedure coordinates the listening network that consists of the current SAN feeding its results to the current patch and finally, the patch sending part of *its* output as feedback back into the SAN. A closed loop is thus created. It guarantees non-linear behaviour of the SAN/Patch tandem. Interesting oscillatory behaviour may develop because of the positive feedback and, at the same time, input from the human interactor is accommodated equally well (see chapter 5, section 5.1 for details). The *evaluate-interaction-function* function first updates the context as explained next.

9.2.1 Consideration of Context

Oscar provides six level sensors that adapt relative to the most recent input context. This context resides in the last few events entered by the human interactor and is typically much sorter than the duration of working memory – thus more accurately reporting the current state of affairs. A simple context sensitive algorithm traces the dynamics of pitch, velocity and duration of the last incoming event. The edges of the current context window are adjusted; the min and max values change, using a multiplicative operator, according whether the event parameter is above, below or inside the context window. This entails a functionality that zooms in into the most recent event. The value of the multiplier ($0.5 < m < 1.5$) controls the speed of adaptation or, in other words, the context sensitivity.

The context object is designed as follows:

```
class context
; instance-variables
name              ; dimension name e.g. pitch
status            ; above, below or inside
history           ; 100 element vector
```

```
min-value        ; low reference value
max-value        ; high reference value
```

Three context objects are instantiated respectively for handling the sensors *low-p* and *high-p* for the dimensions of pitch, velocity and duration. Consider, for example, the *update-pitch-context* method in pseudo code:

```
update-pitch-context (ear method)
    if last-pitch <  min-value
            status = 'below
            min-value = last-pitch – 2
            else
            if last-pitch > max-value
            status = 'above
            max-value = max-value + 2
            else
            status = 'inside

    ;; always
    activation = 1.0 + (remap last-velocity 1 127 0.1 0.5)
    inhibition = 1.0 - (remap last-velocity 1 127 0.1 0.5)
    min-value = (min (max-value - 2)
                        (min-value * activation))
    max-value = (max (min-value + 2)
                    (max-value * inhibition))
```

The context status equals *below, above* or *inside*, according to the whether the last input value is higher than the ceiling of the reference window, lower than the minimum value or inside the reference window. When above, the context window will expand in the upper direction in an effort to incorporate the new extreme. When below, the *min-value* will decrease in a similar attempt. Whatever the result, the

context object will try to minimise the size of its discrimination window and thus enhance its specificity. The net effect is continuous zooming behaviour of the context object.

The *activation* and *inhibition* scaling factors are proportional to the velocity of the last received MIDI input event implying that loud events will exercise more pressure and thus contribute to faster adaptation.

The *update-velocity-context* and *update-duration-context* functions are functionally equivalent to the function depicted above and accommodate the last values of the velocity and duration dimensions of the last MIDI input event, however using fixed multipliers 1.1 and 0.9 for respectively activation and inhibition. Typical behaviour of the context-tracking algorithm is visualised in chapter 4, figure 4.3.

9.2.2 Consideration of Intervals

The *intervals-counts-list* is a melody object instance variable addressed by a number of sensors that examine motion in the data in working memory. It is a 3-element vector that holds the number of pitch intervals that are negative, positive and equal zero. By comparing these three values, one may extract second order information such as eventual melodic profile or ascending or descending pitch tendencies inside that melody.

9.2.3 Handling the Listening Network: Sensors, Brain and Patch

The *handle-ear-analysis* method finally evaluates all 64 current Boolean sensors, updates the sensor graphic user interface so the colour of all LED objects reflects the status of their associated sensors and the neural-activation-inhibition display is equally updated. Next, the function *vector-input-from-san* will feedback part of the

output of the current patch back to the input of the current SAN. A detailed functional description of the listening network is given in chapter 4.

9.2.4 Detection of Halting Behaviour of the Human Improviser

Finally, the *human-considered-just-stopped-p* flag is updated. This information is put to good use in the interaction-trail inference process and the prediction algorithm. In pseudo code:

```
human-considered-just-stopped-p (ear class method)
    mel = buffer ; working memory, visualised in the ear GUI
    cur-gap = current-no-input-gap ; in milliseconds
    average-gap = (sum (gaps0 mel)) / (nr-events mel)
    (and (cur-gap > (average-gap * 1.5))
         (cur-gap < (average-gap * 1.5) + 2000)))
```

The above function attempts to collect evidence in order to potentially conclude that the human improviser can be considered "just stopped". At any time, the *current-no-input-gap* function returns the time delay since the last note-off event was received from the human improviser. We receive incremental confirmation while we wait for a long enough silence to build up. After having waited for some time, it is concluded that the human performer has indeed just played the last event of the current input sequence. This exact moment in time needs to be captured with sufficient precision. A statistical method was developed by trial and error. The function returns true if the *current-gap* falls inside a two seconds discrimination window. The window edges follow the *average-gap* scaled by 150 percent. Experiments confirm robust performance facing totally unpredictable input behaviour of the human performer, in particular, given the idiom of open, non-idiomatic improvisation (Bailey 1980).

9.3 Updating the Interaction-trail

The *update-interaction-trail* function (improviser method) computes the interaction status of the improviser; it returns a value of 0, 1, 2 or 3.

```
h = human-considered-stopped-p
m = player-finished-p          ; using global player object


result =
    if h and m
        return 0               ; both just finished
             if (not h) and (not m)
            return 3     ; both just started
                 if (not h) and m
                   return 1      ; only human just finished
            else
                   return 2      ; only machine just finished
if result not = previous-interaction-status
    previous-interaction-status = result
    ; update interaction-trail history; i.e.,
    ; shift 15 values left in interaction-trail vector
    ; set last element in vector to result
    (interaction-trail 15) = result   ; last element of 16-element vector
```

The *human-considered-stopped-p* function returns true as soon as the *current-no-input-gap* value exceeds 150 percent of the average value of all duration gaps in the ear's buffer (working memory). The trail is updated only if a transition in value occurs. Temporal interaction patterns are collected from the observation of the tail of the *interaction-trail* vector as it reflects the most recent history of the human-machine interaction pattern.

9.4 Evaluation of Human-Machine Interaction

9.4.1 Detection of Temporal Interaction Patterns

The purpose of the *evaluate-interaction* function (improviser class method) is first to check whether any specific situation occurred – for instance, if HP and MP just finished playing at the very same moment in time (given a small tolerance time window) – and, if positive, take the necessary actions accordingly. The number of noteworthy situations is reflected in a set of significant temporal interaction patterns (*TIP*). The *analysis-process* samples transitions in the activity of both man and machine and how their actions happen to synchronise in time. A pattern recognition algorithm can discriminate 16 relevant *TIP*, documented in figure 9.2. All *TIP* are organised into four functional families: detection of response behaviours of MP vs. HP and HP vs. MP, detection of the end of soloist behaviour and, finally, detection of the beginning of a new sequence being played by either HP or MP. The evaluate-interaction method will canalize the information flow into four potential paths according to the detection of a pattern belonging to one of the four pattern families.

For example, any pattern belonging to the list: ((1 0 2 x) (1 2 x) (3 2 x) (1 3 0) (1 3 1) (0 2 0)) signals a "machine just finished playing a response" message. The "x" symbol means "don't care". At this point, one can compute the new similarity between the last sequence performed by HP and the sequence that the MP just finished playing, handle history tracking and run many additional functions. A systematic, detailed description of the four significant situations follows.

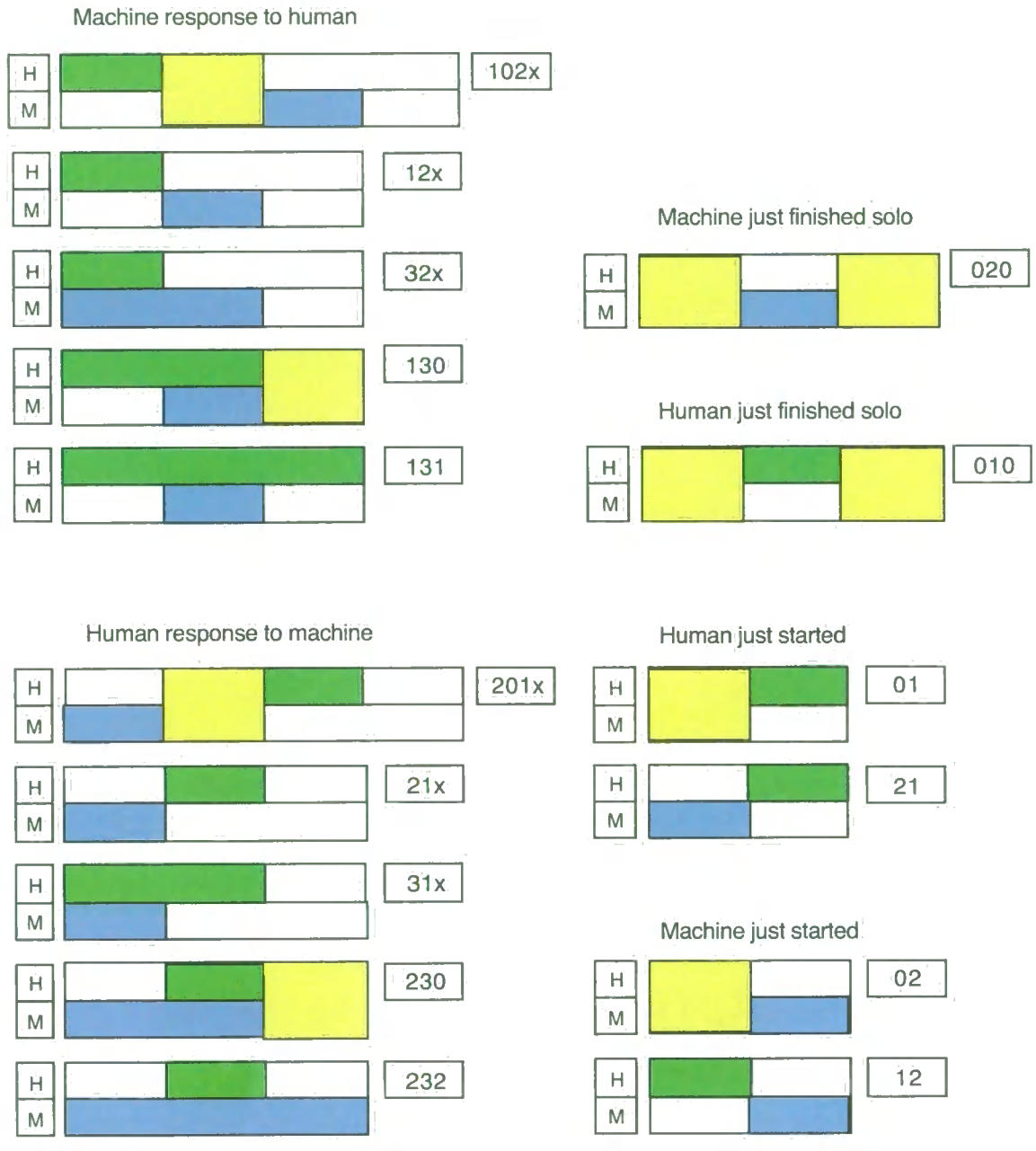


Figure 9.2: Identification of 16 significant temporal interaction micro-patterns. Activity in the human performer (HP) and the machine player (MP) is visualised as two parallel tracks, green denotes human-active, blue denotes machine-active, yellow denotes neither is active; i.e., silence.

9.4.2 Deployment of Temporal Interaction Patterns

9.4.2.1 Pattern nr. 1

We infer that the MP just finished playing its current melody when the analyser detects a *TIP* belonging to the following list: ((1 0 2 x) (1 2 x) (3 2 x) (1 3 0) (1 3 1) (0 2 0)). The following actions are then taken:

1) Compute the similarity between working memory and the current compound function (CF). The CF holds the result of its current musical processing functions in its melody object.

2) Compute the quantity and quality levels of working memory.

3) Update the *SQQ-trail*, argument equals "machine". The *SQQ-trail* is a 16 by 4 element array (used as a FIFO structure) keeping track of the sixteen most recent value sets. The four dimensions 0 to 3, keep values of respectively, similarity, quality, quantity and the symbol *HP* or *MP* – indicating whether the values were obtained by either Human or Machine activity. The information in the *SQQ-trail* – in particular the difference between the last two entries – is consulted throughout the program. The signed difference between the last two entries may provide evidence of *MP* and *HP* getting closer together or drifting further apart.

4) Adjust the fitness of the current compound function according to the difference between current and previous similarity. In addition, the current orientation (integration or expression) of the current drive is considered, it acts as a reference to judge the usefulness of the changes in similarity. See chapter 8 for a detailed description.

5) Adjust the *machine-agreement* vector (improviser class instance variable) as a function of the difference in HP-MP similarity. This two-element vector keeps a two-step history of machine-agreement. The rationale is simple; if the current similarity is higher that the previous similarity, the current machine agreement is scaled by a factor of 1.25 otherwise; it is scaled by a factor of 0.75. The functionally related vector is *human-agreement.*

6) Run the current drive (improviser class method). This method holds two actions: first compute the new activation levels for the competing suggestions (integration and expression) inside the current drive object as a function of the changes in quality, quantity and similarity and, second, compute the efficiency of the current speculation (integration or expression) of the current drive given the change in similarity. Please refer to the details in chapter 8.

7) *Handle-main-function* (improviser class method). This method handles the timing of learning and breeding cycles. For clarity, it was decided to run the main function only when the machine just finished playing a response. The understanding-level of the current drive is adjusted according to the common-understanding level (improviser class method), a value between -100 and

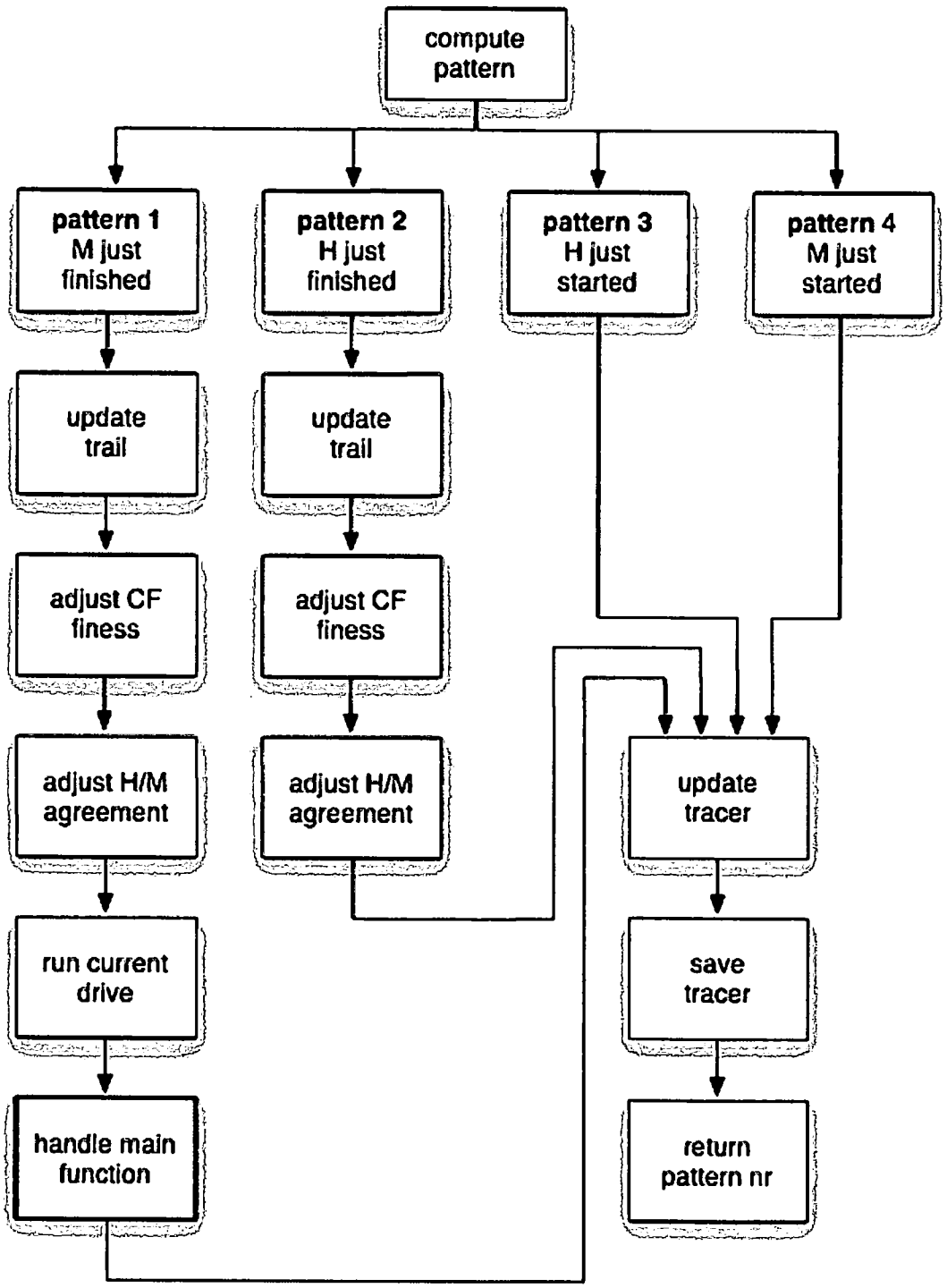


Figure 9.3: Four significant interaction patterns are filtered out of a continuous interaction stream. These patterns signal the start and stop activity in both human and machine and trigger specific further processing.

+100, reflecting a scale of human-machine understanding from 100 percent conflict to 100 percent agreement. With values around zero, no clear opinion is available. Details on the *handle-main-function* function are in section 9.7.

9.4.2.2 Pattern nr. 2

It is inferred that the human interactor just finished playing his response to the most recent machine generated melody, when any temporal interaction pattern is detected belonging to the following set: ((2 0 1 x) (2 1 x) (3 1 x) (2 3 0) (2 3 2) (0 1 0)). In other words, one concludes that the human is considered "just stopped" – this moment in time of transitive value, has to be captured. A variety of real-time segmentation algorithms were implemented to chase this instant in time (please refer to chapter 4, section 4.5.7). The subsequent actions are taken:

1) The last-*sequence-just-stopped* flag (ear instance variable) is set. This flag is consulted in the prediction algorithm that tries to predict the exact moment when the HP will start playing in the future.

2) Update the *SQQ-trail*, argument equals "human". Functionally equivalent to step 3 in pattern nr. 1.

3) Adjust the fitness of the current compound function according to the difference between current and previous similarity. The *delta-similarity* here is the result of human initiative – in contrast to the equivalent action taken in step 4 of pattern nr 1 where the effect follows from the most recent machine-generated initiative.

4) Adjust the *human-agreement* vector, similar to step 5 in pattern nr. 1.

9.4.2.3 Pattern nr. 3

Any pattern in the list ((0 1) (2 1)) indicates that the HP is performing the first event of a new sequence. The *last-sequence-just-started* flag (ear instance variable) is set. It is consumed and reset in the prediction algorithm briefly introduced above and fully documented in chapter 9.

9.4.2.4 Pattern nr. 4

Any pattern in the set '((0 2) (1 2)) reflects that the machine just started playing. This pattern is slightly redundant because one has explicit control over the global player function – a function written on top of the functionality provided by Common Music (Taube 2004) and Midishare. (Orlaley et al. 2004). However, it was kept for completeness and possibly future use.

9.4.3 Analysis of Temporal Interaction Patterns

Figure 9.4 shows a typical histogram of temporal interaction patterns in a given experiment. In short, this experiment uses a test-player; an internal sequence generator that activates the system for test purposes in the absence of a human performer. The occurrence of the first 14 basic micro-patterns from which the three resulting patterns (pattern nr.1 to pattern nr. 3) are extracted is examined.

Two micro-patterns dominate: (1 2 x) meaning "machine just finished" and (2 0 1 x) meaning "human just stopped".

When summing the number of occurrences of the first three groups of micro-patterns we get respectively: 189, 183 and 27. Thus, a grand total of 399 micro-patterns were captured. The frequency of pattern-1 signalling "machine just finished"

and pattern-2 signalling "human just finished" is nearly the same. Pattern-3 meaning "human just started" is detected only 27 times. This low value reflects a low ratio of the play/rest probability in the test-player.

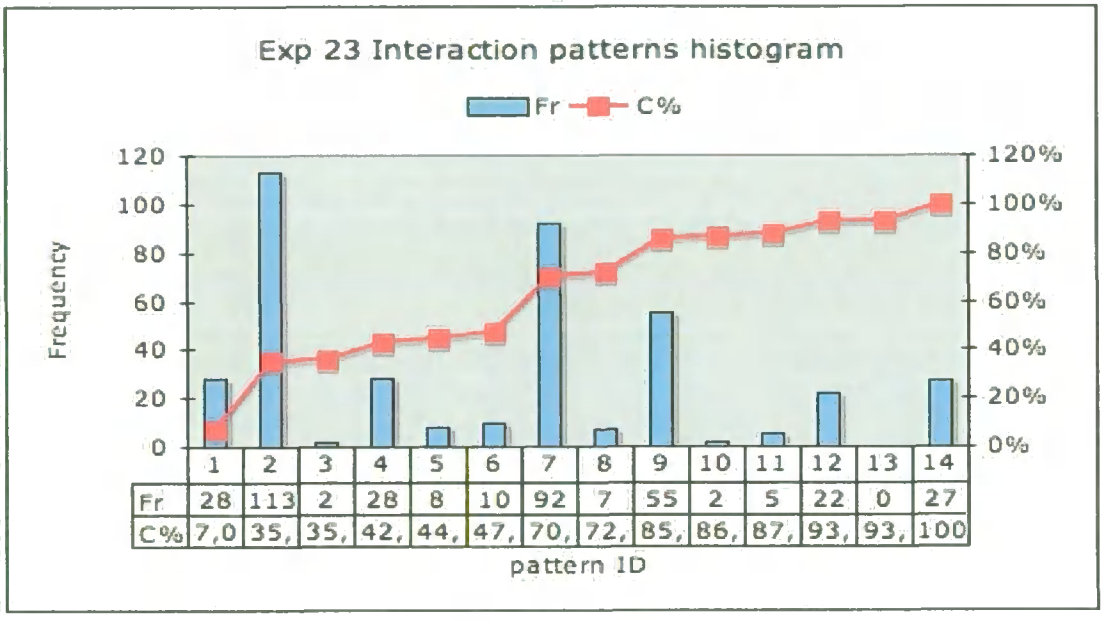


Figure 9.4: Histogram of temporal interaction patterns.

9.5 Prediction and Interaction Scheduling

The purpose of the *predictor* algorithm is to perform forward planning. The results of the predictor are consumed in a concurrent LISP process responsible for coordinating MIDI output (documented in chapter 3, section 3.4.2). The scheduler is organised as to guarantee optimal control of machine playing behaviour. For instance, if the machine player (MP) aspires to integrate with the human performer (HP) the rationale is to anticipate that HP and MP start playing together at the next occasion. Thus, the future start-time of the HP should be predicted so that the next MP response can be scheduled exactly at that point in time.

In case the MP aims for expression, it must wait until the exact moment that HP has finished performing; the MP will start playing at the predicted-stop-time of HP.

Whatever the orientation of the MP, the MP response melody will only start after waiting for a delay determined by the current value of the *predicted-start-time* or the *predicted-stop-time*. The adaptation thrives on three instance variables: *predicted-sequence-duration, predicted-start-time* and *predicted-stop-time* all relating to the HP.

```
run-predictor (improviser-class method)

now = (get-time)          ; current time
qt = quantize now 1000    ; current time quantized to 1 second
qstop = quantize predicted-stop-time 1000
qstart = quantize predicted-start-time 1000

; Human performer just-stopped?

when last-sequence-just-stopped ; consume the flags set by other function
      ; evaluate the present
      if (qt < qstop)
            ; Duration shorter than predicted: qt - predicted-stop-time
        predicted-sequence-duration =
                predicted-sequence-duration * 0.80
    else

         if (qt > qstop)
        ; Duration longer than predicted: now - predicted-stop-time
        predicted-sequence-duration =
                predicted-sequence-duration * 1.20
       else

        ; Correct prediction: now - predicted-stop-time

      ; predict next stop time
      ; when the human starts, the machine makes a prediction of
      ; how long the human will play
```

```
    predicted-stop-time = now + predicted-sequence-duration

; Human performer just-started?

when last-sequence-just-started   ; consume the flags set by other function
      ; evaluate the present
      if (qt < qstart)
            ; Delay shorter than predicted: qt - predicted-start-time
        predicted-sequence-duration =
                predicted-sequence-delay * 0.80
    else

         if (qt > qstart)
        ; Delay longer than predicted: now - predicted-start-time
        predicted-sequence-duration =
                predicted-sequence-delay * 1.20
       else

        ; Correct prediction: now - predicted-start-time

    ; predict next start time
    ; when the human stops, the machine makes a prediction of how long
    ; the human will be silent
    predicted-start-time = now + predicted-sequence-delay
```

The above pseudo code shows two possible points where adaptation takes place.

1) The *last-sequence-just-stopped* function signals that the HP just performed the last event of the current last-sequence; i.e., the contents of short-term-memory are considered completely updated. Oscar checks whether the quantised current time is lower, higher or exactly equals the stop-time. Of course, quantisation to one second also implies a discrimination window of

just one second – which is quite a demanding constraint. According to the conclusion, the predicted-sequence-duration is slightly scaled using a multiplicative operator.

2) In case *last-sequence-just-started* function returns true, Oscar verifies whether the predicted-sequence-delay requires adaptation. If the HP started premature, the predicted-sequence-delay should be decreased, in contrast, when the HP does not start a new sequence before the delay expires, that delay should be increased accordingly. Whatever the outcome, the predicted-start-time is incremented by the currently computed delay.

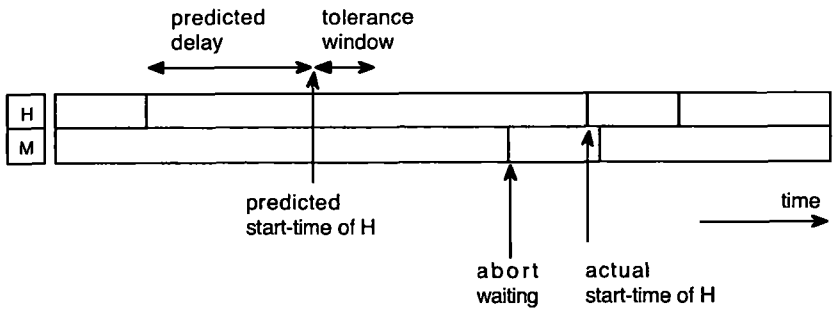


Figure 9.5: Motivation equals integration, erroneous prediction, human performer starts later than predicted.

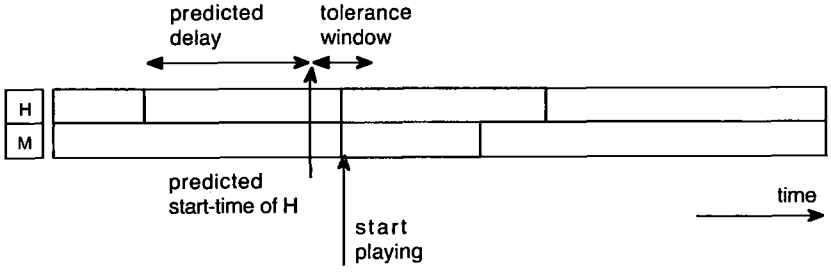


Figure 9.6: Motivation equals integration, correct prediction of start-time of the human performer.

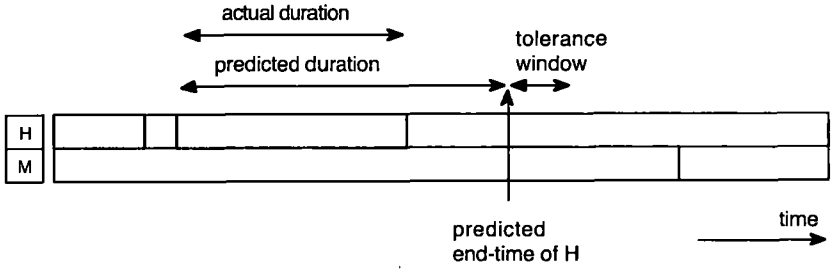


Figure 9.7: Motivation equals expression, erroneous prediction, overdue machine reaction.

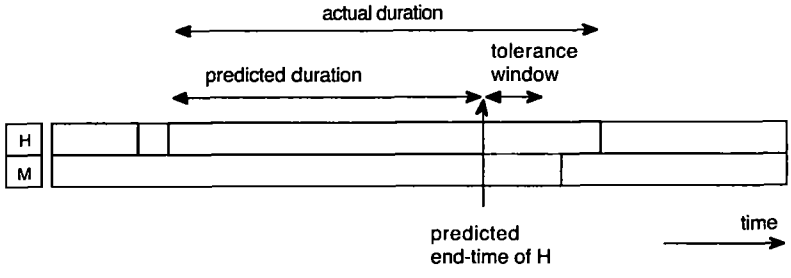


Figure 9.8: Motivation equals expression, erroneous prediction, premature machine reaction.

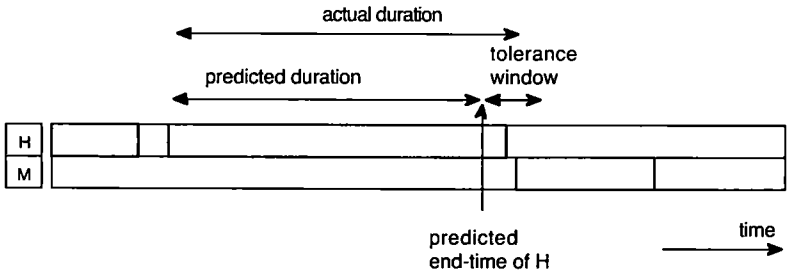


Figure 9.9: Motivation equals expression, correct prediction of the duration of the human performer's response.

The results of the prediction algorithm are helpful to coordinate the playing activity in the players agency. A LISP background process, the *improviser-player-process*, computes the exact moment in time when a machine response fires off (see implementation details in chapter 3). In LISP-like pseudo code:

```
if current-orientation equals integration
     waiting-time = (cm::midigettime) - (predicted-start-time self)))
     if (cm::midigettime) > predicted-start-time
          (format t "~% Integration-play, ~12a min-sec-msec behind
                    predicted START time." waiting-time)
               (run-player-agency)

        if (abs waiting-time) > *max-waiting-time*)
          (progn (format t "~% Integration-play, abort waiting")
               (run-agency)
          else
          (format t "~% Integration-play, continue waiting: ~d" waiting-time)

else
;; current-orientation equals expression

     waiting-time = (cm::midigettime) - predicted-stop-time
        if (cm::midigettime) > predicted-stop-time
(format t "~% Expression-play, ~12a min-sec-msec behind
     predicted STOP time." (cm::midigettime) - predicted-stop-time
                (run-player-agency)
```

```
if (abs waiting-time) > *max-waiting-time*
     (format t "~% Expression-play, abort waiting")
     (run-agency)
     else
     (format t "~% Expression-play, continue waiting: ~d" waiting-time)
```

When *current-orientation* equals integration, the player agency waits until the expected start-time of the human player, however with a tolerance of **max-waiting-time**; when the HP remains silent the run-agency function will fire anyway. A similar functionality is available to keep track of the expected stop time of the HP. The **max-waiting-time** global variable is typically 5000 milliseconds.

9.6 Coordination of learning and evolution

The handle-main-function function (highlighted in figure 9.3 and detailed in figure 9.10) plays a central role in timing the learning and breeding activity in the system. The following sequence of actions takes effect. First, the understanding-level of the current drive is updated according to the current system-common-understanding; i.e., the degree of conflict or agreement that typifies the current interaction (chapter 8, section 8.6). Two interlocking counters are considered next: the breeding-counter and the learning-counter – a schematic representation is given in chapter 10, figure 10.1. Learning activity is addressed first.

9.6.1 Coordination of Learning

When it is time to *learn*, the evolution-data collector is put to work, the data block summarized in chapter 10, table 10.2 is saved to disk with a header equal to *Learn* – when breeding the header equals *Breed* – the headers provide additional information as to at what point in time a given data block was acquired.

The *exploration-exploitation-ratio* is computed next. When learning-mode equals *explicit*, the current value of the performance-counter, as it moves from zero to 100 during any experiment, will function as exploration-exploitation-ratio. When the learning-mode is *real-time*, the ratio is computed from the momentary relationship between the system's accumulated values of *exploration-pressure* and *exploitation-pressure* as follows:

```
(setq exploitation-exploration-ratio
     (round (if (> (exploitation-pressure self)
                   (exploration-pressure self))
          (- 100    (* 100 (/ (exploration-pressure self)
                       (exploitation-pressure self))))
          (* 100    (/ (exploitation-pressure self)
                       (exploration-pressure self))))))
```

The *exploration-exploitation-ratio* plays a crucial role in the selection procedures that address four populations of evolved objects: the collection of sensor-activator-networks held inside the brain object, the collection of patches held inside the patcher, the drives inside the drives-pool and finally, the list of compound-functions inside the compound-function-pool. The objective is to select the *best* object – not necessarily the fittest but the most *promising* one; this makes the difference between counting on short-term effectiveness; i.e., selecting the fittest through the act of exploitation – or, in contrast, expressing trust in exploration; i.e., hoping to discover even fitter objects by random selection. The exploration-exploitation-ratio controls the balance between the relevance of both alternatives.

The value of the exploration-exploitation-ratio acts as a probabilistic specification for either choice; when random(100) < exploration-exploitation-ratio then exploit else explore.

Exploitation can only be successful when enough objects in the current population have managed to develop non-zero fitness levels – at least half of the population size in most of my experiments. Too few fit objects entail the risk of premature convergence; the search gets stuck in a local minimum of the fitness landscape. In practice, the list of objects is sorted according decremental fitness; as a result the first two objects of the sorted list deliver genetic material to breed the next population. In case of insufficient fit candidates, exploration takes effect anyhow. *Exploration* simply selects random parents without considering fitness.

The selection scheme described so far is applied to sensor-activator-networks and patches. The selection process of the third kind of objects – the drives – is, however, slightly more complex. Since drives are objects endowed with the faculty to learn, they receive particular treatment as described in chapter 8, section 8.5.

Finally, every agent in the players agency will be associated with a given compound-function as selected from the compound-function-pool, this process is pictured in chapter 7, figure 7.1. A function called *set-agents-CF-according-understanding* creates a list of potential CF by considering the current orientation of the current drive, for instance, if that drive aims *integration*, the algorithm will search for CF that have proven to provide functionality for narrowing the musical gap between man and machine; i.e., CF that have developed high *integration-fitness* levels during previous interactions.

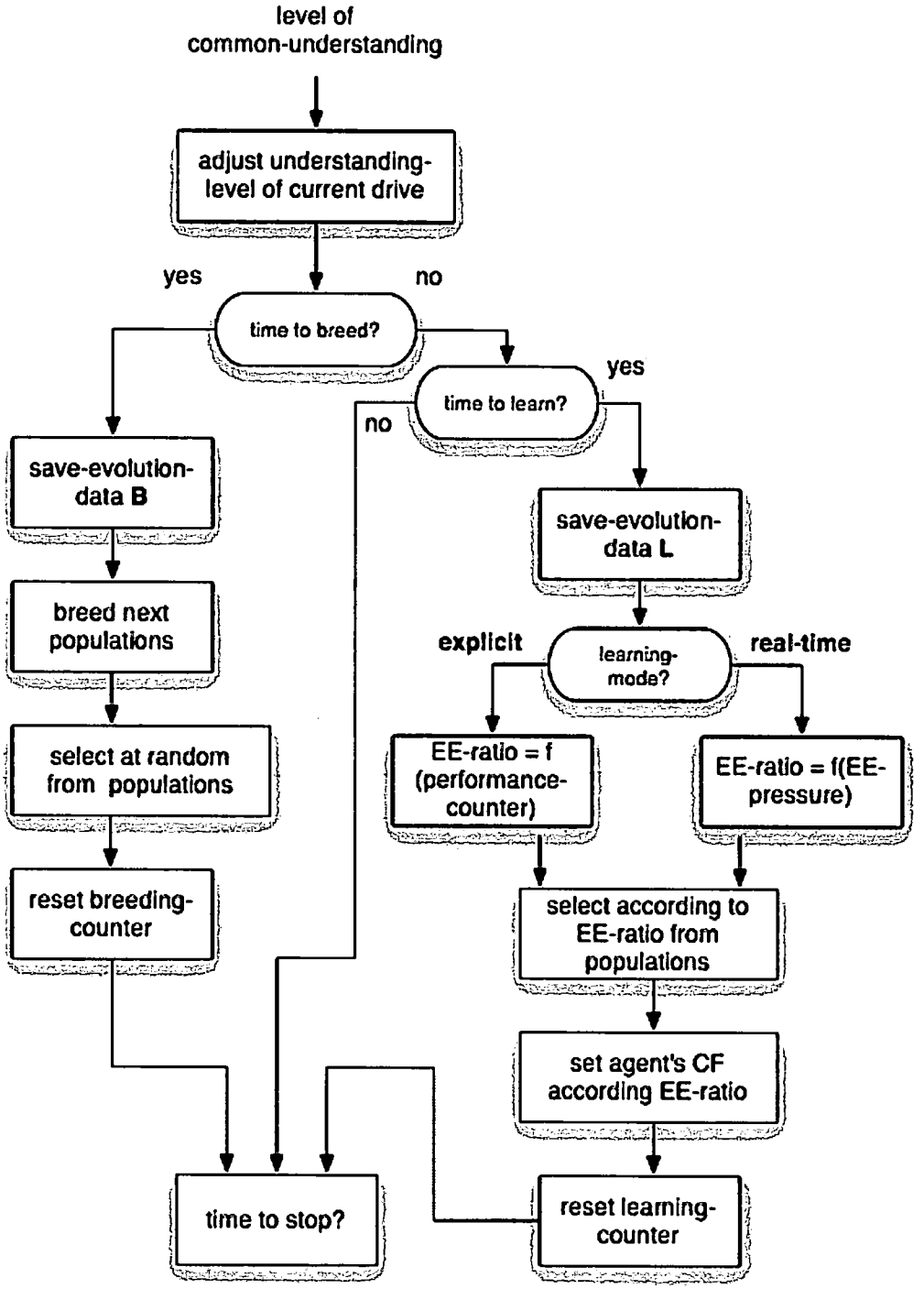


Figure 9.10: Handle-main-function coordinates timing of learning and breeding activity, the selection of fresh instances from the respective populations and the distribution of compound-functions to the players-agency.

It is assumed the drive effectively developed an orientation; the contrast between integration-level and expression-level must indeed be at least 10%. In case the drive's orientation equals NIL, CF are distributed at random.

Otherwise, CF's are distributed according the *select-specific-function* algorithm (figure 9.11) that takes four arguments:

1) Exploration-exploitation-ratio: as detailed at the start of the current section.

2) The current global orientation of the current drive: a value between -100 and 100, indicating a range from total expression to total integration.

3) A *stringency* parameter: a value between 0 and 100, it specifies which particular objects will be selected from the list of potential fit candidates. In other words, the list of objects with positive fitness is further restricted as to provide an indication of the required relative fitness strength of the candidates. In case stringency equals zero, any fit candidate may be selected, irrespective of its actual fitness level. In case stringency equals 100, only the fittest candidate is subject to election. When stringency equals 50, only the first half of the candidates can be subject of further selection.

4) Operation mode: either *linear* or *random*. Given a linear mode, for instance, when stringency equals 100, the fittest object will be chosen, when zero, the item with lowest fitness is taken, when stringency equals 50, the median value of the list is selected. When mode equals random, the algorithm creates a partial list of all candidates – the length of that list being proportional to stringency.

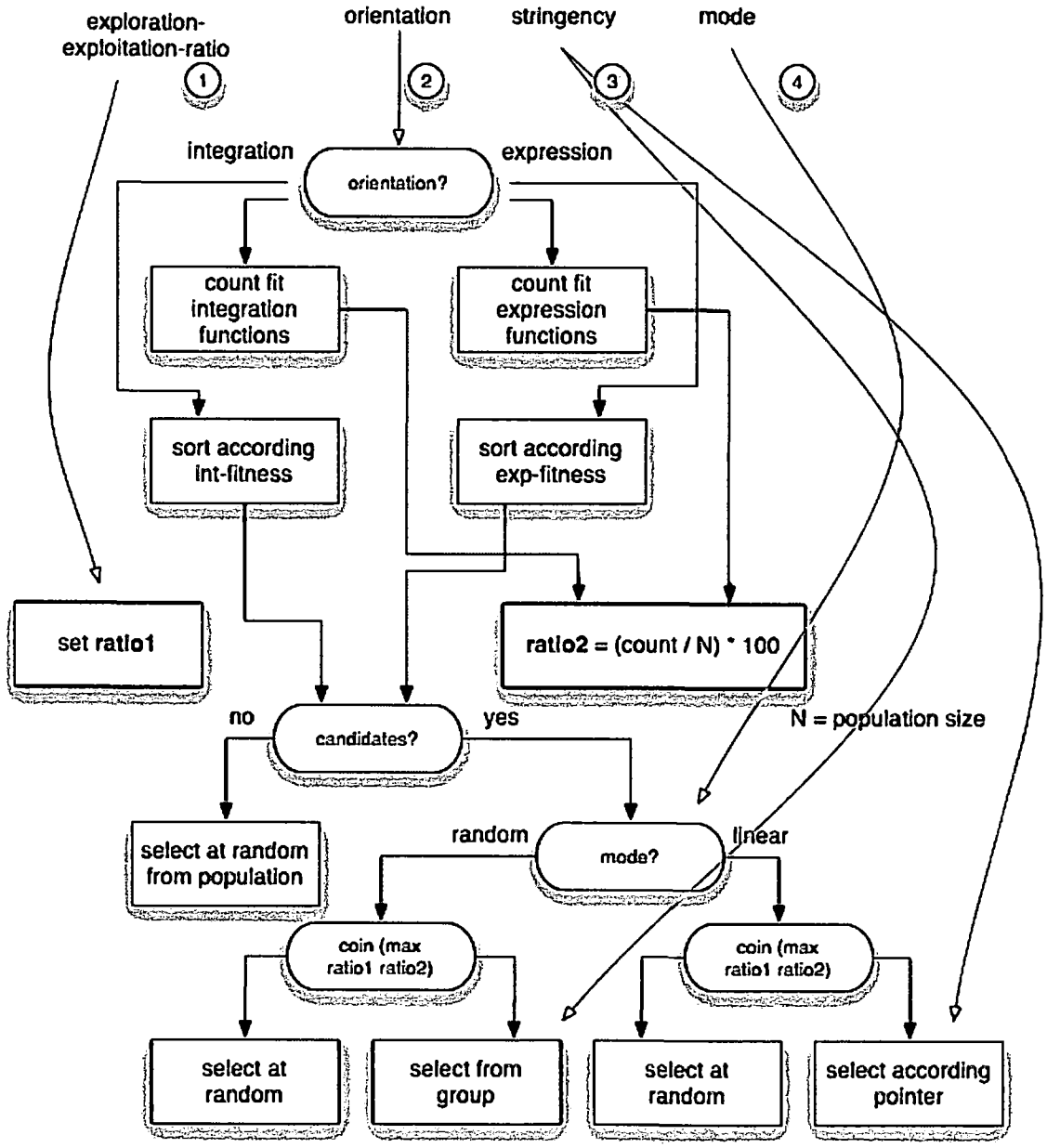


Figure 9.11: Select-specific-function: the algorithm selects a compound-function from the compound-function-pool according to four parameters.

Note that the parametric *exploration-exploitation-ratio* is set as *ratio1*. It is competing for attention with a second, internal *ratio2*, it is computed as follows:

```
ratio2 = (number-of-fit-compound-functions / population-size ) * 100
```

Ratio2, in percent, is thus relative to the number of CF that succeeded in developing non-zero fitness levels. The ratio that becomes effective is computed as:

```
actual-ratio = (max ratio1 ratio2).
```

In other words, when facing a highly fit CF-pool, *ratio2* will supersede *ratio1*; i.e., exploitation will be favoured over exploration – irrespective of the value of ratio1. In contrast, given a low percentage of fit CF in the CF-pool, the external exploration-exploitation-ratio might be decisive. The rationale is to condition selection by the highest ratio on hand.

9.6.2 Coordination of Breeding

Let us now consider *reproduction* when the breeding-counter instructs the system it is time to breed; i.e., *time to breed?* in figure 9.10 fires *yes*. The evolution-data is first saved including a *Breed* header. The genotype in all four populations (sensor-activator-networks, the patcher, the drives-pool and the CF-pool) is now subject to crossover and mutation, a random object is selected from the fresh respective populations, a number of bookkeeping counters are reset and the system embarks on the next evolutionary epoch – until the end of the experiment; i.e., when the performance-counter hits its maximum value. Please refer to the individual chapters for more details on the respective breeding processes.

9.7 Conclusion

Four temporal interaction patterns manage to organize decision-making according to starting and halting behaviour of both human and machine. However, pattern number 1, indicative of *machine-just-finished,* plays a special role as it channels the *handle-main-function,* a function that coordinates the timing of learning and breeding.

Experiments prove that the prediction algorithm is quite successful in adapting to large variations in articulations of human input. Consider the following LISP listener trace from experiment e10 (fully documented in chapter 10), an experiment featuring human input (not the internal test-player):

Excerpt of information trail as printed in the LISP listener:

```
Improviser common understanding: 19 percent CONFLICT.

Expression-play, 2 sec 435 msec behind predicted STOP time.

    Segmentation process -> Beginning of new phrase iot: 479, phrase-
    detection-threshold: 407.

HUMAN just finished: Pattern detected: (2 0 1 -1) -- time: 290847.
Duration longer than predicted: 3411 msec.

MACHINE just finished: Pattern detected: (1 2 -1)  -- time: 296325.
Improviser common understanding: 32 percent AGREEMENT.

    Segmentation process -> Beginning of new phrase iot: 265, phrase-
    detection-threshold: 194.

Improviser common understanding: 24 percent AGREEMENT.

Expression-play, 2 sec 608 msec behind predicted STOP time.

    Segmentation process -> Beginning of new phrase iot: 390, phrase-
    detection-threshold: 304.

HUMAN just finished: Pattern detected: (3 1 -1) -- time: 300570.
Duration longer than predicted: 3344 msec.
```

Segmentation process -> Beginning of new phrase iot: 752, phrase-detection-threshold: 652.

MACHINE just finished: Pattern detected: (3 2 -1) -- time: 304395.

Improviser common understanding: 25 percent AGREEMENT.

HUMAN just finished: Pattern detected: (2 3 0) -- time: 304968.
Duration shorter than predicted -4243 msec.

HUMAN just started: Pattern detected: (0 1) -- time: 305502.
Delay longer than predicted 13292 msec.

Segmentation process -> Beginning of new phrase iot: 620, phrase-detection-threshold: 611.

HUMAN just finished: Pattern detected: (0 1 0) -- time: 307163.
Duration shorter than predicted -5113 msec.

Improviser common understanding: 28 percent CONFLICT.

HUMAN just started: Pattern detected: (0 1) -- time: 308216.
Delay shorter than predicted -2076 msec.

Improviser common understanding: 27 percent CONFLICT.

MACHINE just finished: Pattern detected: (3 2 -1) -- time: 311577.

Improviser common understanding: 32 percent CONFLICT.

Segmentation process -> Beginning of new phrase iot: 1031, phrase-detection-threshold: 1000.

HUMAN just finished: Pattern detected: (2 3 2) -- time: 314882.

MACHINE just finished: Pattern detected: (3 2 -1) -- time: 315428.

Improviser common understanding: 53 percent CONFLICT.

HUMAN just finished: Pattern detected: (2 3 0) -- time: 315985.
Duration shorter than predicted -4780 msec.

Improviser common understanding: 23 percent AGREEMENT.

HUMAN just started: Pattern detected: (0 1) -- time: 316539.
Delay longer than predicted 4434 msec.

Improviser common understanding: 25 percent AGREEMENT.

MACHINE just finished: Pattern detected: (3 2 -1) -- time: 319328.

Improviser common understanding: 53 percent AGREEMENT.

Segmentation process -> Beginning of new phrase iot: 1079, phrase-detection-threshold: 1000.

HUMAN just finished: Pattern detected: (3 1 -1) -- time: 321005.
Correct prediction: error is 322 msec.

Improviser common understanding: 69 percent AGREEMENT.

HUMAN just started: Pattern detected: (0 1) -- time: 321617.
Correct prediction: error is 604 msec.

Duration: 0 min 5 sec 100 msec.

MACHINE just started: Pattern detected: (1 2) -- time: 322119.

MACHINE just finished: Pattern detected: (1 2 -1) -- time: 322636.

```
Improviser common understanding: 62 percent AGREEMENT.

    Segmentation process -> Beginning of new phrase iot: 1045, phrase-
    detection-threshold: 1000.

HUMAN just finished: Pattern detected: (2 3 2) -- time: 325463.
Correct prediction: error is -238 msec.

Improviser common understanding: 53 percent AGREEMENT.

MACHINE just finished: Pattern detected: (3 2 -1)  -- time: 327065.

Improviser common understanding: 51 percent AGREEMENT.

MACHINE just started: Pattern detected: (0 2) -- time: 331910.

Improviser common understanding: 37 percent AGREEMENT.

Expression-play, 4 sec 69 msec behind predicted STOP time.
```

It is tempting to echo minute information reflecting the behaviour of each and every algorithm comprising the system, however, such an approach yields a massive, overwhelming amount of data that proves very difficult to follow and interpret while the system is running.

The trace above is a *filtered* list of information; it includes the current degree of human-machine understanding (improviser common understanding), the kind of orientation of the machine responses – *expression-play* means a given player agent performs a compound-function that has the intention to express itself irrespective of the human-suggested context, information on the background input segmentation-

process that runs independently of the analysis-process and, finally, information on starting and stopping activity in man and machine.

The behaviour of the prediction algorithm is highlighted by red text. The quantisation of the clock times in experiment e10 equals 2 seconds; the system thus adapts until the intended start and/or stop times fall within an error-zone of less than 2 seconds. The precise moment the human will *stop* playing his/her current sequence is accurately predicted the first time – with an overestimation of 622 milliseconds, the last correct prediction yields an underestimation of 238 milliseconds.

Also observe the delicate balance in degrees of human-machine understanding; extended zones of continuous values of agreement or conflict intermingle with sudden discontinuities, typical behaviour of a complex dynamical system indeed.

Chapter 10: Experiments

The objective of the experiments documented in this chapter is to offer an analysis of general system behaviour from a two different perspectives. Firstly, we are interested to study the overall behaviour of the system components, how the three major sub networks (figure 1.1) individually behave in time and how they mutually interact. We look for macroscopic patterns that may emerge as the system evolves over time. Secondly, on a more microscopic level, the actual musical output of the system is analysed. We examine the relationship between the various systems parameters that have direct impact on the nature of the outcome of the player agency.

10.1 Organization of the experiments

We first comment on a significant series of systematic experiments conducted to examine and characterize the behavioural complexity of the system relation to different high-level system parameters. All experiments take place in a given time frame; a timing scheme was developed to run experiments and collect behavioural data at specific points in time. Referring to figure 10.1 – a single performance cycle includes a number of breeding cycles and every breeding cycle contains a number of learning cycles.

Let us first address the system parameters:

1) The *performance-mode* affecting the management of the drives objects: the application of evolution only or a combination of evolution and learning (chapter 8, section 8.5).
2) In case learning is switched on: the type of learning can be either *explicit* or *real-time*. The learning type instructs what information is used to compute the *exploration-exploitation-ratio* – this ratio (0 to 100%) acts as a stochastic

threshold to select either from fit members of a given population (exploit) or search for possible even fitter individuals (explore). Given explicit learning, the linear progression depicted in figure 10.1 is employed – the ratio corresponds to the performance-pointer as it moves from 0 to 100 during the course of the experiment. Real-time learning implies that the exploration-exploitation-ratio is sensitive to the momentary relationship between Oscar's present *exploitation-pressure* and *exploration-pressure*. It is computed as follows:

```
(setq exploitation-exploration-ratio
      (round (if (> (exploitation-pressure self)
                    (exploration-pressure self))
                 ;;
                 (- 100 (* 100 (/ (exploration-pressure self)
                                  (exploitation-pressure self))))
                 ;;
                 (* 100 (/ (exploitation-pressure self)
                           (exploration-pressure self))))))
```

3) The timing of the experiments: the number of process cycles employed for learning, evolution and performance. Referring to figure 10.1, the learning-cycle equals 2 implying that the present system objects (the current sensor-activator-network, patch, drive and compound-function) are all applied twice; i.e., during two consecutive process cycles. The breeding-cycle equals 10 meaning that potentially 10 different objects can be selected from the respective object pools.[3] The ratio of the performance-cycle and the breeding-

[3] The relationship between the pool size (eight to maximum 30 objects, typically eight to 16) and the breeding-cycle influences the degree to which the individual objects in the pool are actually put to work. For example, given a breeding-cycle of 8 and a pool

cycle determines the amount of genetic activity in the system; thus, in figure 10.1, the number of evolutionary epochs is 10.

4) The top-level inclination of Oscar: *selfish* or *social.* For details, please refer to chapter 6, respectively figure 6.10 and figure 6.11.

5) Two input options: two types of input stimuli are employed to activate the listening network. Some experiments capture external MIDI events input by a *human performer*, some use an internal *test-player*. The test-player addresses the following bottleneck: since our computational strategy is based on selectionism driven by human-provided aesthetic criteria, one would seemingly require long lasting sessions of human-machine interaction while one gathers behavioural data for later, offline analysis. The test-player makes it feasible to run unsupervised experiments for many hours. The test-player acts as an internal generator of MIDI sequences sent directly to the current sensor-activator network. Two types of generative algorithms were investigated; a simple Lindenmayer-systems based algorithm (Prusinkiewicz and Lindenmayer 1990) and a generator using weighted randomness. Both algorithms offer a simple means to generate a wide spectrum of different MIDI sequences while still operating in a stylistically coherent niche. The output of a human performer is also considered coherent in terms of style, though obviously equally subject to utterly unpredictable changes in behaviour. Since our implicit aim is to explore of structural coupling in a collection of interacting networks, the assumption is that the actual nature of the input

---

size of 8, statistically speaking, all pool objects have equal chance to be chosen just once though many may apply many times while some may not apply at all.

sequences is less critical than the structural changes they create inside the system. Therefore, we focus on charting the systems' behavioural complexity rather than studying the effects of exhaustive exploration of the state space of all potential input sequences.

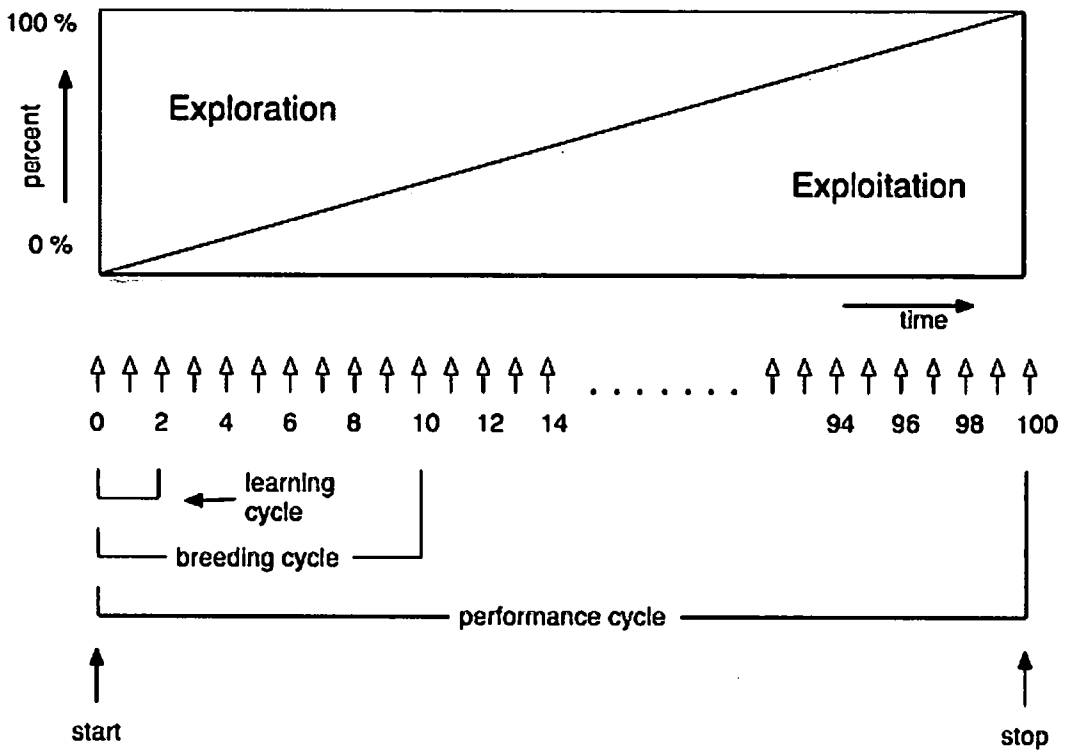


Figure 10.1: Prototypical experimental setup showing a gradual transition from only exploration towards novel resources at the beginning to only exploitation of existing resources at the end.

Figure 10.1 shows a schematic overview of the orientation of the activity in a genetic selectionist algorithm. These algorithms are operational in the SAN, the patcher, the compound-function-pool and the drives objects.

Note that the actual duration of the performance-cycle depends on the musical material produced by man and machine and the actual scheduling activity as coordinated by the prediction algorithm described in chapter 9, section 9.5.

10.2 Data Acquisition Methods

Oscar contains functionality to trace the internal system activity by monitoring a large number of vital systems variables. Two data acquisition modules are in operation: (1) the tracer-data collector algorithm (in short, the *tracer* module) and (2) the evolution-data collector. The accumulated content of both data structures is saved as individual data files targeting posterior off-line analysis. Let us first address the tracer-data.

10.2.1 The Tracer-Data Collector

The *tracer* software module collects and visualises its data in a private interface containing nine panes (figure 10.2). The tracer offers immediate inspection of a number of critical systems variables.

The data logging activity is handled by Oscar's analysis-process, a background LISP process that typically wakes up every second. However, the next data sample is collected only when a new interaction pattern appears. As explained in chapter 9, interaction patterns signal significant events, for instance, when man and/or machine just finished performing their most recent sequence. These potentially pivotal points during interaction are considered instrumental for tracking changes in huhuman-machine melodic distance as well as changes in many system components. The system is fast enough to write samples to disk without jeopardizing real-time performance.

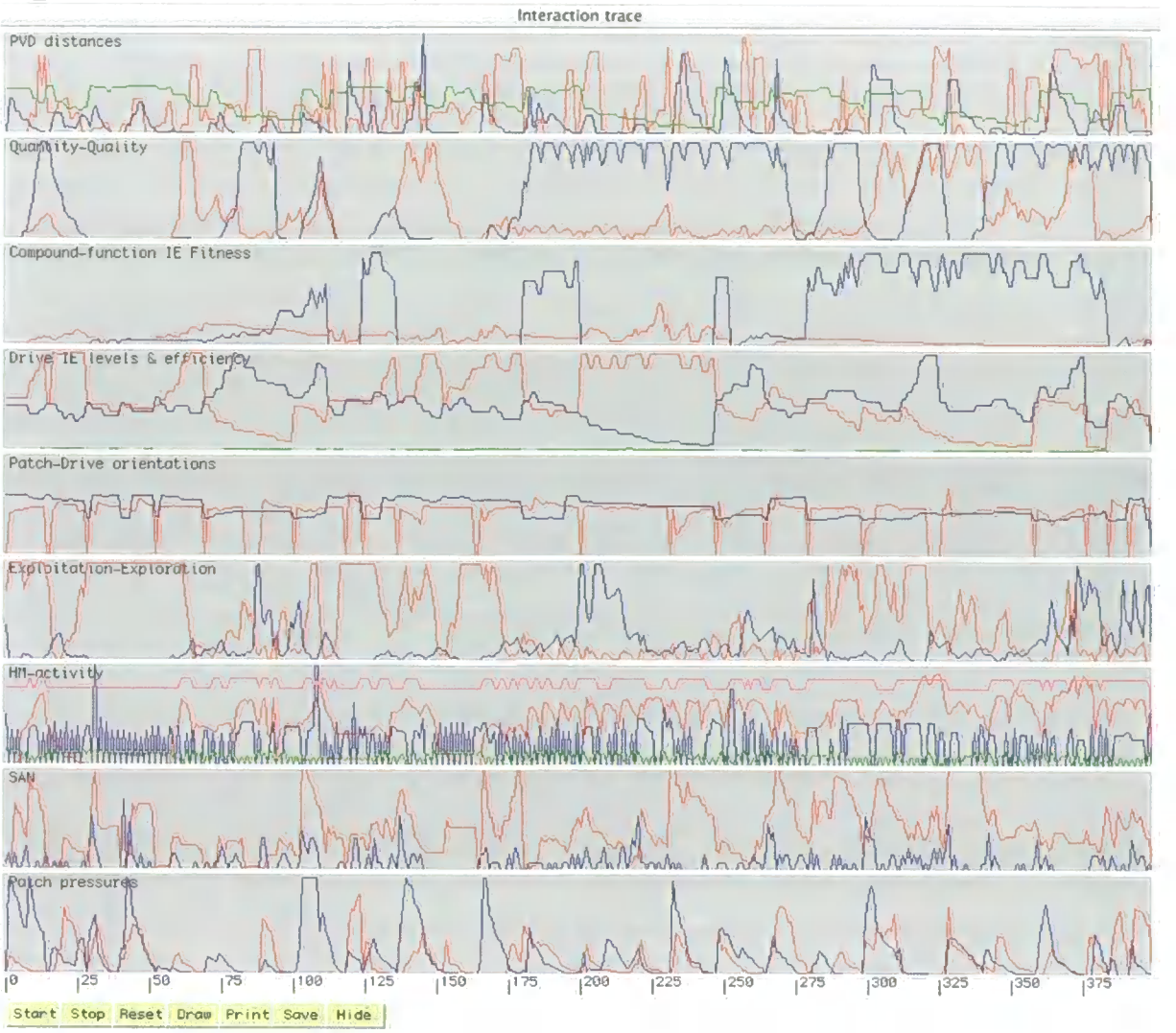


Figure 10.2: A nine-pane tracer window reveals distinctive systems behaviour. This figure documents data acquisition over the first 400 steps of a typical test run of Oscar.

Referring to their label (upper left corner) in figure 10.2, the panes are organised as follows:

1) *PVD distance*: the distance between the last sequence produced by the human interactor and the most recently performed machine-generated melody. This graph thus reflects the similarity between the current contents of the last-sequence and the contents of the most recently played compound-function. The distance is computed using a matrix overlap algorithm (chapter 8, section 8.3), individual dimensions of the MIDI events are considered (and visualised)

separately; pitch (red), velocity (blue) and duration (green). The total distance is computed from a weighted average of all three dimensions (for clarity, not shown in the graph).

2) *Quantity-quality*: this pane provides a picture of how the MIDI input stream changes over time. Quantity (red) mirrors the loudness of the most recent input event and the general density of the input stream. Quality (blue) is computed from the consideration of melodic diversity in the input stream. Both values are subject to multiplicative increments/decrements at every process cycle. Further details are found in chapter 4, section 4.6.8.

3) *Compound-function IE fitness*: this pane shows the global average fitness of all the compound-functions in the current compound function pool. There are two competing fitness values; the first (red) shows total fitness for the intention of integration with the human interactor and the second (blue) value shows the current fitness level for the purpose of expression; i.e., producing musical output in conflict with the human proposed context.

4) *Drive IE levels* and *efficiency*: this pane shows two competing machine motivations; one is pushing towards integration, the other towards expression. Both motivation pressures are modulated around a centre value of 50 percent. The final motivation – basically thought of as a machine *suggestion* to integrate or express – is decided by the highest bidding value. The efficiency of the drives (green) reflects how well their current relationships actually

contributed to the realisation of their intentions. Note the unpredictable, non-linear interactions between both levels. Details are found in chapter 9.

5) *Patch-drive orientations*: this pane documents the levels of the global-orientations (-100 to 100, for visualisation, scaled to a value between 0 and 100) of respectively the patch (red) and the drive (blue). (Note that some detail gets lost because of scaling).

6) The *exploration-exploitation* levels documented in pane 6 show the exploration and exploitation pressures as manipulated by the nature of the input signal to the listening network. In particular, changes in human-responsiveness control the scaling algorithm as explained in chapter 4, section 4.5.8. The actual momentary levels of the exploration and exploitation pressures will guide the selectionist activity in all current object populations given a learning-mode equalling *real-time*.

7) *HM-activity* in pane 7 documents three elements: the level of human activity as reflected in the *human-responsiveness* level (red line), the level of *machine activity* (blue) as suggested by the number of events in the melody performed by the reference agent in the current players agency and, finally, the ID of the *current interaction pattern* (green). Again, some detail gets lost because of the low resolution of the panes in the GUI – yet the global picture still communicates a lot of data. Finally, the fourth data item (pink colour) shows a signal alternating between two values (low and high), they reflect the momentary orientation of the players agency – the option of *responsive*

performance (high) or *autonomous* performance (low). Note that the options are not just binary opposites but just indicative orientations that provide opportunities for weighted performance merging aspects of autonomous and responsive behaviour (chapter 7, section 7.3).

8) The pane labelled *SAN* depicts activity in the listening network. The red line shows the current sum of the *activation-inhibition-vector* (see chapter 5, section 5.5) of the current SAN, the blue line reflects the *number-of-neurons-changed*, it is therefore indicative of the present fitness of the current SAN.

9) The bottom pane shows the competing levels for *integration* and *expression* of the patch. Two values accumulate the sum of respectively all positive levels (objective is integration) and all negative levels (objective is expression) of the output levels of the patch. The winning orientation and its strength follows from the ratio between both resulting values.

The following system variables are effective in the test run documented in figure 10.2: *population-size* = 16, *learning-cycle* = 5, *breeding-cycle* = 50, *performance-cycle* = 500, top-level system inclination is *selfish*, both learning and evolution are in use and the experiment employs the internal test-player. Thus, the experiment includes 10 epochs of 50 cycles each. The total experiment duration is 57 minutes and 50 seconds.

1	Time in milliseconds since start of program
2	Total man-machine similarity
3	Man-machine similarity for the dimension of pitch
4	Man-machine similarity for the dimension of velocity
5	Man-machine similarity for the dimension of duration
6	Global orientation of current patch
7	Global orientation of current drive
8	Quantity of human (or test-player) input
9	Quality of human (or test-player) input
10	Total fitness of compound function pool for the purpose of integration
11	Total fitness of compound function pool for the purpose of expression
12	Integration level of current drive
13	Expression level of current drive
14	Efficiency level of current drive
15	Human agreement level
16	Machine agreement level
17	Global orientation of current patch
18	Global orientation of current drive
19	System exploitation level
20	System exploration level
21	Human-responsiveness
22	Number events in current compound-function of current reference agent
23	Sum of neural-activation-inhibition levels of current sensor-activator-network
24	Number of neurons changed in current sensor-activator-network
25	Integration level of current patch
26	Expression level of current patch
27	Input-pressures-adapted vector of every agent in the players agency
28	List of the ID of the compound-function resident in every agent of the agency

Table 10.1: List of systems variables collected by the tracer-data collector.

Interesting correlations are observed between the various parallel data streams. In addition, the complexity of the data in figure 10.2 echoes the capacity of the test-player algorithm (using simple weighted randomness in all four MIDI dimensions) to produce considerable musical consistency while still managing to influence the

listening network in significant ways. For instance, the second pane labelled *Quantity-Quality*, shows a wide range of orientations – large scale oscillations as well as a forced plateau of nearly 100 % *quality* of the input MIDI stream erecting at sample 175. The third pane, *Compound-function IE fitness*, shows how different areas of populations of musical processing functions develop that excel for the purpose of expression (blue line). Pane 6 proves that exploration and exploitation pressures also interact in intricate ways. In addition, a complex relationship is evident between human input activity (pane 7) and the data in pane 6.

Our current implementation offers immediate data inspection as shown in figure 10.2. However, another similar tracer algorithm tracks *additional* systems variables into a temporary data structure and intermittingly writes its contents to disk. In effect, the tracer-data collector tracks 28 variables as listed in table 10.1.

The *save-improviser-tracer-data* function appends the new data block to the current file; the function takes effect whenever a significant interaction pattern occurs as clarified in chapter 9.

10.2.2 The Evolution-Data Collector

As explained above, two types of data are gathered during an experiment; the *tracer-data* and the *evolution-data*, both reflect significant information at different moments during an interactive session either using a human interactor or the internal test-player. The *save-improviser-evolution-data* function – reporting data gathered by the evolution-data collector – fires less often than the *save-improviser-tracer-data* function – it is only executed at the end of every learning cycle – thus also just before breeding the next population; i.e., when the last learning cycle finishes within a given breeding cycle. As such, the evolution-data reflects information that happens to be

accumulated during the course of a single epoch – as a consequence, it informs on the impact of learning and reports on the genetic activity in the system. The following items are saved to disk:

1	Time in milliseconds since start of program
2	Interaction-counter
3	Performance-counter
4	Number of runs of every SAN in current SAN-pool
5	Fitness of every SAN in current SAN-pool
6	Number of runs of every patch in current patcher
7	Fitness of every patch in current patcher
8	Number of runs of every drive in current drives-pool
9	Efficiency of every drive in current drives-pool
10	Integration-level of every drive in current drives-pool
12	Expression-level of every drive in current drives-pool
13	Understanding-level of every drive in current drives-pool
14	Number of runs of every CF in current compound-function-pool
15	Integration-fitness of every CF in current compound-function-pool
16	Expression-fitness of every CF in current compound-function-pool
17	Global-orientation of current patch
18	Global-orientation of current drive
19	System global-orientation
20	System common understanding
21	Current interaction-trail (16 element vector)
22	List of the ID of the compound-function resident in every agent of the agency

Table 10.2: List of systems variables collected by the evolution-data collector.

All experiments reported here address the data brought together by the tracer-data and evolution-data collectors. In order to grasp the state space of the system, we ran a number of pilot experiments. Then, a series of systematic experiments was developed with the objective of analysing the behaviour of Oscar in more detail with respect to the various parameter settings, in particular, studying the effect of system inclination

(social or selfish) and leaning-mode (explicit or real-time). The next section describes the results of the initial experiments.

10.3 Initial experiments

10.3.1 Experiments e1 to e4: A Comparative Study of System Consistency

N	Steps learn	Steps breed	Steps perform	System inclination	Learn	Input	Learning-mode
16	2	20	400	Social	Yes	Test-player	Explicit

Table 10.3: Parameter settings for experiments e1 to e4.

The goal of this series of experiments was to assess the consistency of the behaviour of the system – in particular the compound-function pool. Given the same parameter settings for all components and given the same input stimuli, we would expect a certain degree of similarity in the behaviour of the networked components. Four simulations are set up in order to compare the respective interaction patterns over time. The simulation parameters are given in the table above.

Since the total simulation takes 400 steps and one epoch takes 20 steps, the number of epochs is also 20. The number of significant temporal interaction patterns captured by the tracer and total duration, respectively for the four experiments was as follows:

Experiment	**e1**	**e2**	**e3**	**e4**
Nr patterns	957	985	962	1011
Duration	43 m 2 s	40 m 48 s	39 m 31 s	41 m 14 s

Table 10.4: Number of patterns and durations of experiments e1 to e4.

The test-player generates short input sequences (between three and eight MIDI events) by randomly selecting values – for pitch, velocity, duration and inter-onset-times – from user specified lists. The list values together form a small stylistic database. This experiment addresses the compound-function pool, the patcher and the drives. Only the tracer-data is addressed in the analysis described next.

1) Fitness of the compound-function pool.

Figures 10.3 to 10.6 reveal quite interesting behaviour. Fitness levels fluctuate in episodes; more or less regularly spaced at the beginning and more irregular towards the end. Levels develop momentum inside single episodes and an overall incremental fitness for both orientations in the long run. The compound-function pool hence manages to optimise all functions collectively; fitness levels maintain relative stability between epochs. Generally speaking, the expression level supersedes the integration level. Logically, we may conclude that it requires more evolved expertise to integrate with an unpredictable input source and less expertise to express a personal character, that is, sound very different than that input source.

Figure 10.3 develops an incremental expression fitness starting around sample 480 and lasting till sample 880 while integration remains exceptionally steady. Nearly 50% of the evolutionary process (about ten epochs of 20 in total) behaves in a remarkably linear fashion. This is evidence that the CF-pool is able to adapt gracefully facing irregular input. The integration fitness level suddenly boosts at sample 840 and sample 880 signals complete breakdown of expression and integration fitness. Evolution is thus a process of gradual modification *and* abrupt changes as reflected in this experiment.

Figure 10.4 shows more dramatic effects of the mutation operator. Expression fitness suddenly peaks to 90 % at sample 760 and then manages to keep sustained at just over 20 %. Integration level increases at sample 800 and stabilises around 30 %. Similar behaviour is noticed in figure 10.4 and to a lesser extends in figure 10.5. It is concluded that genetic programming applied to the musical processing functions inside the CF-pool is successful. After many generations of irregular lower fitness values, both values increase and settle into a more or less stable pattern. The CF-pool adapts into a dynamic equilibrium.

2) Patcher global orientation level.

Every sample in figures 10.7 and 10.8 shows the momentary total average of all output values of all 16 patches in the current patcher. The values fluctuate between approximately +20 and -30. The global result is either positive or negative; the current relationships interpret the current sensor output via the current relationships. Positive output is interpreted as an attitude of the input source (human or test-player) to *integrate*, a negative output denotes a wish for *expression*. Notice that the negative values are rather more apparent, this implies that expression slightly dominates over integration. This observation agrees with the conclusions made above. Notice that the polynomials (black curves) in both figures are a sign of slow wave-like behaviour; the peaking patcher output oscillates at a very low frequency. It is natural to expect emergent dynamic stability in a living system through the presence of many simultaneous oscillating components. The patcher, as part of the global listening network, is subject to quite erratic peaking activity, yet its constructive effect results in a gentle

behavioural drift that is also clearly observed. The last 200 samples in figure 10.8 tend to become positive signalling an interpretation of the input source to provide more pressure to integrate rather than to express. This observation correlates exactly with the evolution of the data in the compound-function pool above.

3) Drives global orientation level.

Figures 10.9 and 10.10 show further evidence of the existence of underlying wave-like behavioural patterns. The global average output values of all current drives in the drives pool are shown. The patterns oscillate between -10 and +10. The signal reveals areas of relative continuity in between sharply peaking swings. The polynomial in figure 10.9 documents a mainly negative orientation at first; this implies an initial tendency for the machine motivation to prefer *expression*. After sample 800, the orientation becomes *integration*. Again this correlates with the findings in figures 10.6 and 10.8. In conclusion, for both the patcher and the drives-pool negative values dominate indicating structures that are fitter for the purpose of expression than integration.

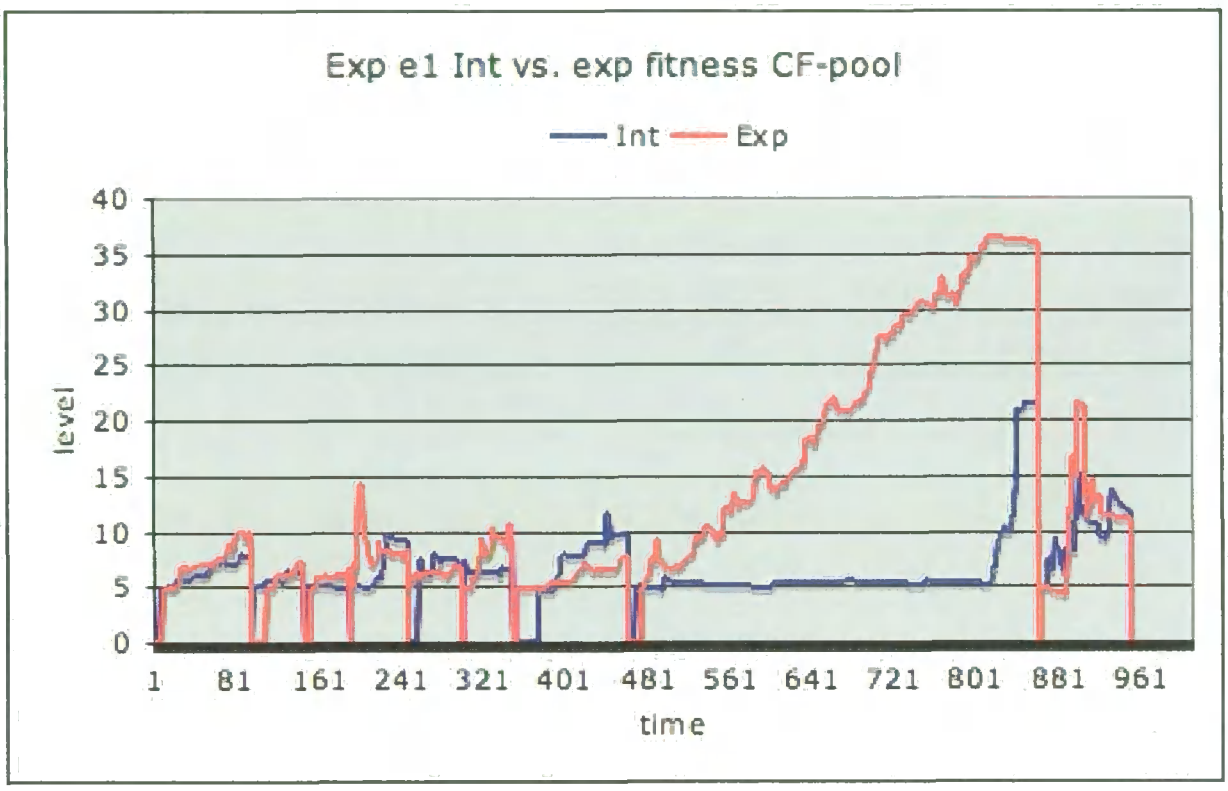


Figure 10.3: Compound-function pool integration vs. expression fitness levels in e1.

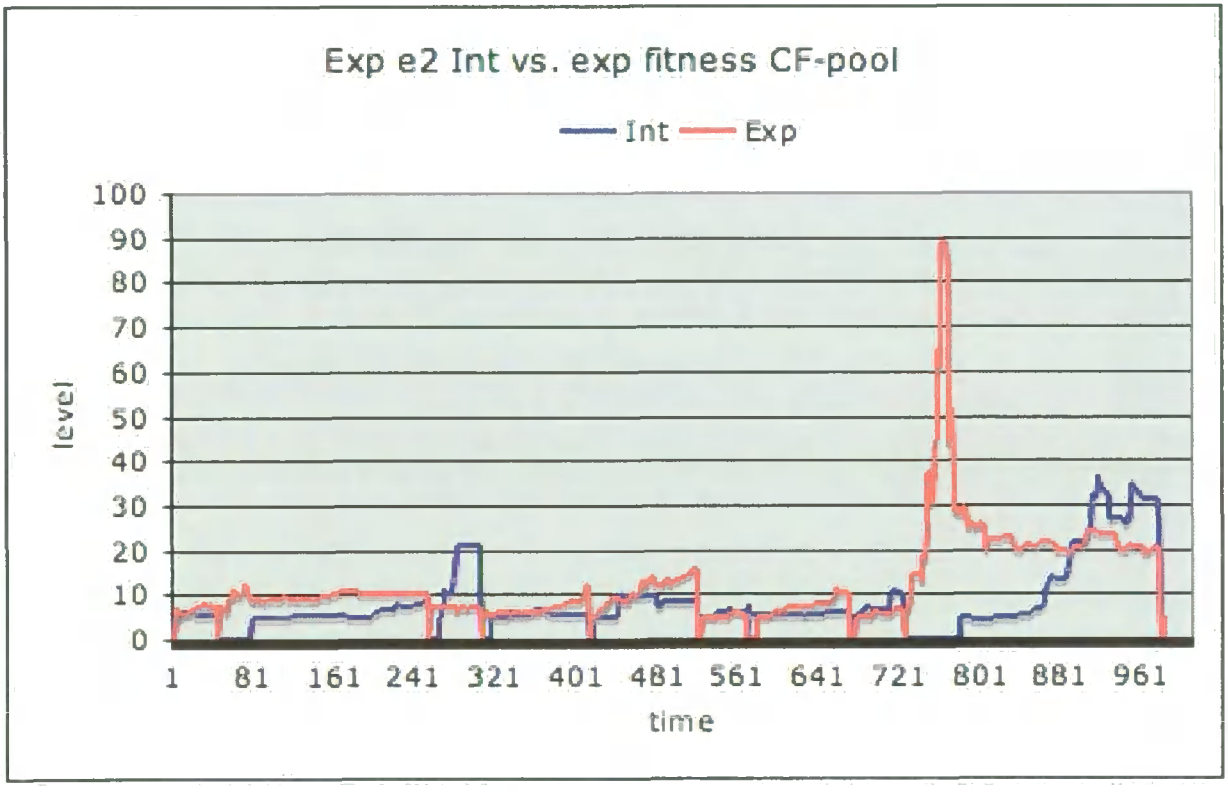


Figure 10.4: Compound-function pool integration vs. expression fitness levels in e2.

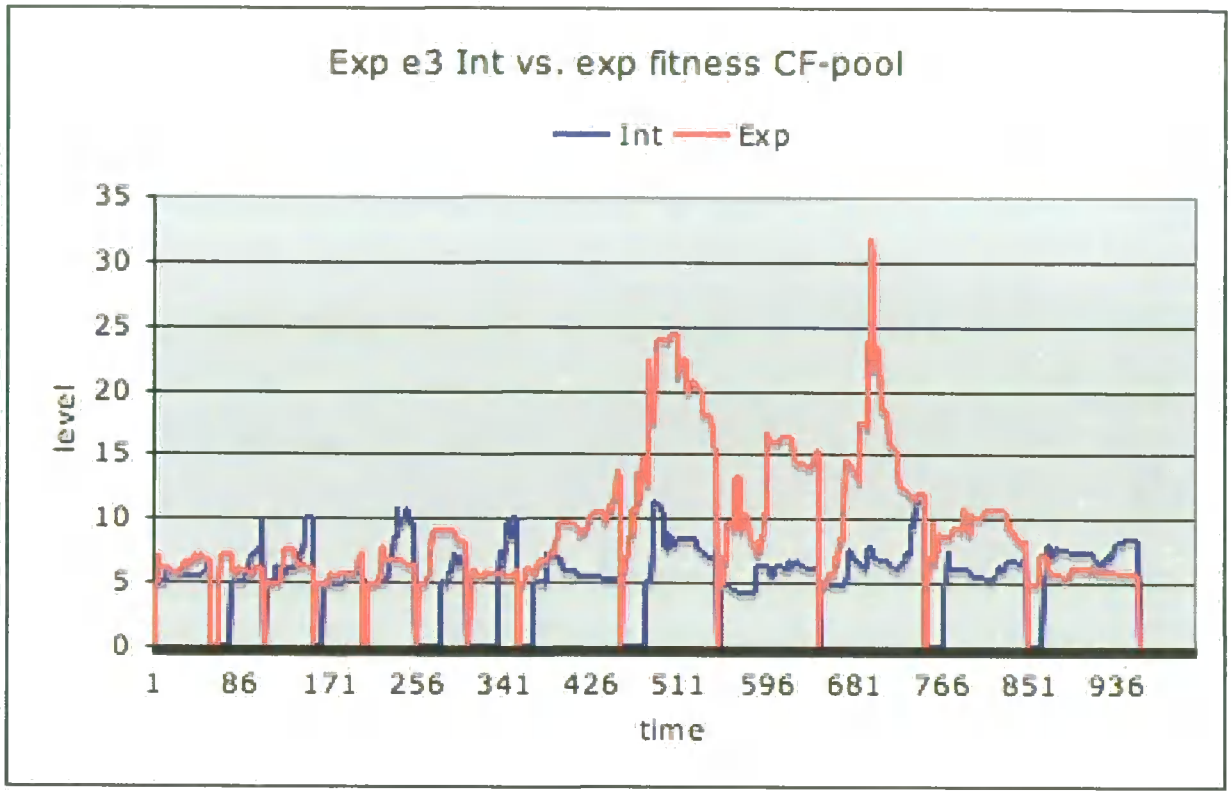


Figure 10.5: Compound-function pool integration vs. expression fitness levels in e3.

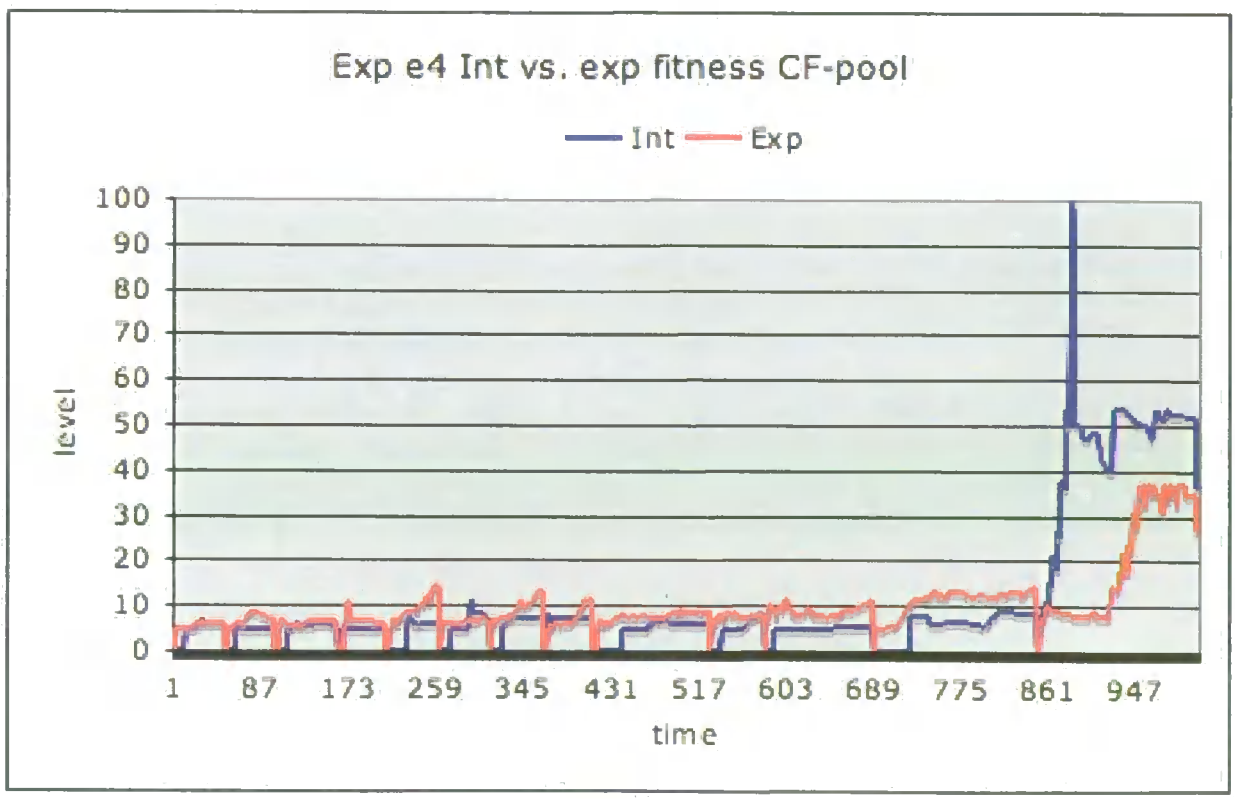


Figure 10.6: Compound-function pool integration vs. expression fitness levels in e4.

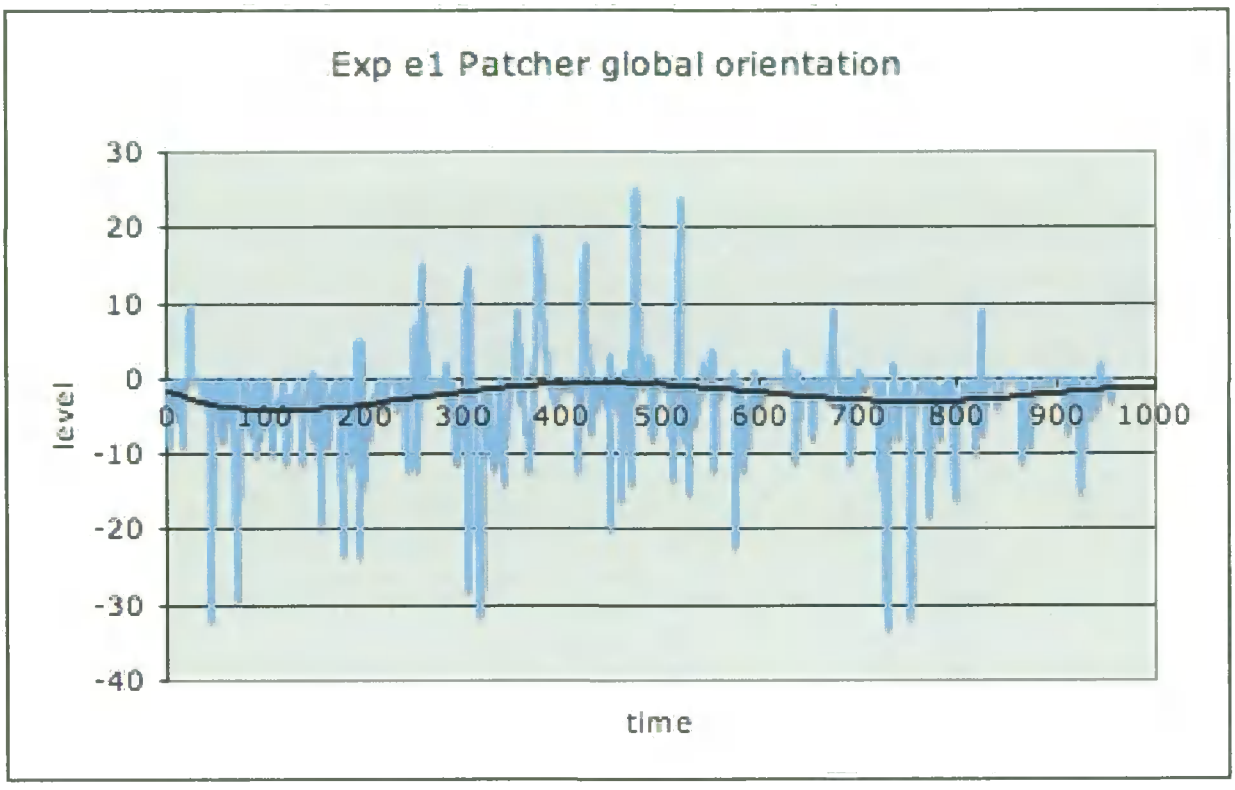


Figure 10.7: Patcher global orientation levels in e1.

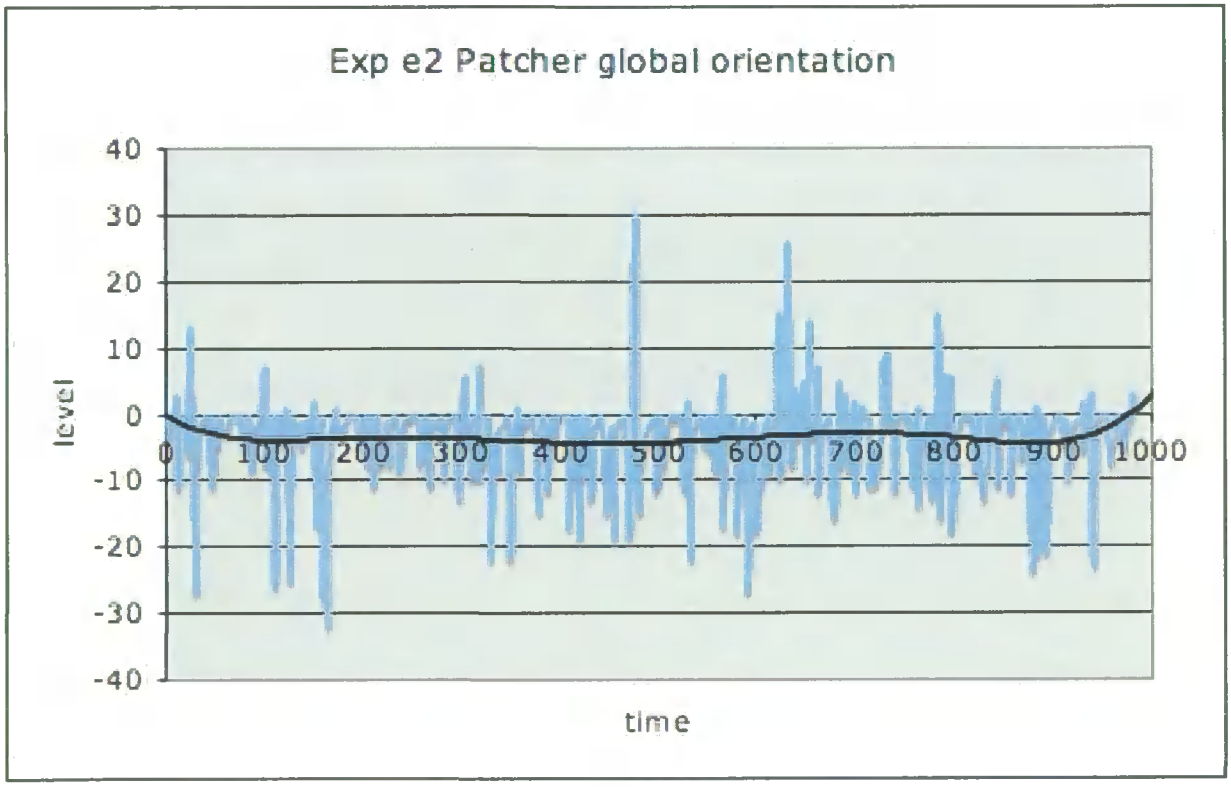


Figure 10.8: Patcher global orientation levels in e2.

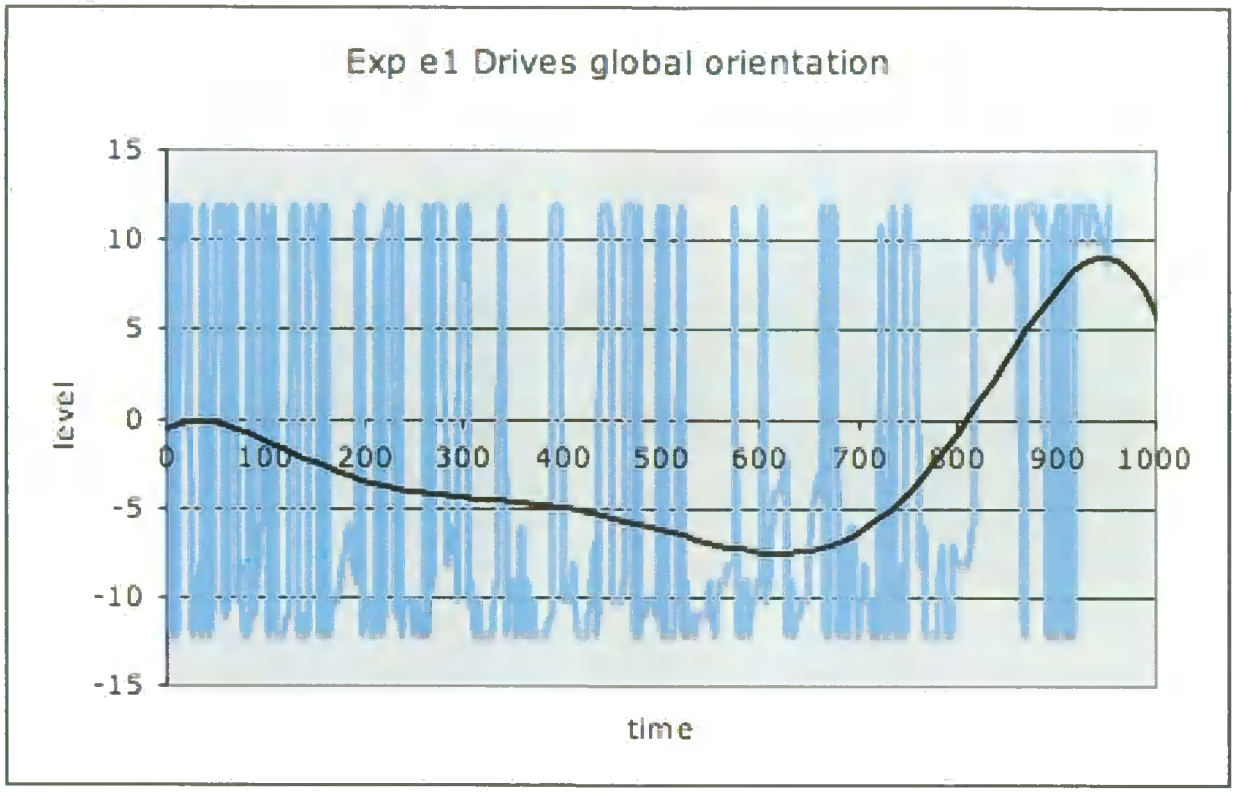


Figure 10.9: Drives global orientation levels

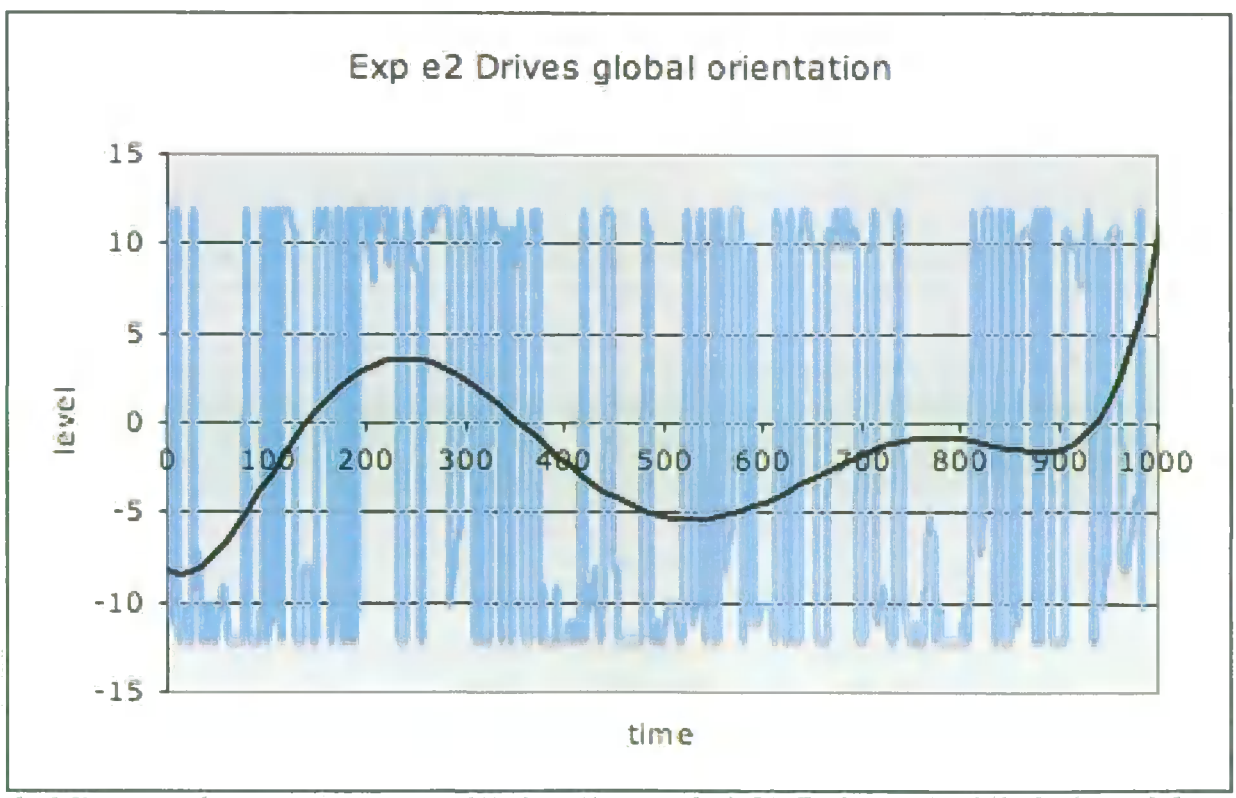


Figure 10.10: Drives global orientation levels

10.3.2 Experiments e5 and e6: a comparative study of system inclination

	N	Steps Learn	Steps Breed	Steps Perform	System Inclination	Learn	Input	Learning-mode
e5	16	1	20	200	Social	Yes	Human	Explicit
e6	16	1	20	200	Selfish	Yes	Human	Explicit

Table 10.5: Parameter settings for experiments e5 and e6.

The objective here is to compare the impact of the global machine orientation parameter: the system *inclination* set either to *social* or *selfish*. In case of social inclination, the system will compare the outputs of the current patch and current drive. If both values carry the same sign, man and machine are considered in agreement else they are viewed as in conflict. The amplitude of the agreement may then condition the increment in fitness of the current compound-function.

In case of selfish inclination, the output of the patch is not considered. The comparator takes the output of the drive (the current machine motivation) as a reference to guide the fitness of the compound-function that was executed last; i.e., the one that contributed most recently to obtain the current situation.

Experiment	**e5**	**e6**
Tracer patterns	508	523
Evolution patterns	105	105
Duration	26 m 7 s	25 m 13 s

Table 10.6: Number of tracer and evolution patterns and durations of experiments e5 and e6.

Both experiments employ input from a human interactor, use a single learning step, feature ten epochs of 20 cycles and combine evolution and learning. One examines whether just ten epochs generates enough evolutionary momentum to acquire interesting interaction patterns. The tracer is addressed with an analysis of the data provided by the currently performing objects – not the average data of all 16 objects in a given population.

Figures 10.11 and 10.12 document the orientation of the currently selected patch. The polynomial shows that the orientation levels remains slightly negative throughout the experiment. However, the signals are widely spiking (full range -100 to +100 in experiment e5) though some oscillations stay positive or negative for almost 50 process cycles.

Figure 10.13, 10.14, 10.15 and 10.16 illustrate the integration and expression levels of the current drive. The polynomials show that the underlying oscillations of integration and expression levels are slightly out of phase. Both levels seem to "follow" each other's activity with a given delay; for example, the activity in figure 10.13 just before cycle 400 and the activity in figure 10.15 just after cycle 400.

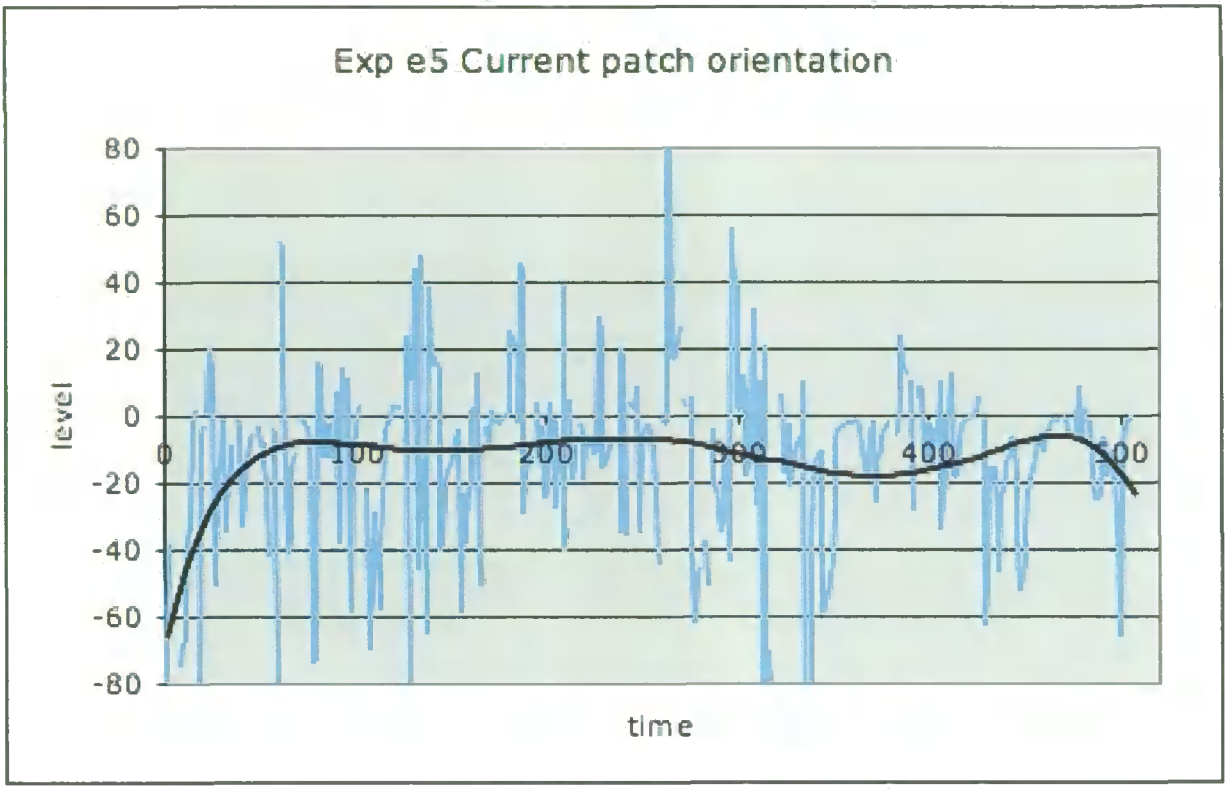


Figure 10.11: Current patch orientation levels in e5.

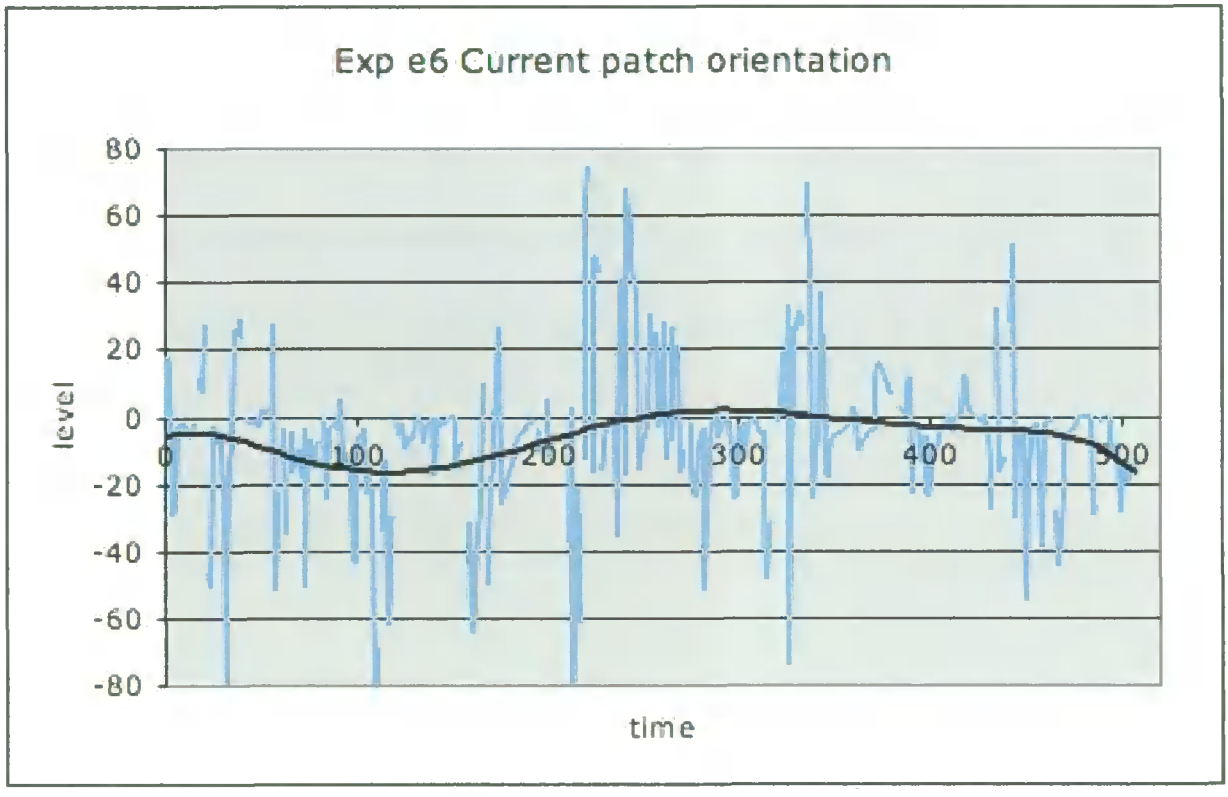


Figure 10.12: Current patch orientation levels in e6.

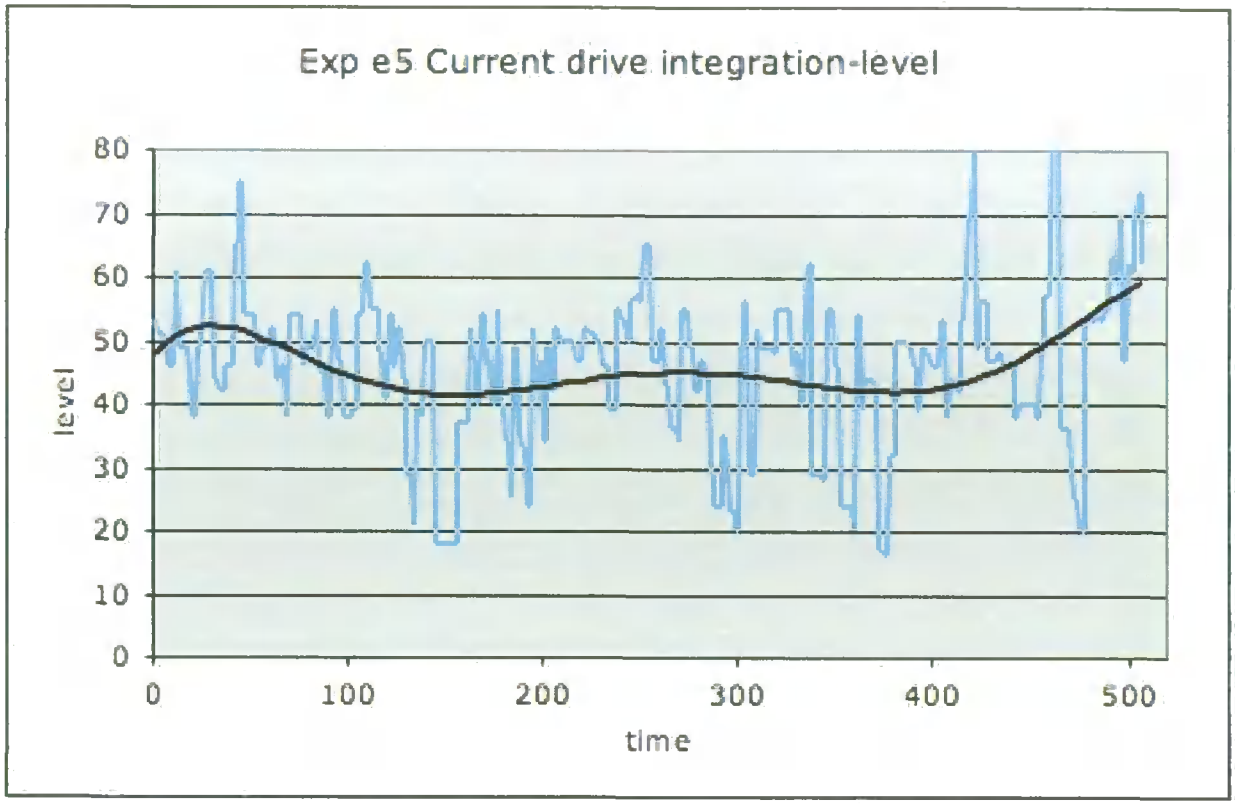


Figure 10.13: Current drive integration level in e5.

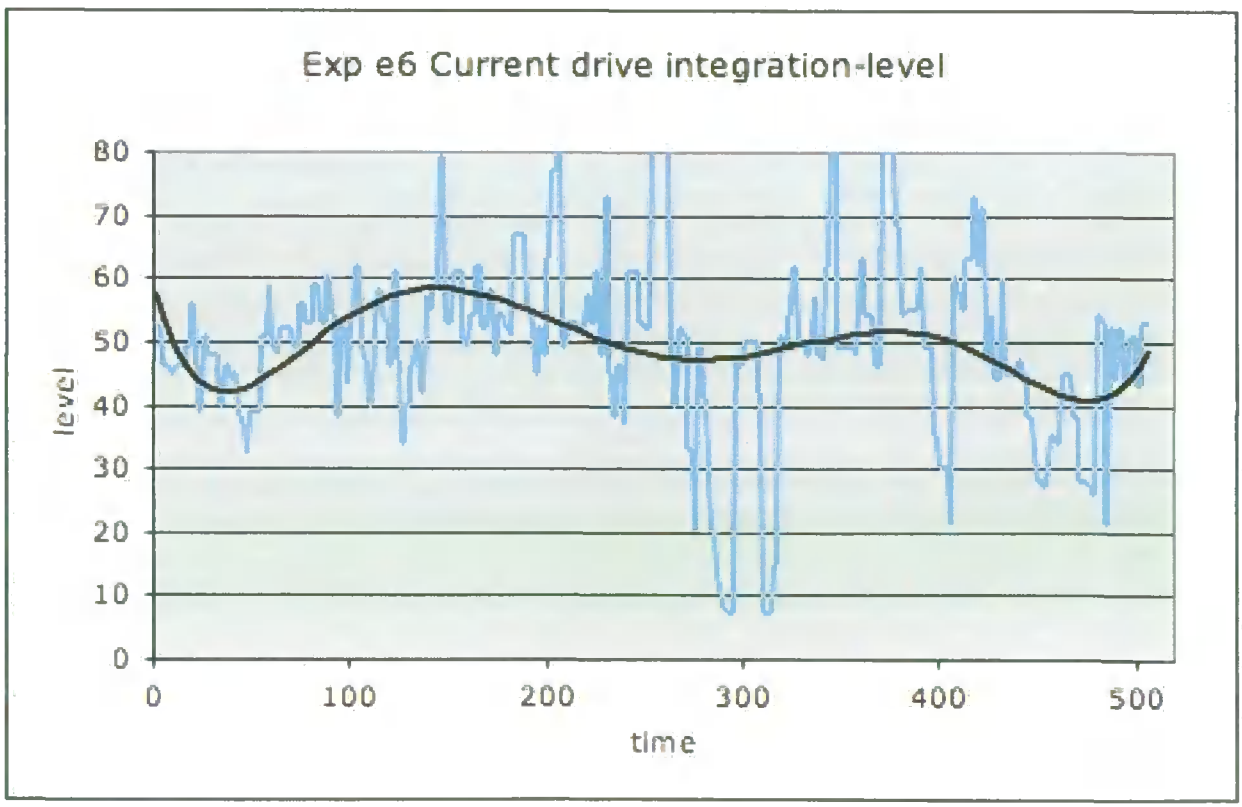


Figure 10.14: Current drive integration level in e6.

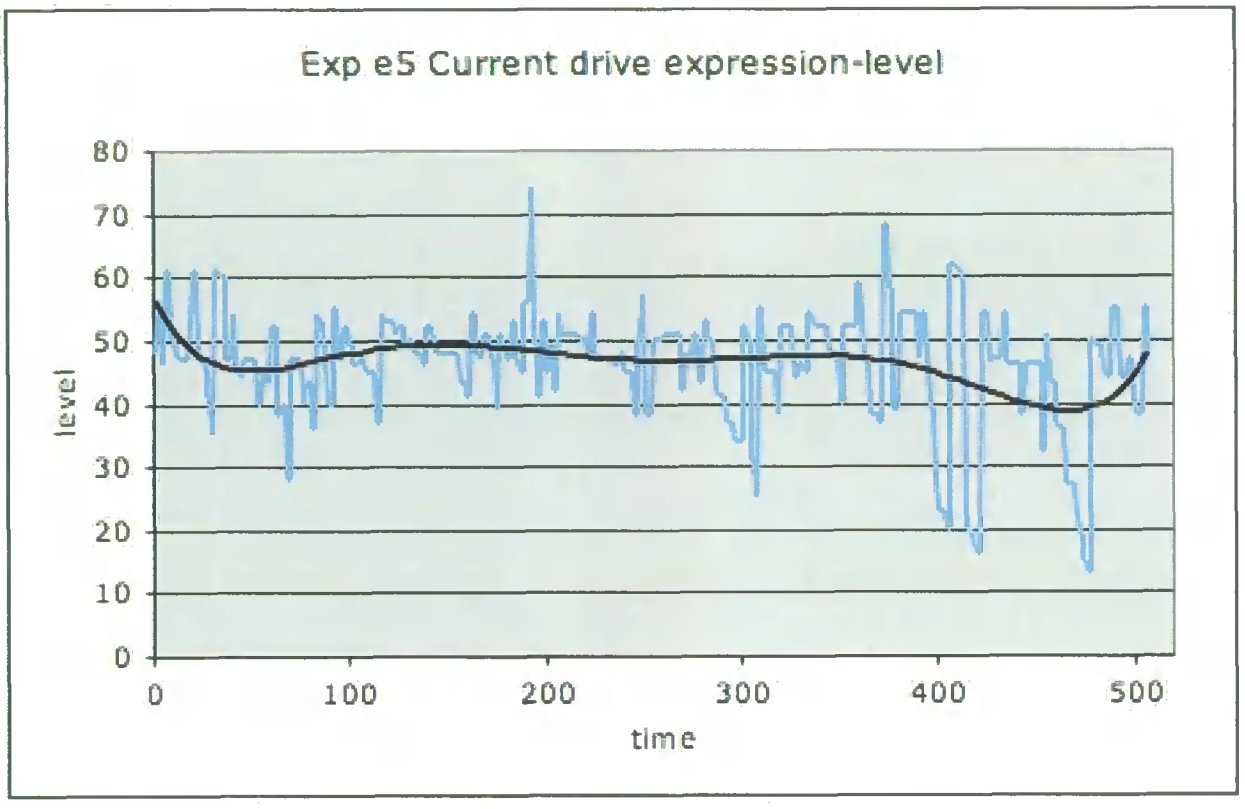


Figure 10.15: Current drive expression level in e5.

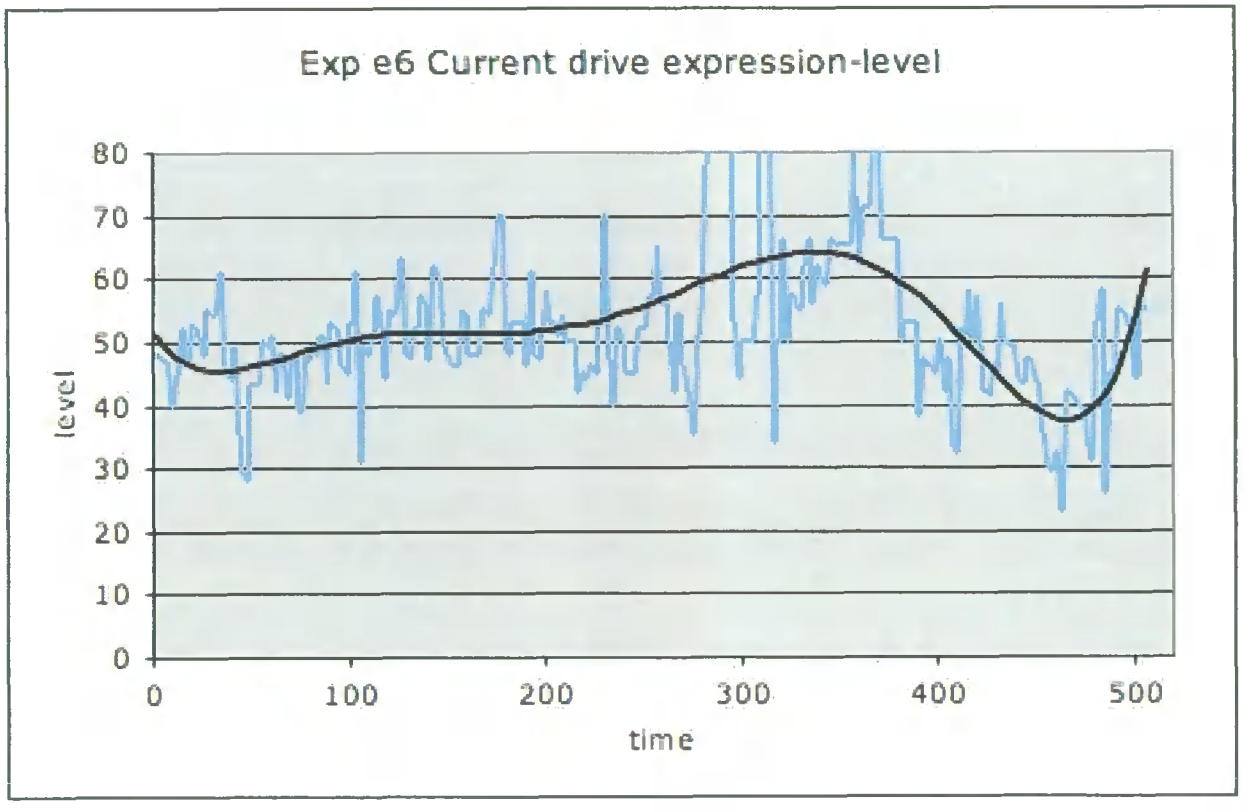


Figure 10.16: Current drive expression level in e6.

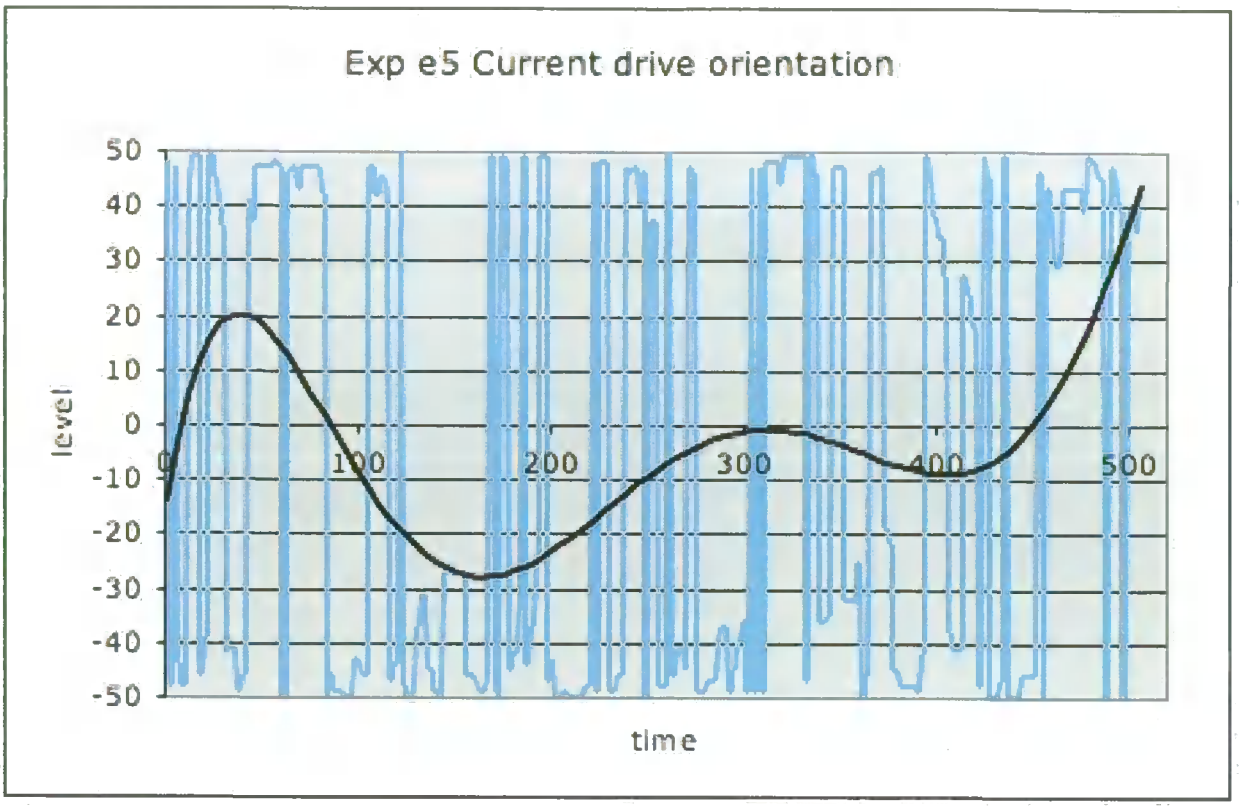


Figure 10.17: Current drive orientation level in e5.

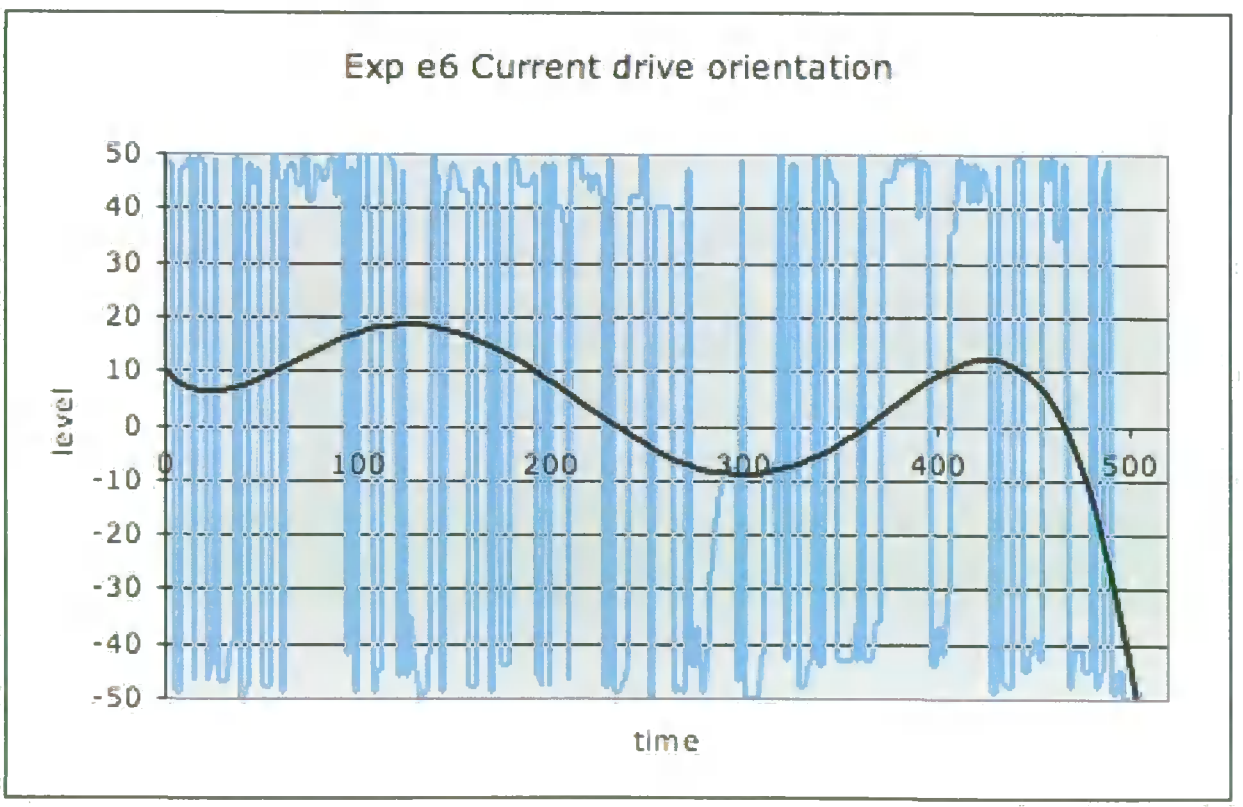


Figure 10.18: Current drive orientation level in e6.

Figures 10.17 and 10.18 show the history of the drive orientations, a value between -50 and 50. The drives alternate between strong negative values (expression) and strong positive (integration) with short areas of relative stability. The polynomials in figures 10.17 and 10.18 are about 180 degrees out of phase relative to each other. This illustrates the competitive nature of the orientation process. In addition, the tendency in figure 10.17 is to increment, in figure 10.18 the opposite is noticed. In experiment e5, the global system inclination is *social* as the incremental nature of the drive seems to confirm; the drive in e5 moves in the direction of more integration indeed. Likewise, the drive orientation level gradually moving to -50 (maximum expression) confirms the *selfish* inclination in experiment e6.

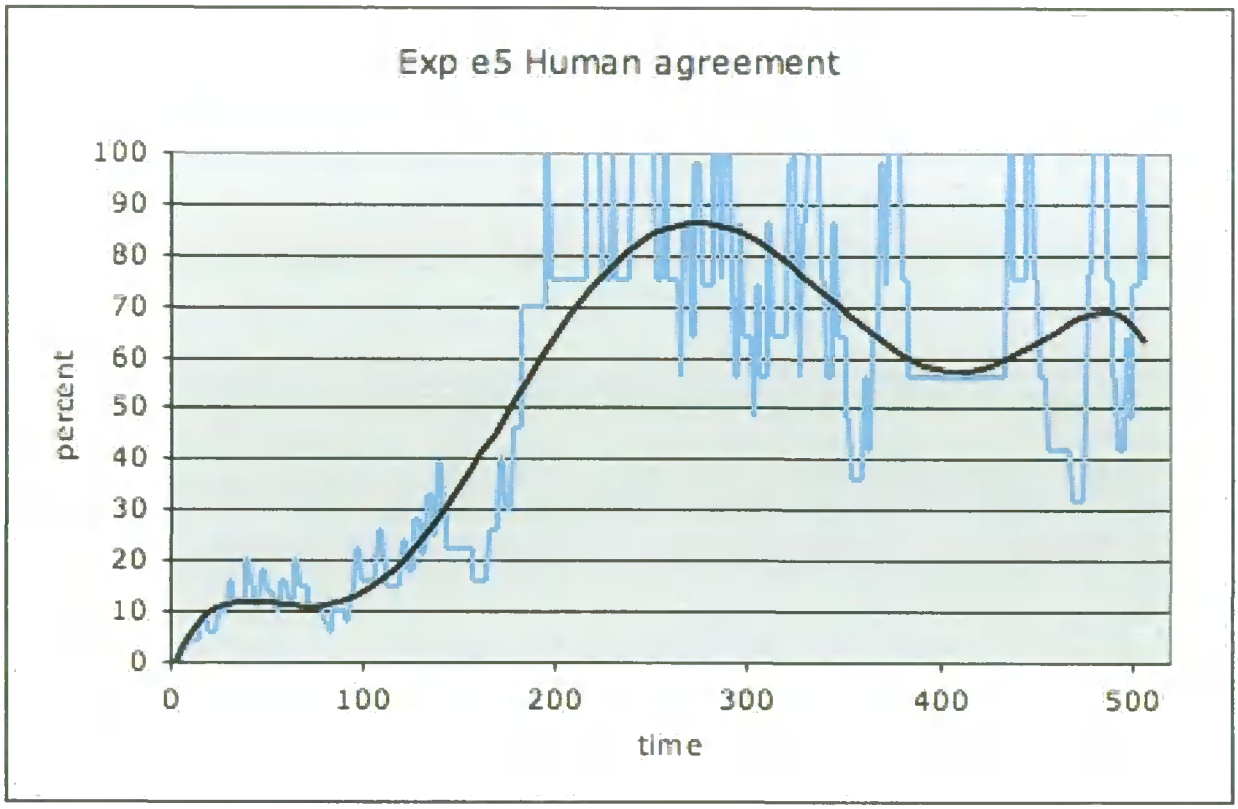


Figure 10.19: Level of human agreement in e5.

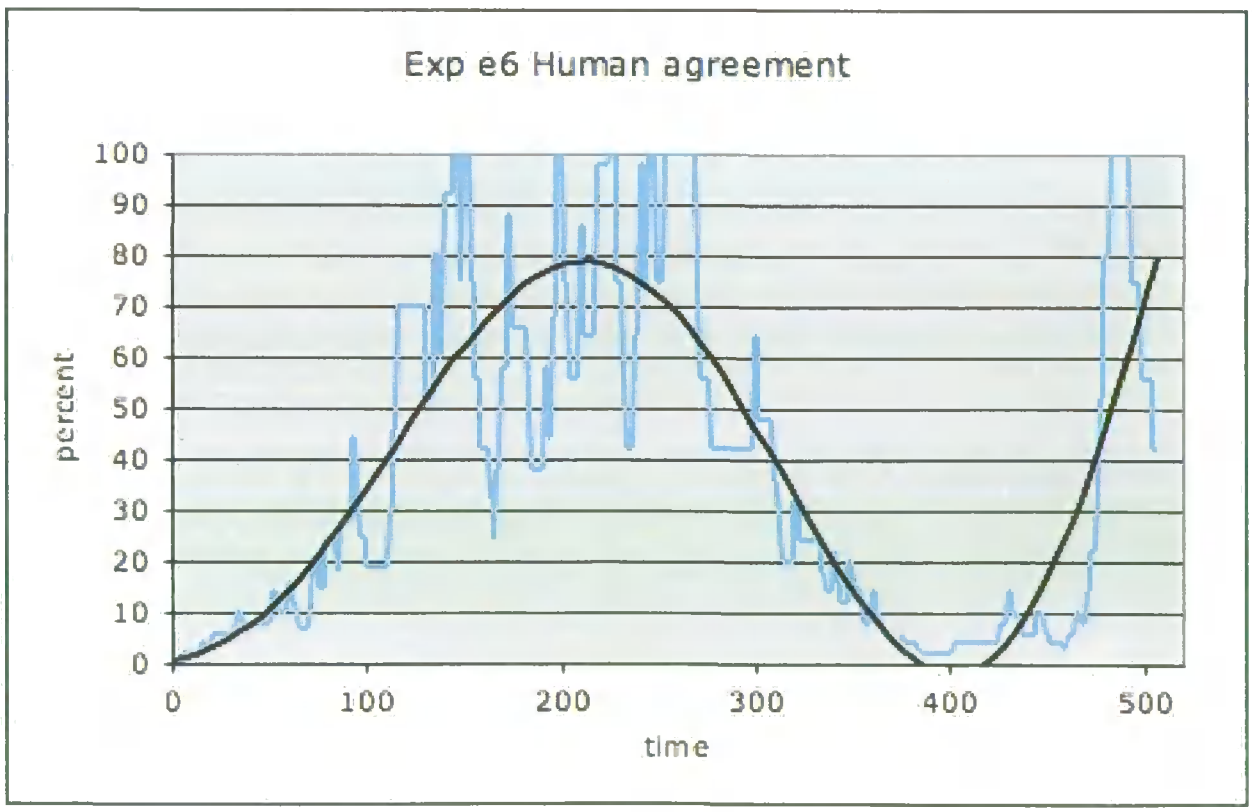


Figure 10.20: Level of human agreement in e6.

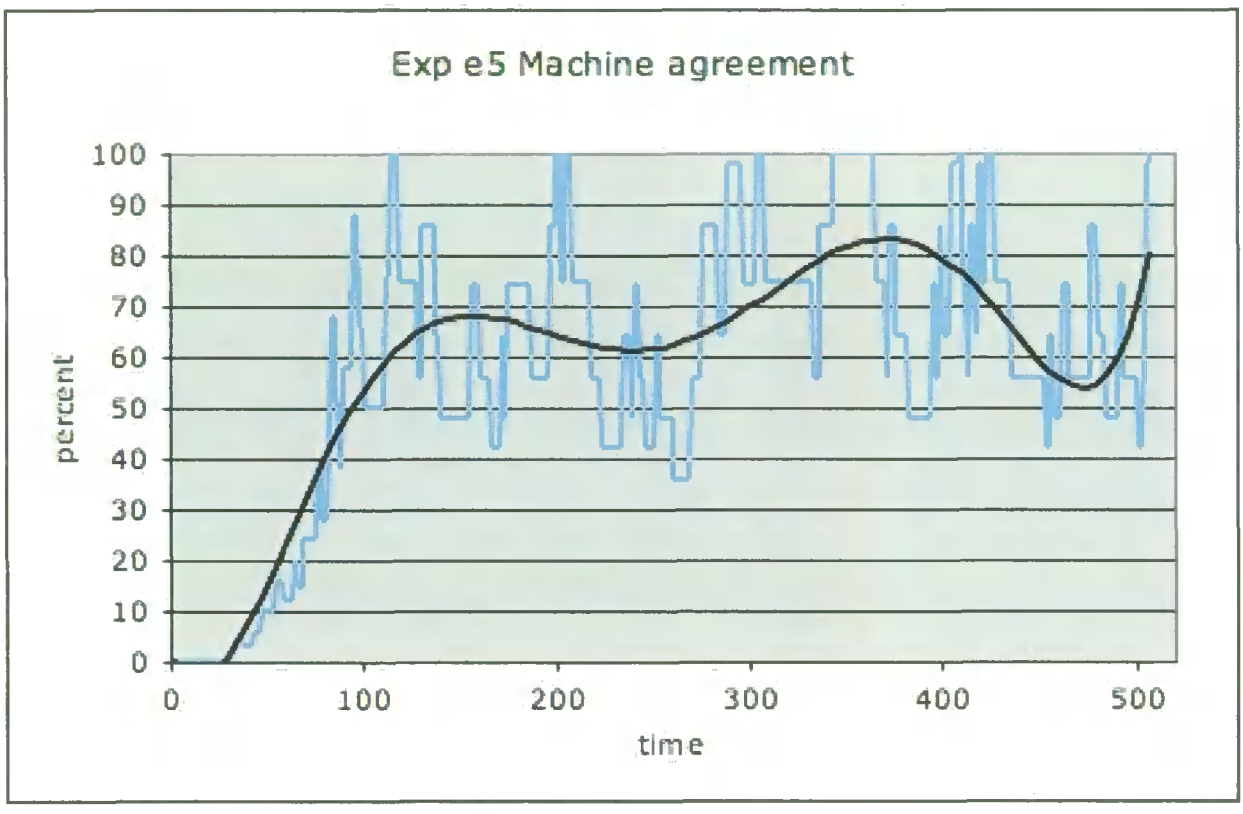


Figure 10.21: Level of machine agreement in e5.

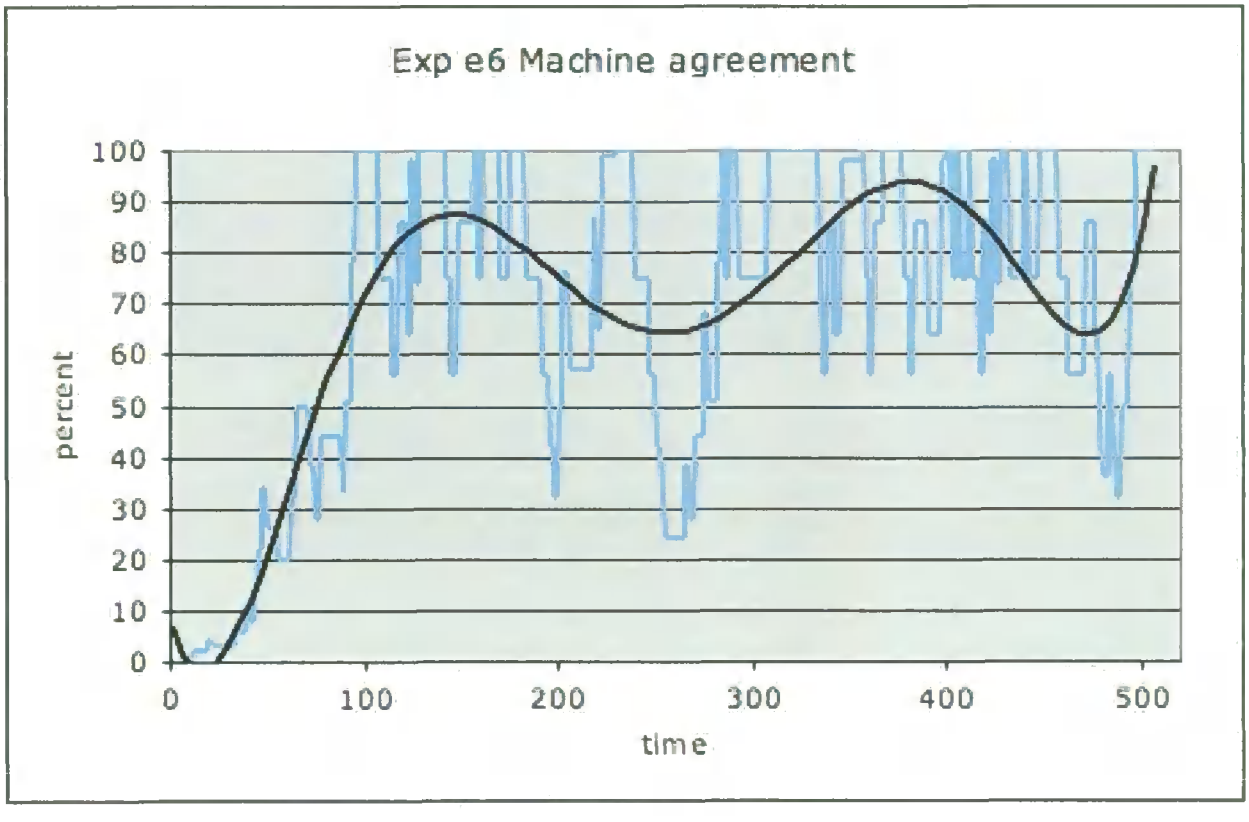


Figure 10.22: Level of machine agreement

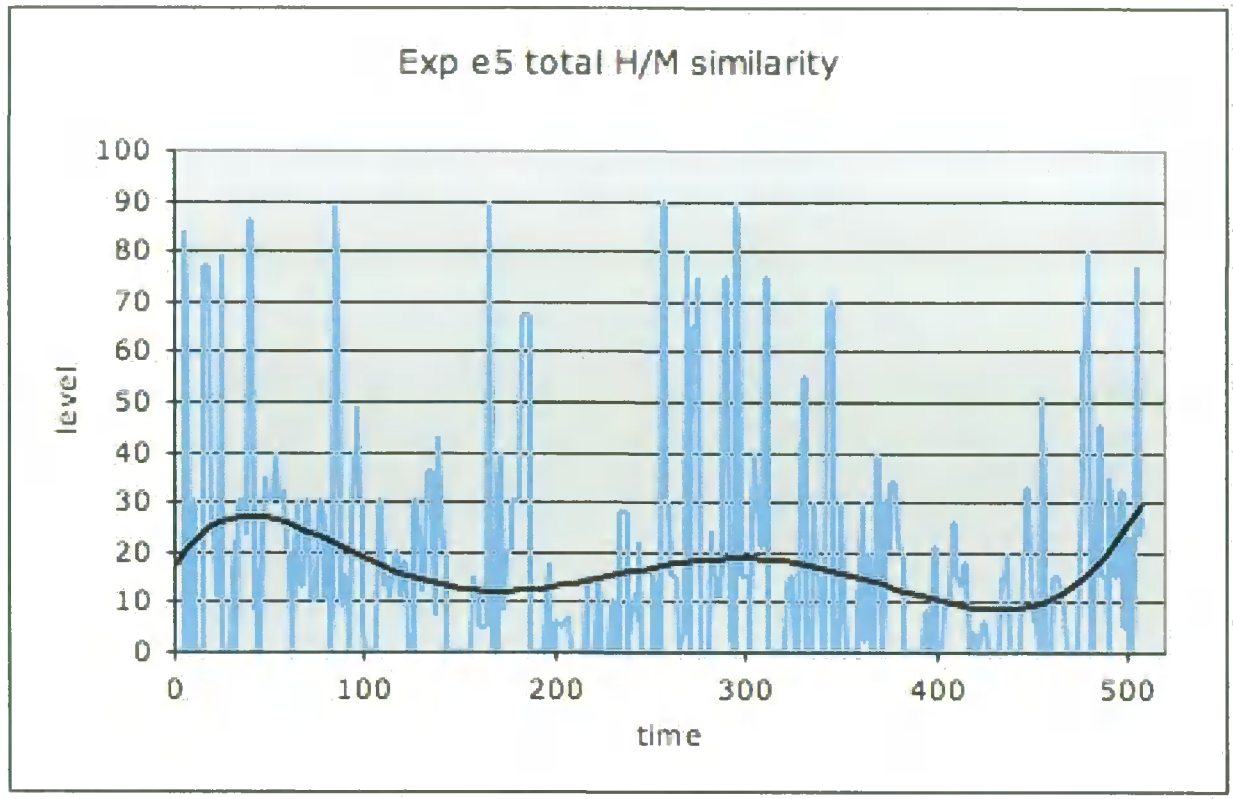


Figure 10.23: Total human/machine similarity in e5.

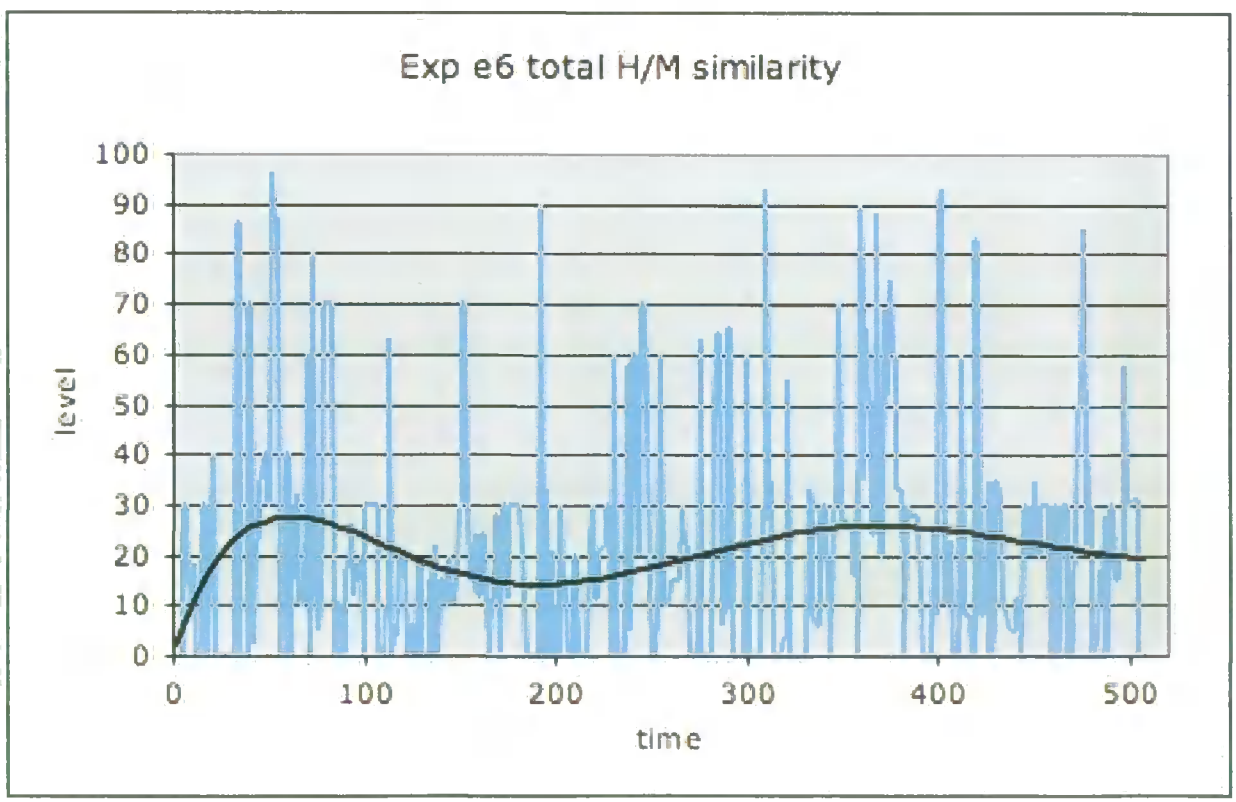


Figure 10.24: Total human/machine similarity in e6.

Figures 10.19 to 10.22 demonstrate the relationship between the levels of human and machine agreement. Both levels are computed by considering the similarity of the most recent sequence produced by the human interactor and the very last machine produced sequence; i.e., the contents of the melody constructed by the reference agent in the agents society.

The agreement levels are either scaled up or down according to the change in similarity; for instance, machine-agreement is scaled up (factor 1.5) when current similarity is higher than the previous similarity, when smaller it is scaled down (factor 0.75). Thus agreement is directly coupled to a change in melodic similarity at a given process step (see also chapter 9, section 4.2) . All agreement levels build up at the beginning and enter an oscillatory pattern. Both agreement levels are computed at two different points in time: at the clock tick when the respective players *just* finished playing. Still, the agreement levels in figure 10.19 and 10.21 are complementary signals out of phase by nearly 180 degrees. This emergent association follows from the relationships that are active inside the current drive. Remember that changes in similarity are interpreted by these relationships and so lead to a machine motivation to get either closer to the human performer (integrate) or move away (expression). The different drives in the drives pool offer many graded options; the statistical effect is that man and machine happen to function at a fluctuating average musical distance.

The agreement algorithm is quite sensitive and adapts very quickly to changes in the input style of the human performer as seen in figure 10.20.

The total similarity between man and machine is shown in figures 10.23 and 10.24. Similarity is computed using the matrix comparator method (see chapter 8, section 8.3).

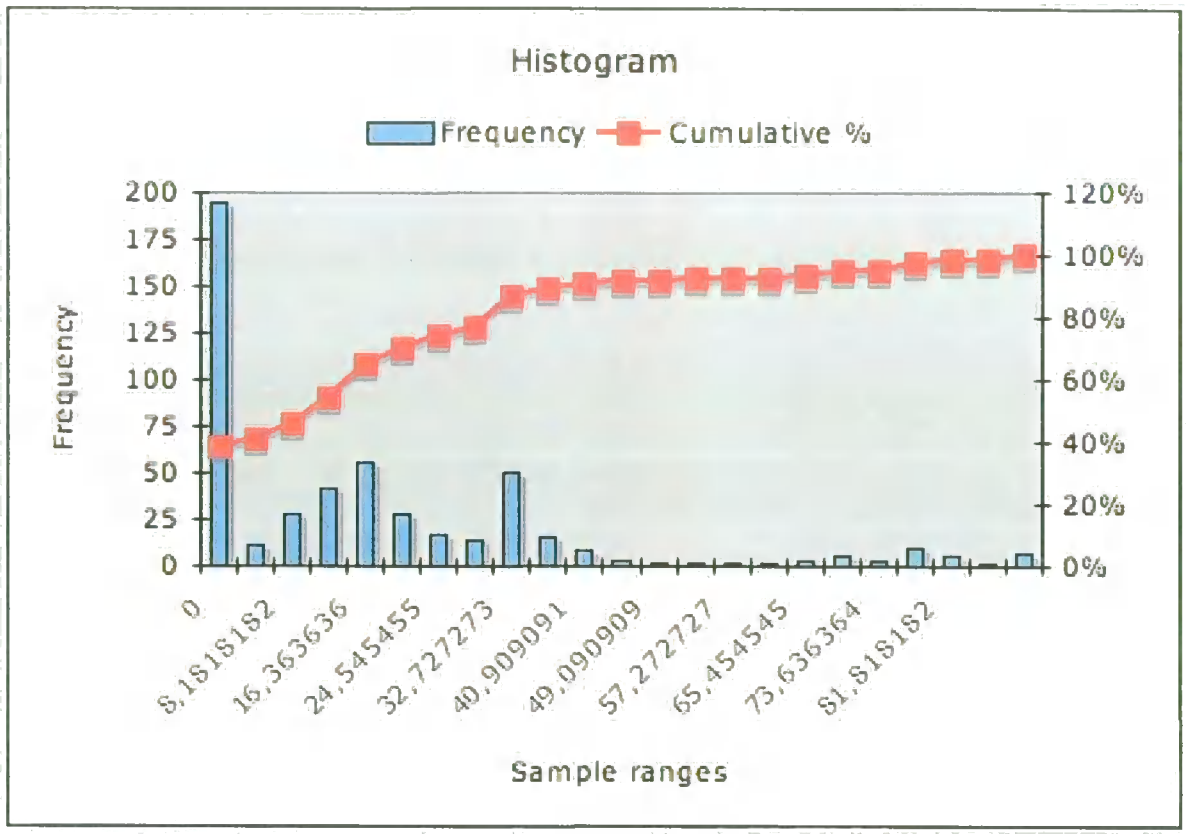


Figure 10.25: Histogram of levels of human-machine similarity in e5.

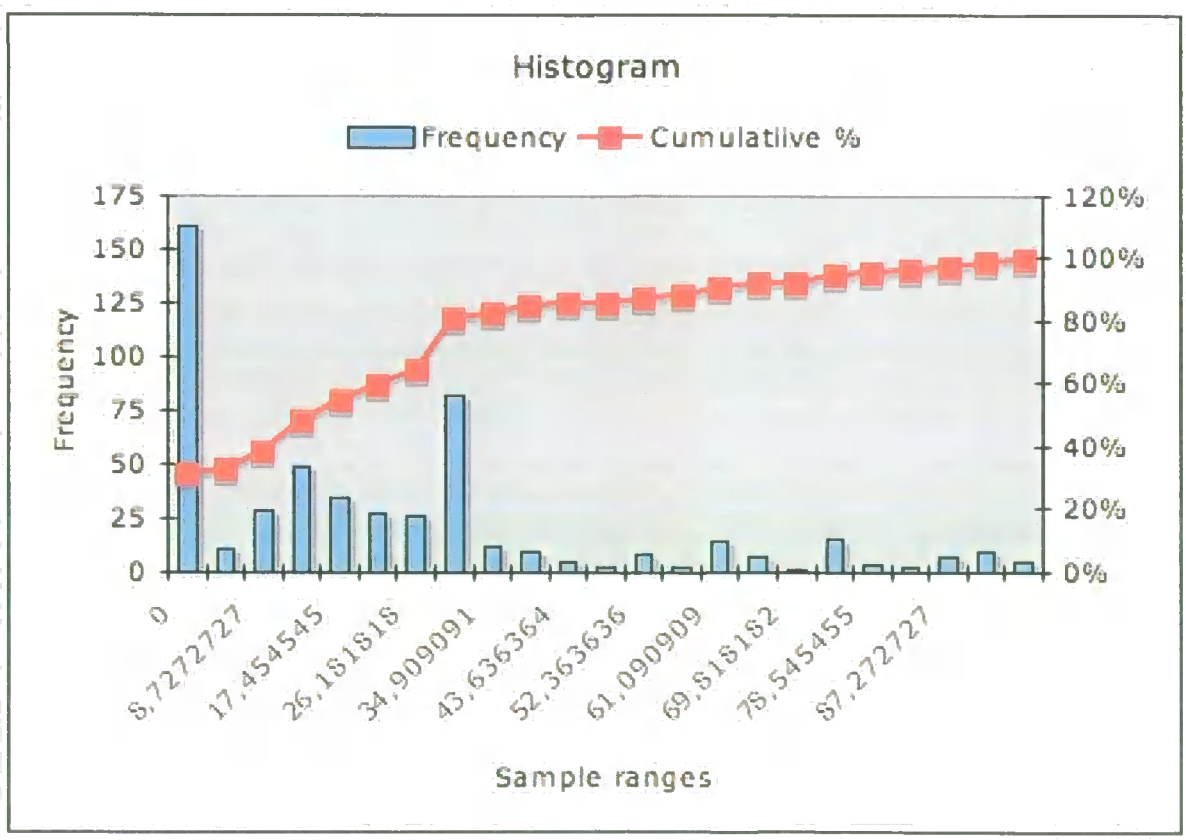


Figure 10.26: Histogram of levels of human-machine similarity in e6.

The data is reminiscent of Brownian noise but also reveals a certain low level periodicity. The relative strength of the periodicity is disclosed in the histograms in figures 10.25 and 10.26. The cumulative percentage stabilizes after the first nine sample ranges; after 90 percent of the data has been considered. As expected, zero values dominate in both experiments. An unanticipated characteristic minor peak occurs at the end of the incrementing cumulative percentage. In between the two peaks, the data shows a Gaussian-like distribution, as to be expected.

Figures 10.27 to 10.30 show data gathered at the end of every evolutionary epoch, just before a new population is generated. At those particular points in time, one may evaluate the efficiency of the motivations generated by the drives. Figure 10.27 shows a remarkable relationship of the drives total (averaged) integration levels for the two different system inclinations, social (blue) and selfish (red). The complementary nature of both signals proves a remarkable consistency in drives behaviour – and consequently assures coherent operation of how the drive selects functions in the compound-function pool. Both experiments use the same algorithm in the test-player; the melodies generated are thus not identical but only statistically similar. In most cases, the integration levels move in opposite directions, most prominent and consistent around generation 40. Both experiments are run individually yet the amplitudes of the changes nearly always synchronize perfectly. This reveals that the system has the capacity to repeatedly position itself at particular points in state space. Therefore, the system exhibits a certain musical *personality* since characteristic patterns occur in a consistent way. Also surprisingly, the complementary psychological nature of the two inclinations is made explicit by the behaviour of the drives pool.

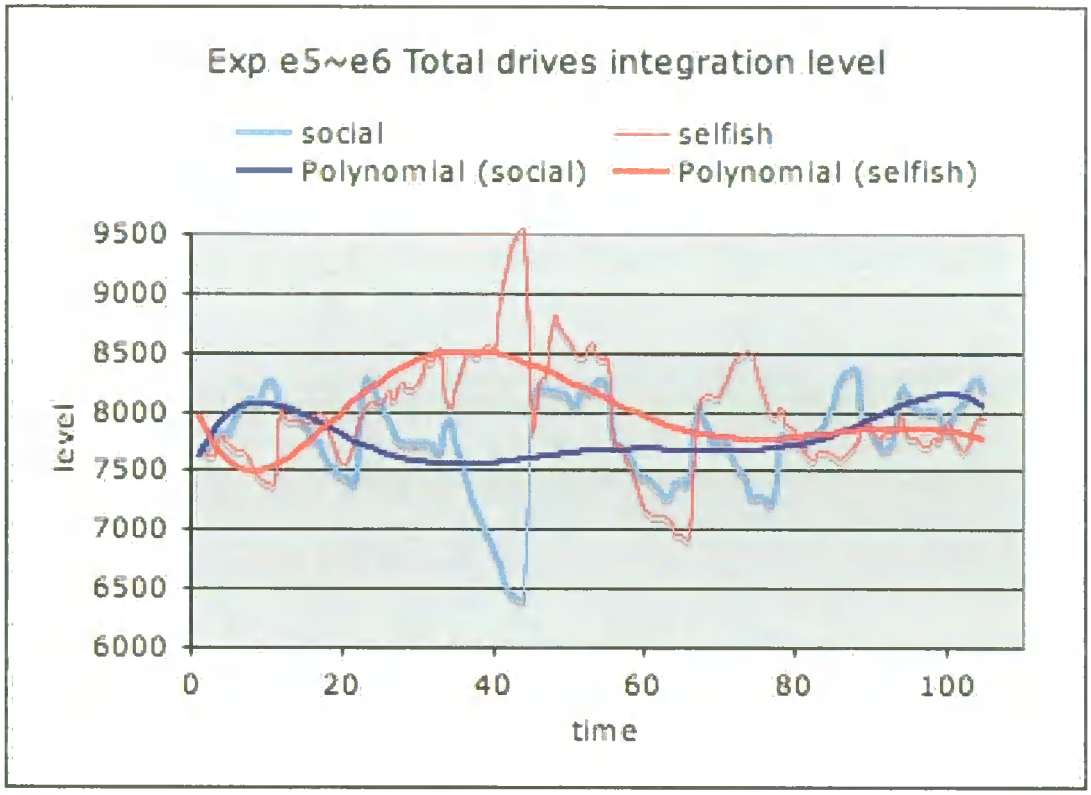


Figure 10.27: Drives total integrations level in e5 and e6.

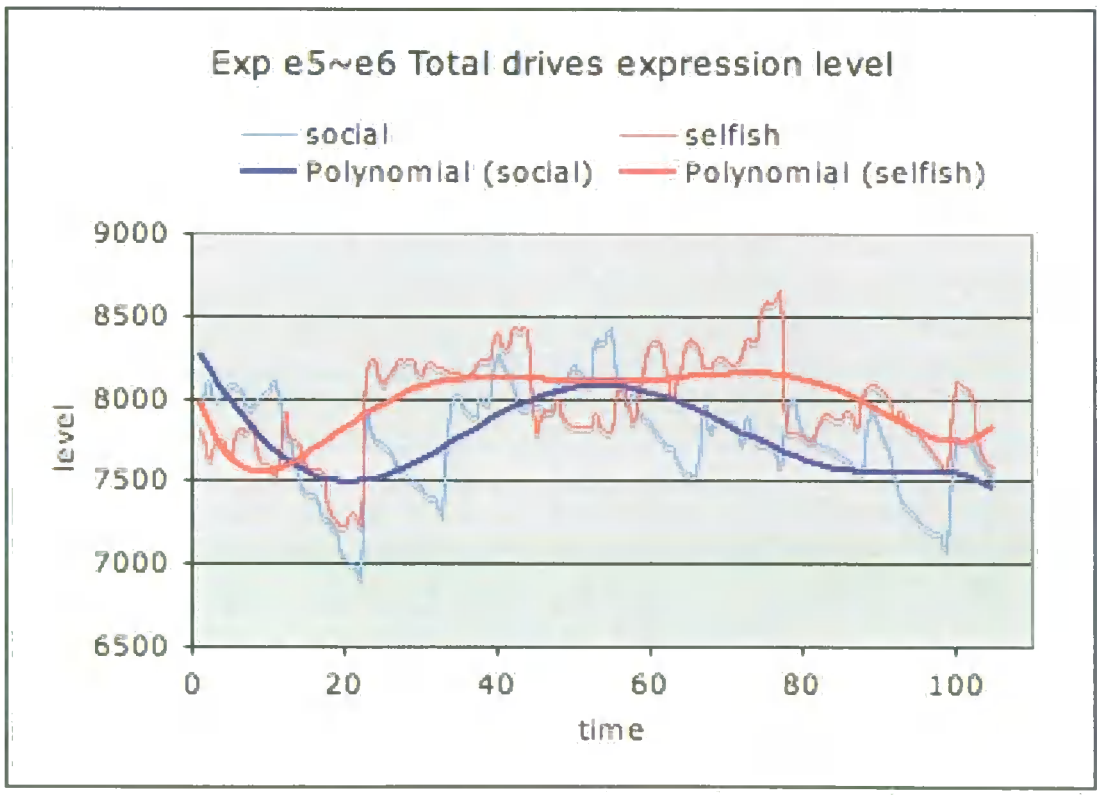


Figure 10.28: Drives total expression level in e5 and e6.

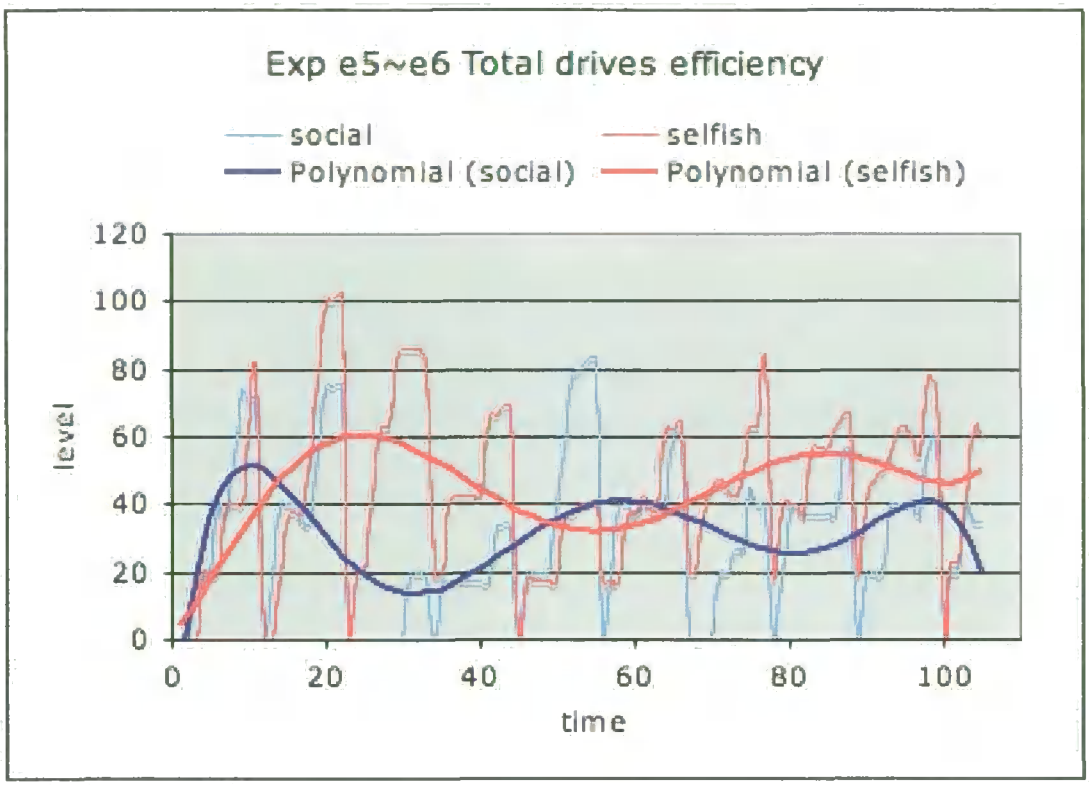


Figure 10.29: Drives total efficiency level in e5 and e6.

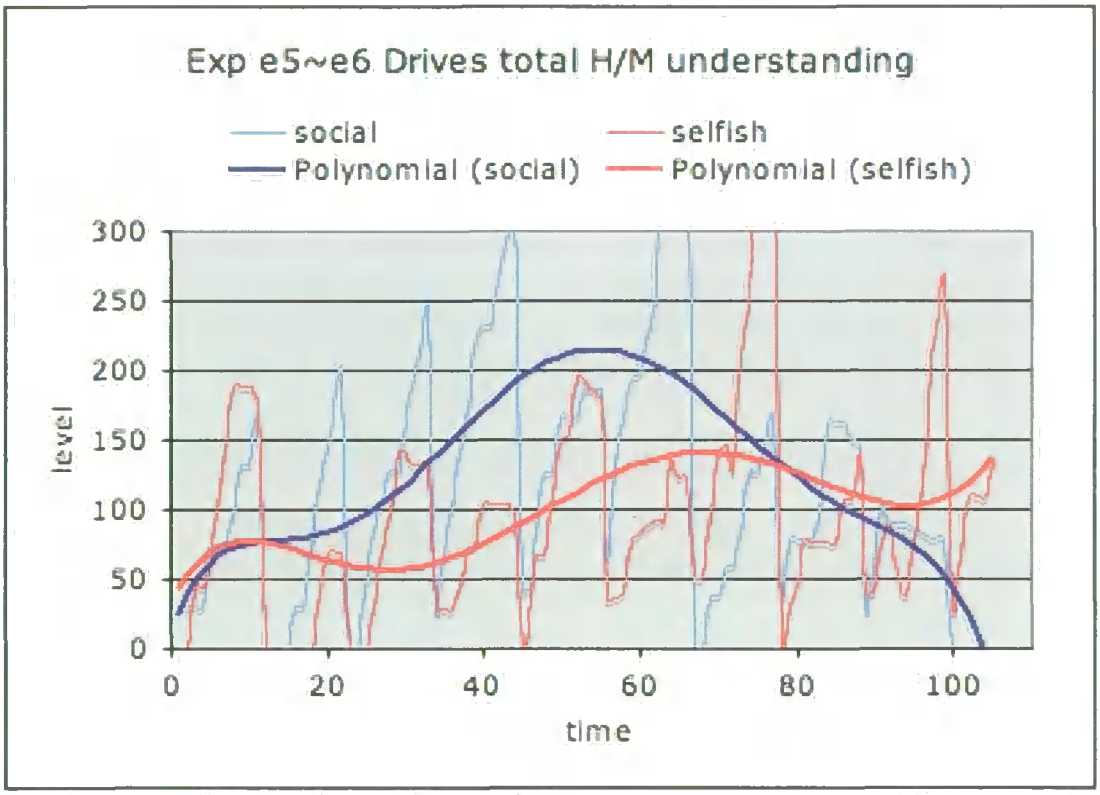


Figure 10.30: Drives total human/machine understanding in e5 and e6.

When turning to figure 10.28 one notices less articulation of the activity of the drives expression levels. Experiment e5 (social system inclination) shows extended areas of a relatively flat, continuous drive expression level. The drives pool as a whole stabilizes on a point attractor. Experiment e6 (selfish inclination) clearly suggests expression levels that gradually decrement in between any two breeding points.

Integration *and* expression levels generated from a selfish inclination manage to globally supersede the levels given a social inclination. This observation suggests that selfish inclination provides the best results; i.e., the maximum signal amplitudes, whatever the systems' orientation. In other words, both experiments prove that the drive object can tune itself to be in harmony with its implied orientation (integration or expression).

Figure 10.29 shows the history of the total efficiency of all drives in the drives pool. Both polynomials suggest a low frequency oscillation pattern with 1 or 2 cycles over 105 evolutionary steps. The efficiency levels are much slower to adapt than the integration or expression levels. The same observation goes for the drives total human/machine understanding as depicted in figure 10.30. Social inclination involves an arch-like evolution of the understanding level. The blue polynomial shows an initial strong increase of understanding level followed by a prompt descent. The unmistakable shape of his pattern underpins the instability and competitive nature of the underlying tension in the man-machine interaction. Given a selfish systems inclination (red polynomial), the signal is of much lower amplitude and slowly moves up in a near linear way. This piece of evidence corresponds remarkably with the idea of a selfish oriented system; the level of understanding manages to increase very slowly on a relative scale from 50 to about 150.

10.4 Experiments e7 to e10: A Series of Systematic Experiments

10.4.1 Introduction

The following experiments aim to acquire understanding of Oscar's behavioural space given a human interactor while considering the impact of two main parameters; the system inclination and whether learning is enabled or not. Additional parameters are as follows; the performance-cycle (figure 10.1) equals 320 with 16 breeding-steps implying an evolutionary process spanning a total of 20 epochs. The population size (N) of all system components (SAN, patcher, CF-pool, and the drives-pool) is eight objects.

Interaction is evaluated according to the real-time learning-mode; the learning process is informed by the exploration/exploitation ratio in real-time. This ratio decides on how a drive is selected, it influences the relationship between the amount of time the system focuses on the drives that proved to give rise to productive results in the past (drives having fitness greater that zero) and, on the other hand, selecting drives at random hoping for the discovery of the potentially interesting results they might entail.

The learning-cycle parameter is one, thus every CF is applied only once per learning-step. The same human subject provides input for all four experiments via a standard MIDI keyboard. A large number of state variables reflecting the current system's activity is traced in real-time and written to disk. As explained above, the tracer-file and the evolution-file offer two perspectives of the system's behaviour in time. Only information from the evolution-file is taken up here; all four experiments providing 169 data samples. In addition, the music input to the system and the output of the system are also captured and saved as MIDI files for later analysis. Each

interactive session took approximately 40 minutes. Parametric conditions and the number of collected output data samples are summarized in tables 10.7 and 10.8.

Exp	N	Steps Learn	Steps Breed	Steps Perform	System Inclination	Learn	Input	Learning-mode
e7	8	1	16	320	Social	Yes	Human	Real-time
e8	8	1	16	320	Selfish	Yes	Human	Real-time
e9	8	1	16	320	Social	No	Human	n.a.
e10	8	1	16	320	Selfish	No	Human	n.a.

Table 10.7: Parameter settings for experiments e7 to e10.

Experiment	**e7**	**e8**	**e9**	**e10**
Tracer samples	755	776	797	787
Evolution samples	169	169	169	169
Session duration	38'24"	44'23"	39'04"	41'57"

Table 10.8: Number of samples and durations of experiments e7 to e10.

10.4.2 Experiments 7 and 8: Visualisation of Tracer Data

This section presents a visualisation of the *tracer data* gathered during the four interaction experiments shown in table 10.8. The subsequent section (10.4.3) provides a visualisation of the *evolution data* collected from the same experiments. Finally, the next following section (10.4.4) addresses analysis and interpretation of the actual data in a variety of ways.

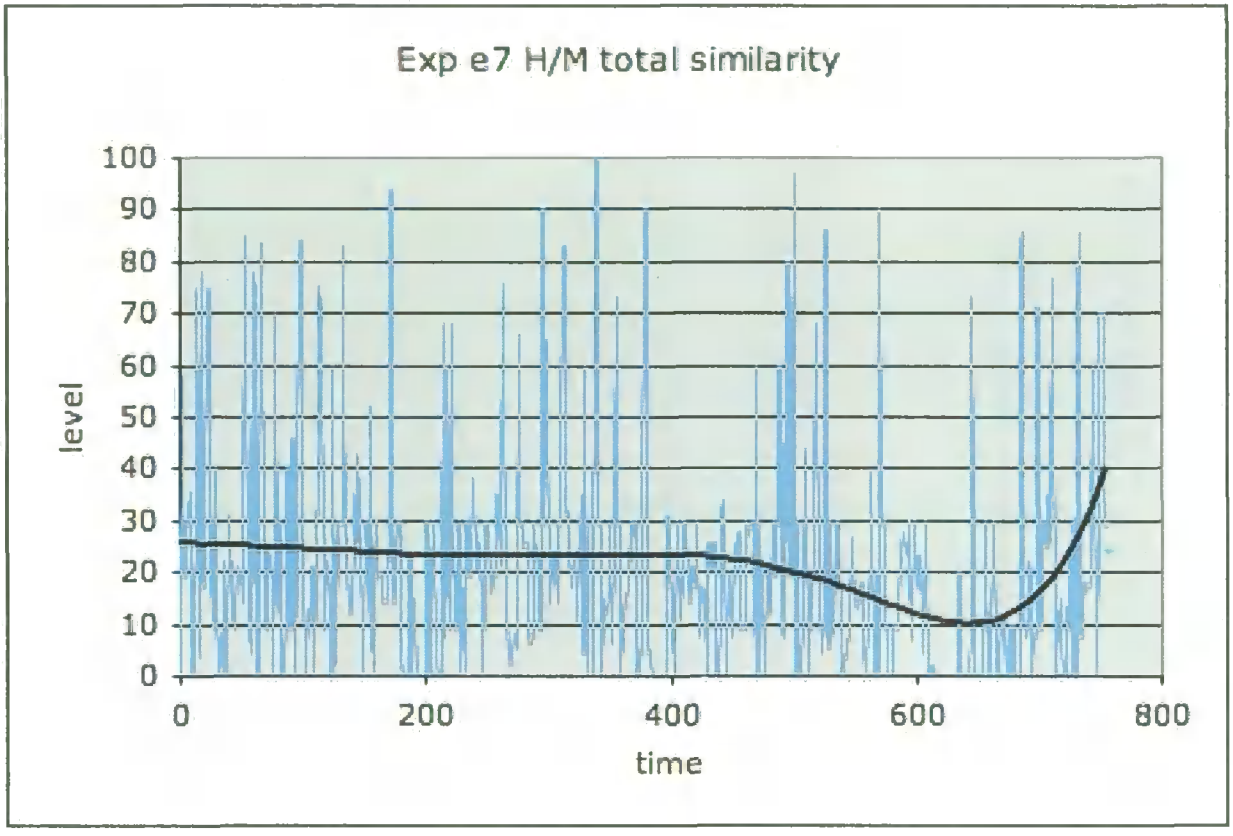


Figure 10.31: History of human-machine total similarity in e7.

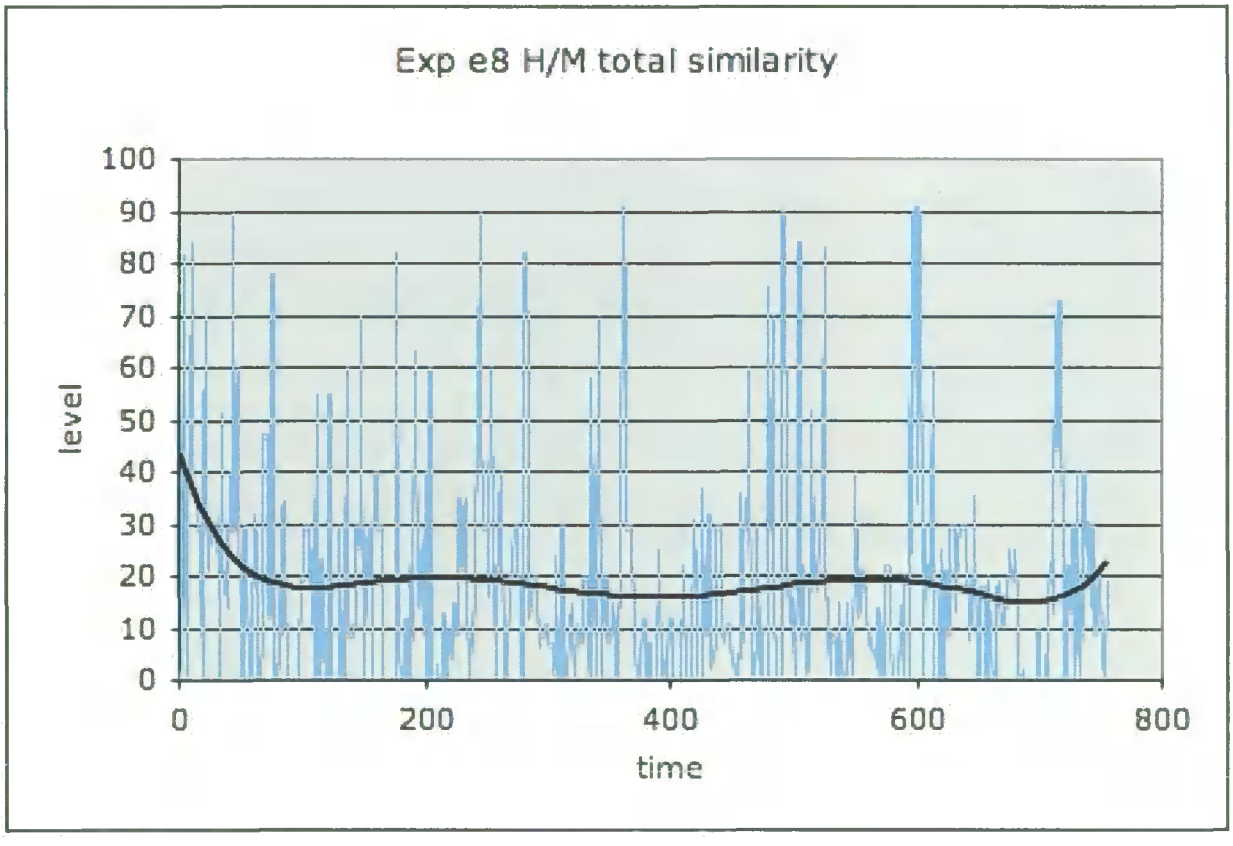


Figure 10.32: History of human-machine total similarity in e8.

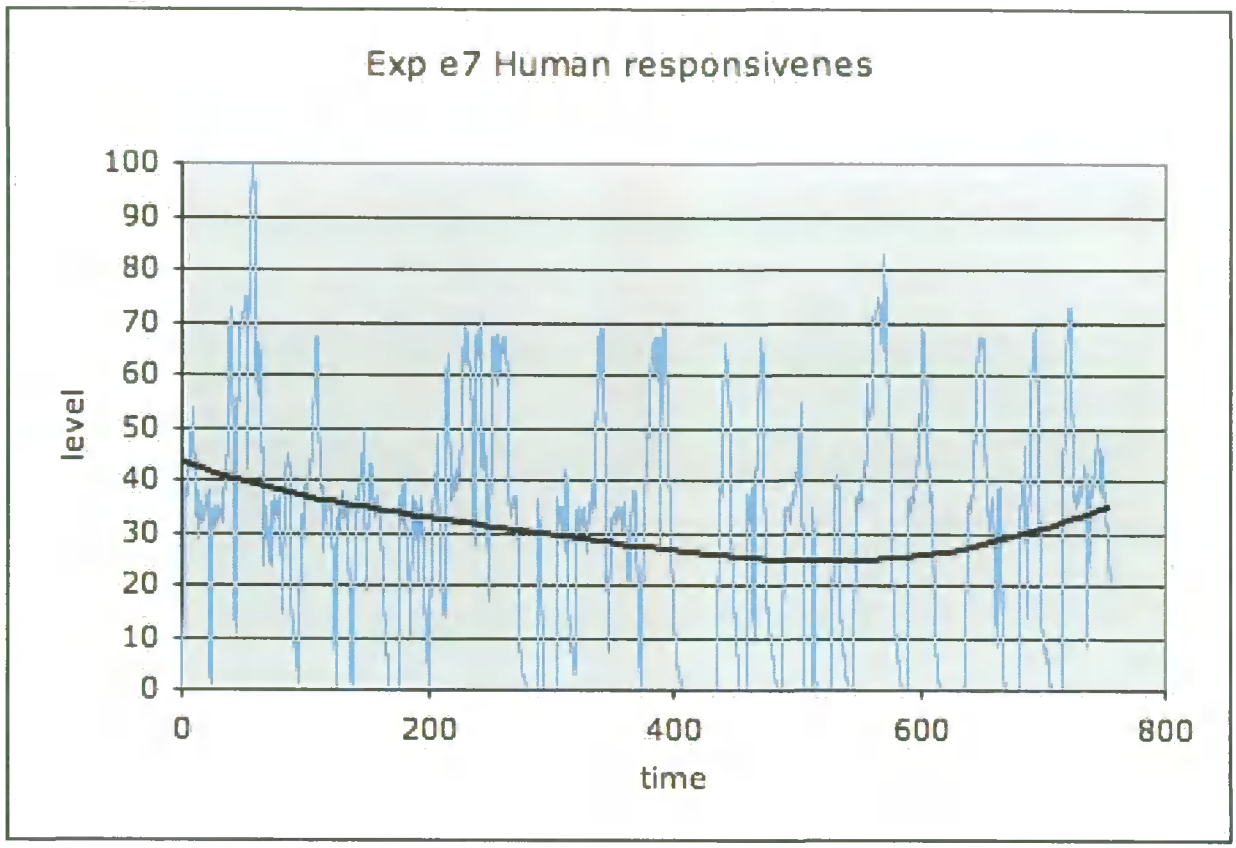


Figure 10.33: History of human responsiveness in e7.

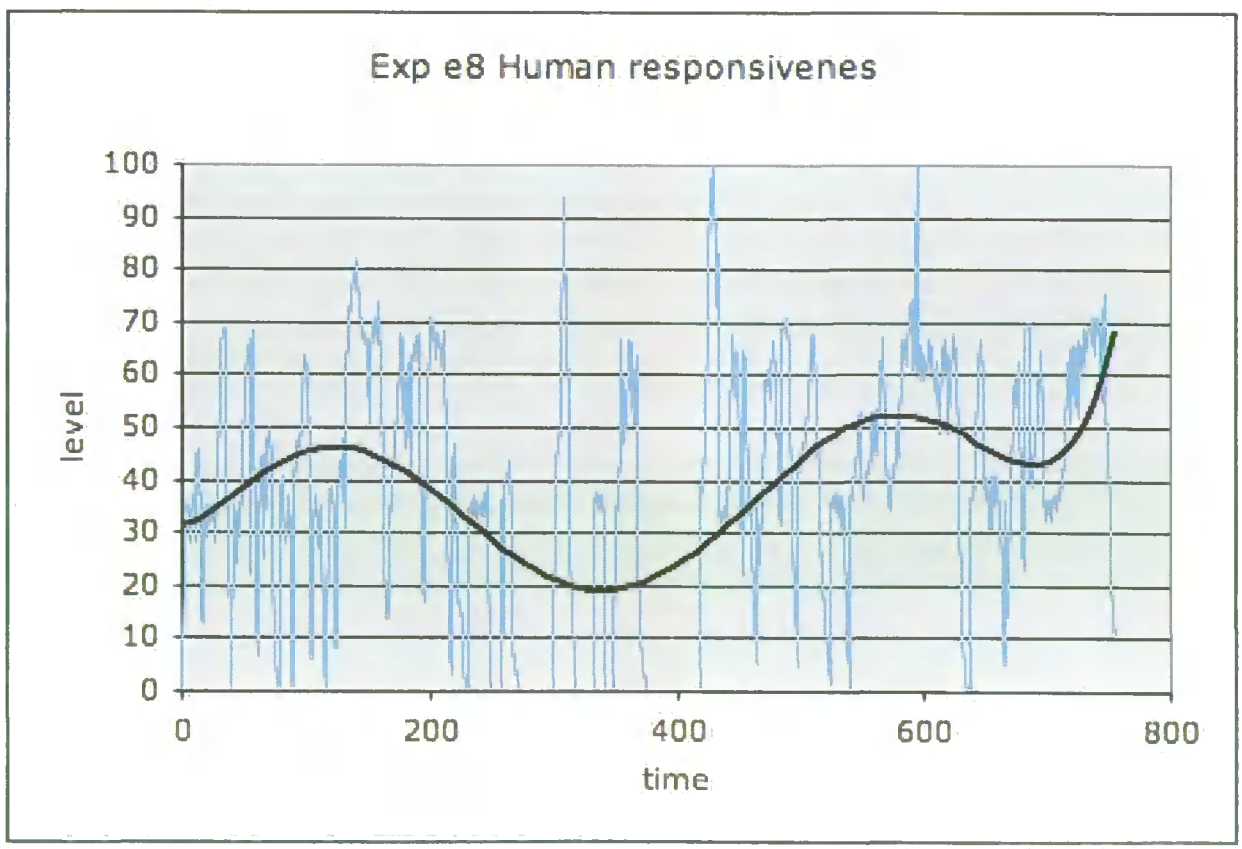


Figure 10.34: History of human responsiveness in e8.

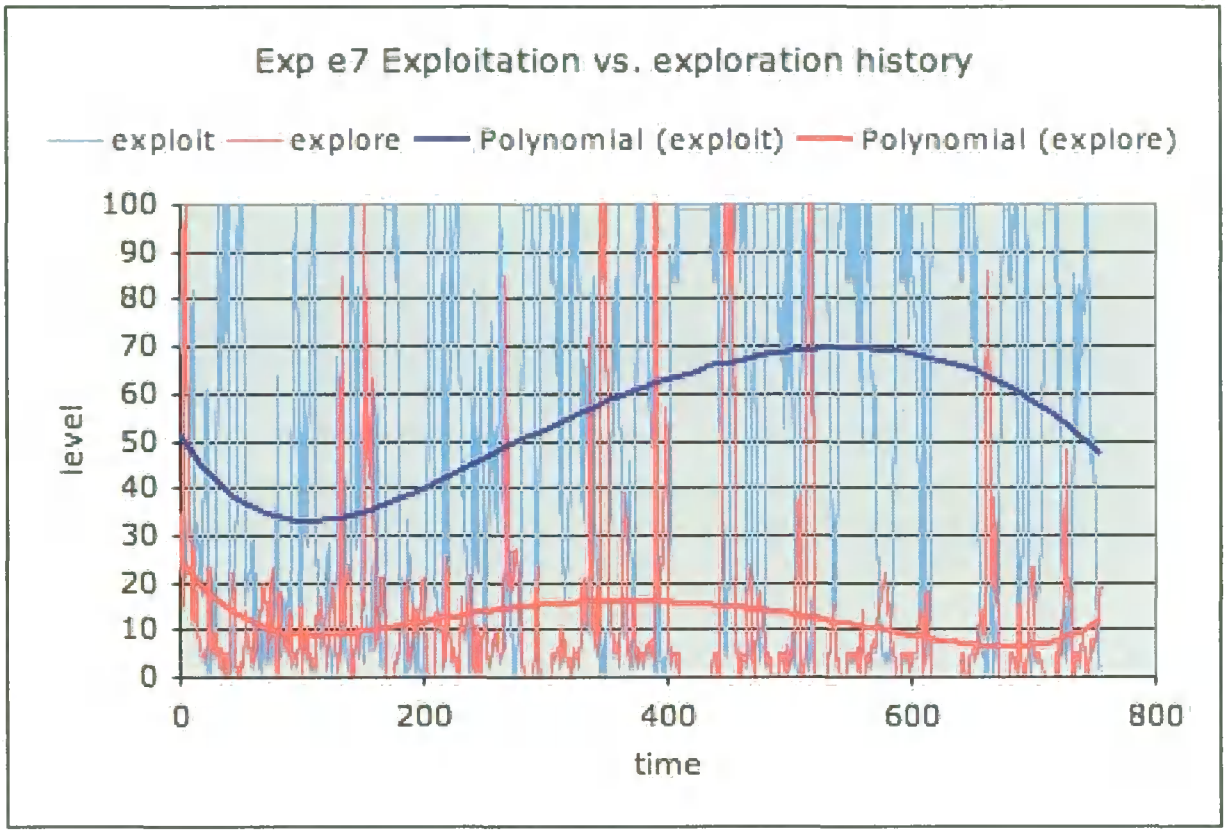


Figure 10.35: Exploitation vs. exploration pressures in e7.

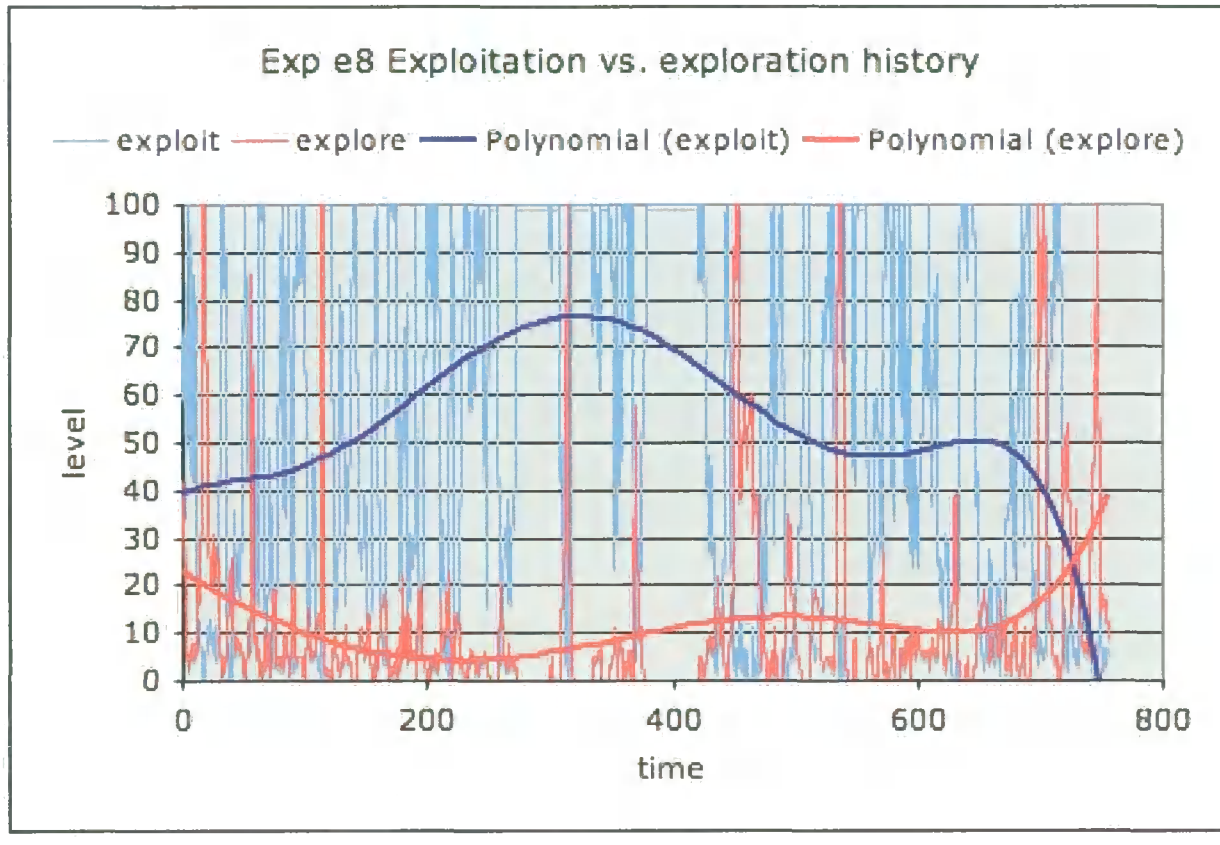


Figure 10.36: Exploitation vs. exploration pressures in e8.

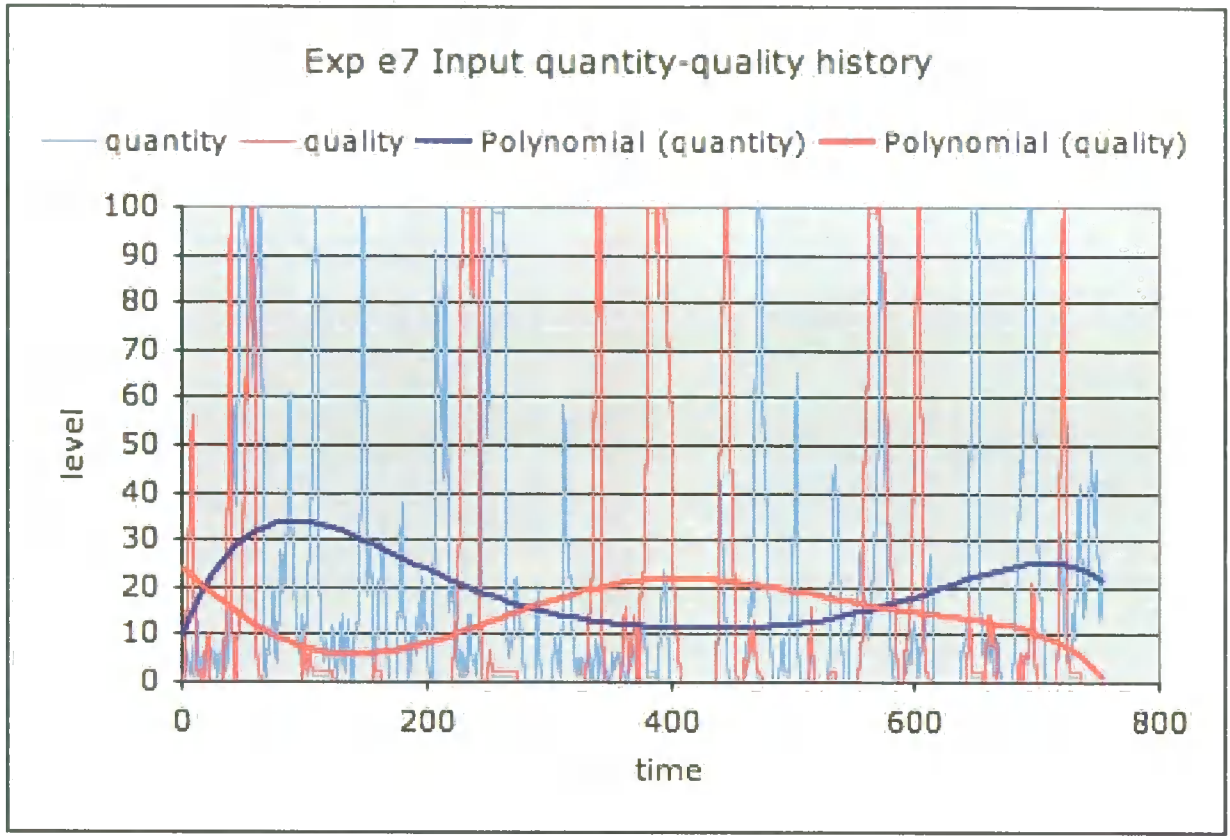


Figure 10.37: Quantity vs. quality of human input in e7.

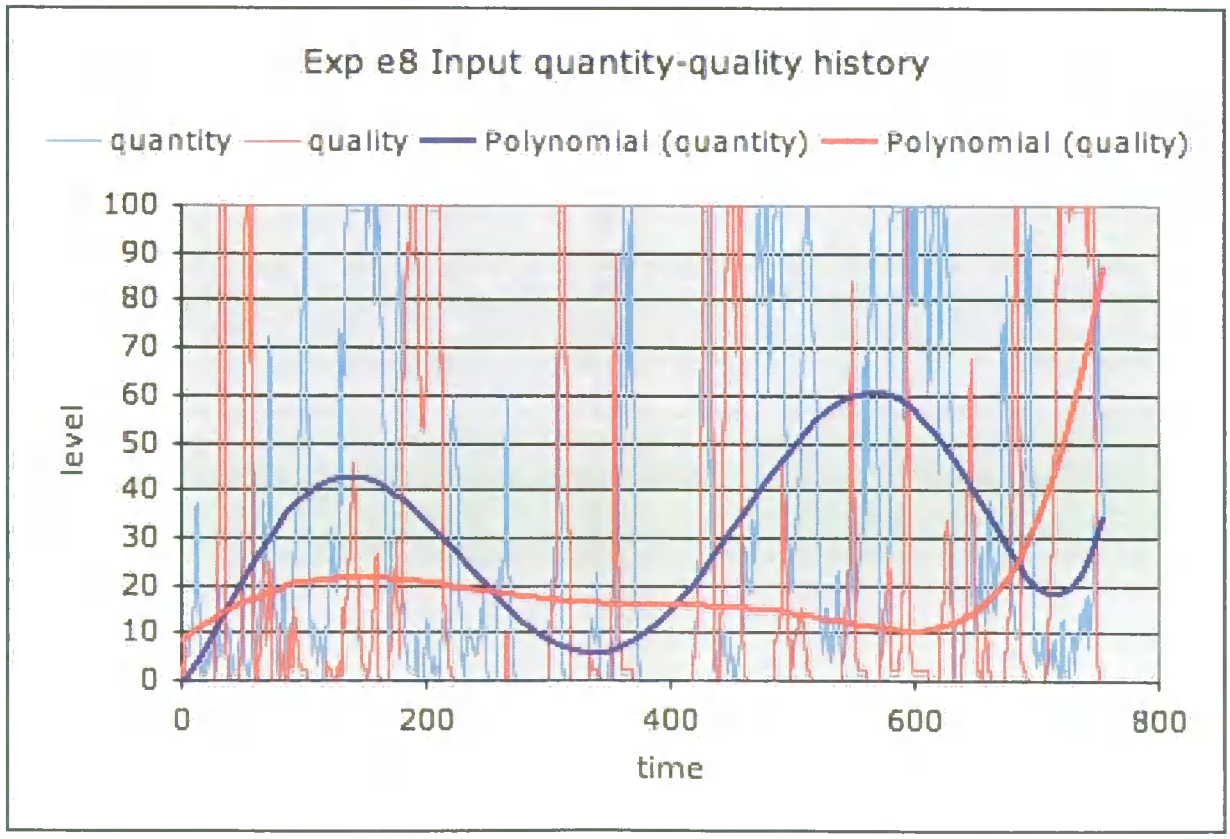


Figure 10.38: Quantity vs. quality of human input in e8.

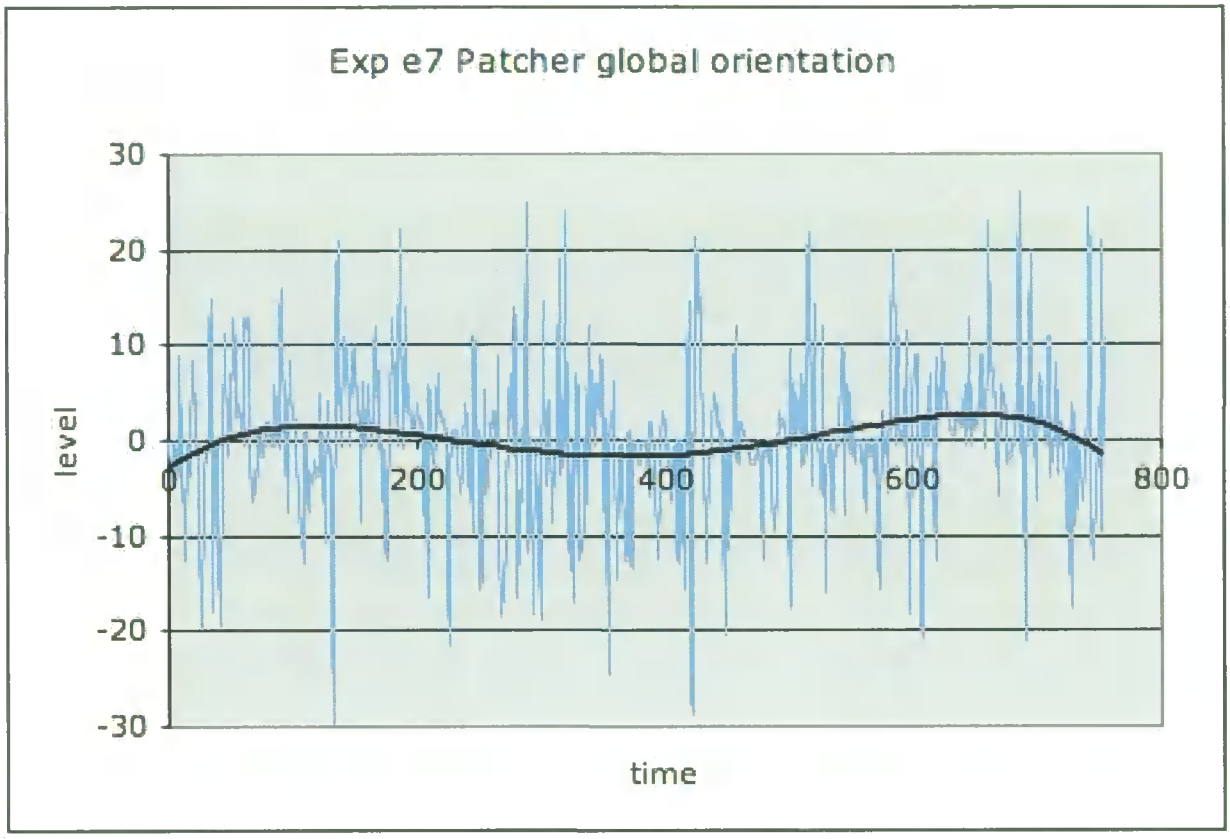


Figure 10.39: Patcher global orientation in e7.

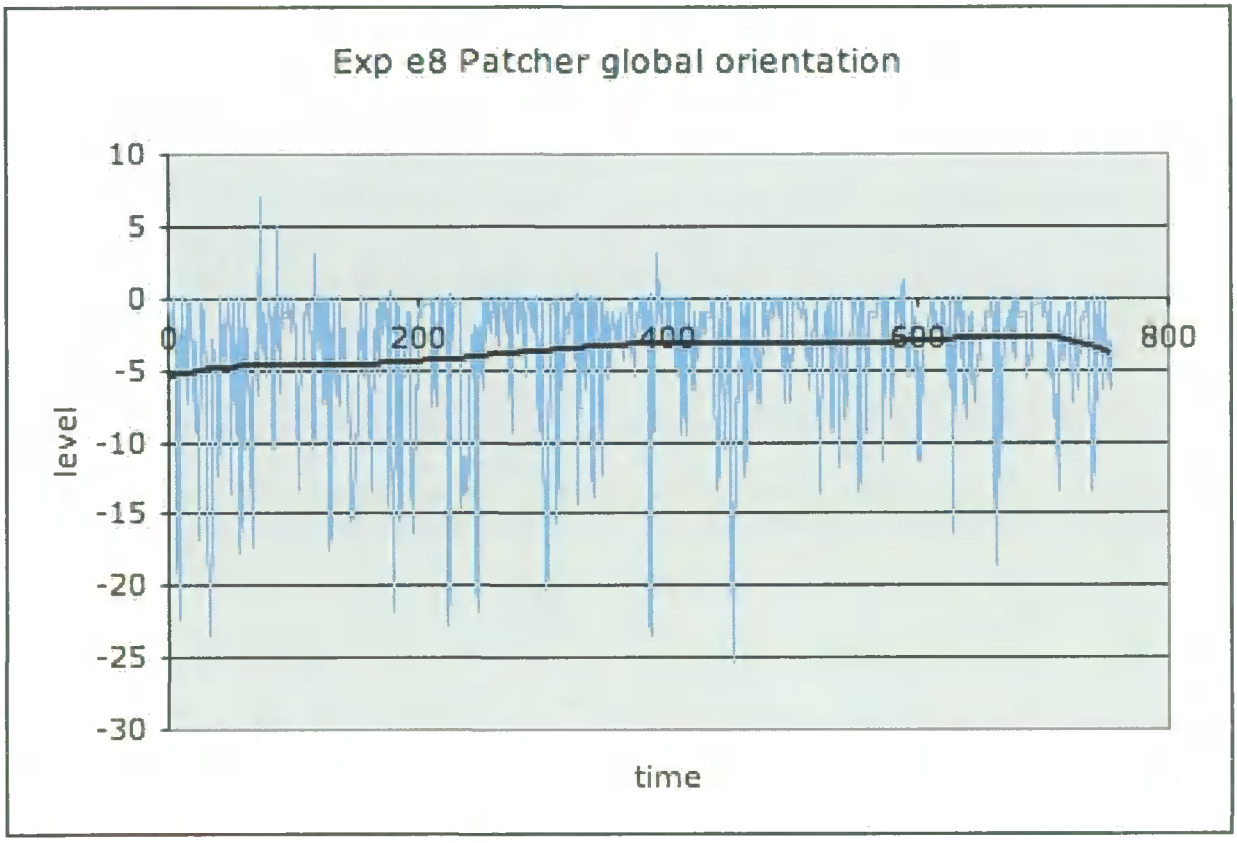


Figure 10.40: Patcher global orientation in e8.

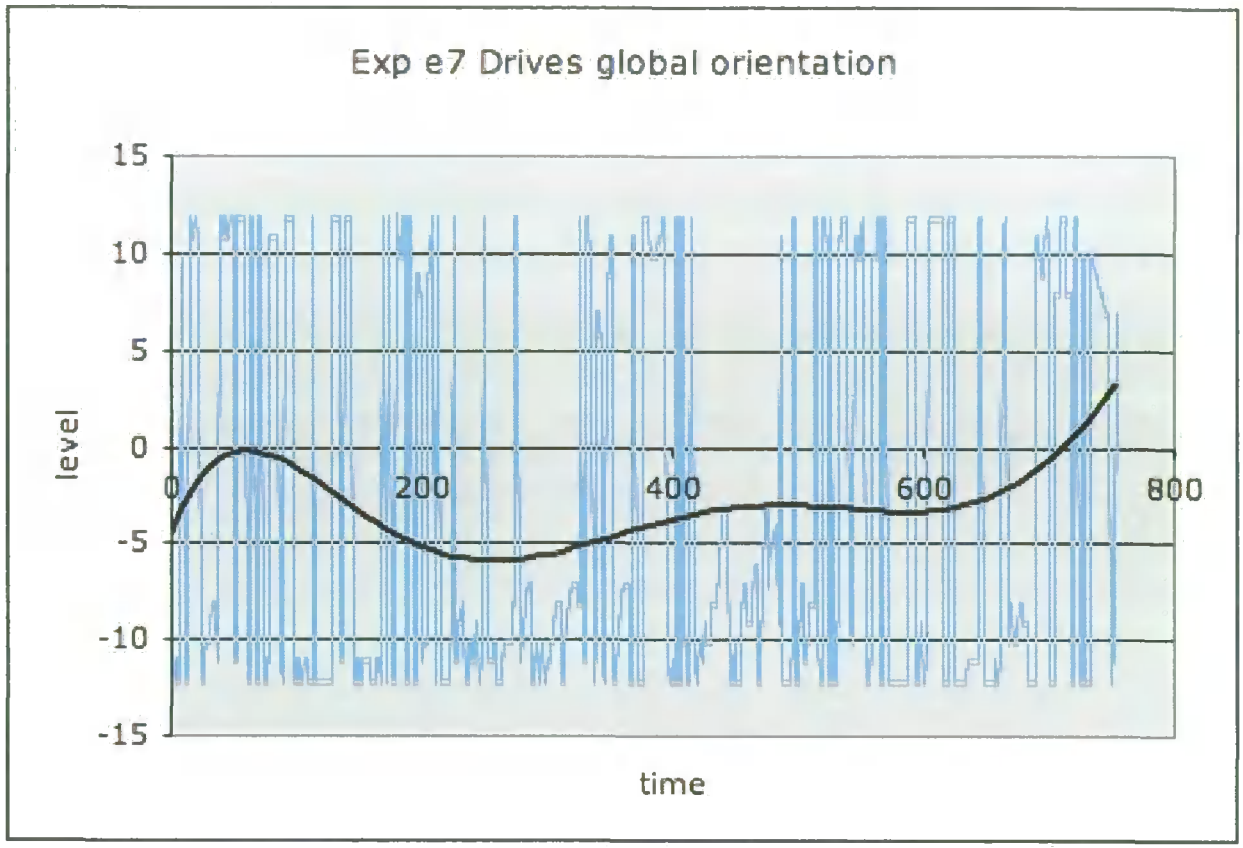


Figure 10.41: Drives-pool global orientation in e7.

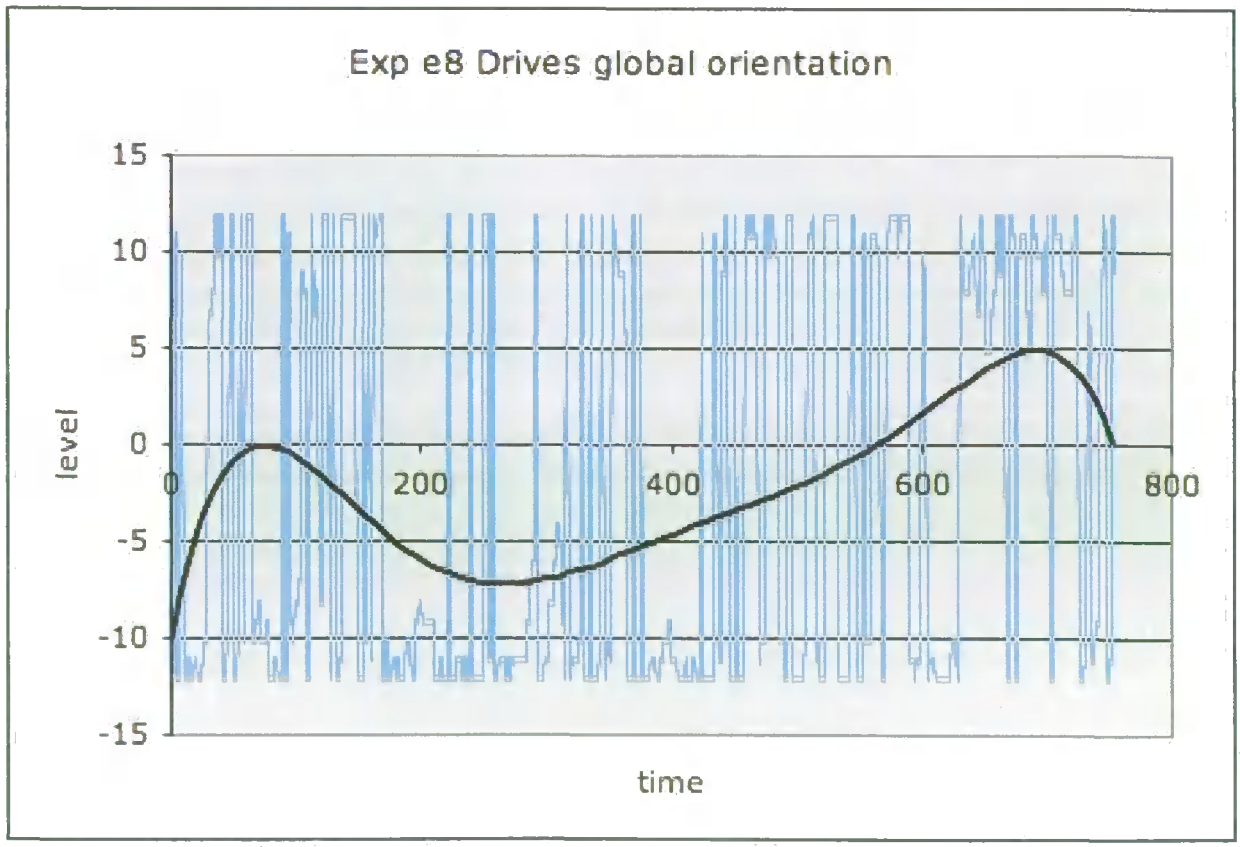


Figure 10.42: Drives-pool global orientation in e8.

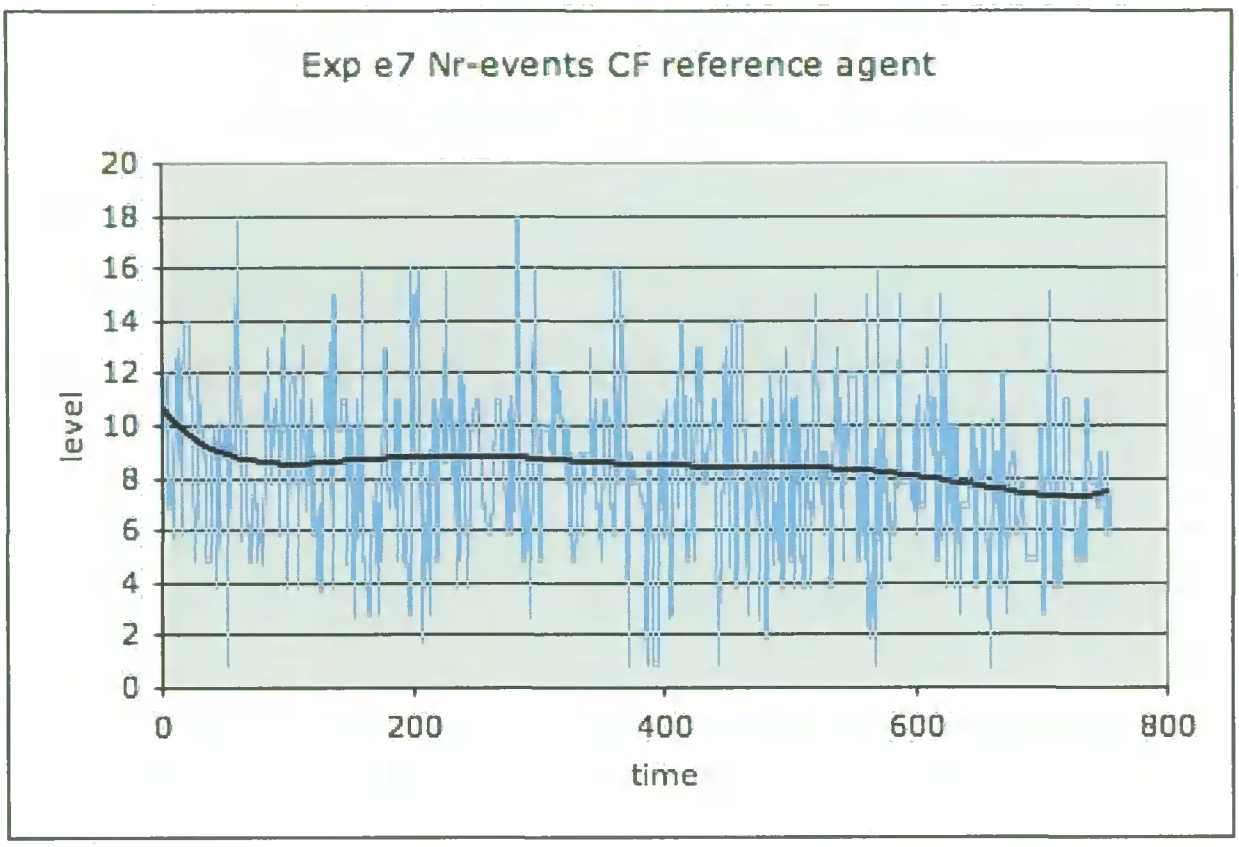


Figure 10.43: Number of events in the compound-function held by the reference agent in e7.

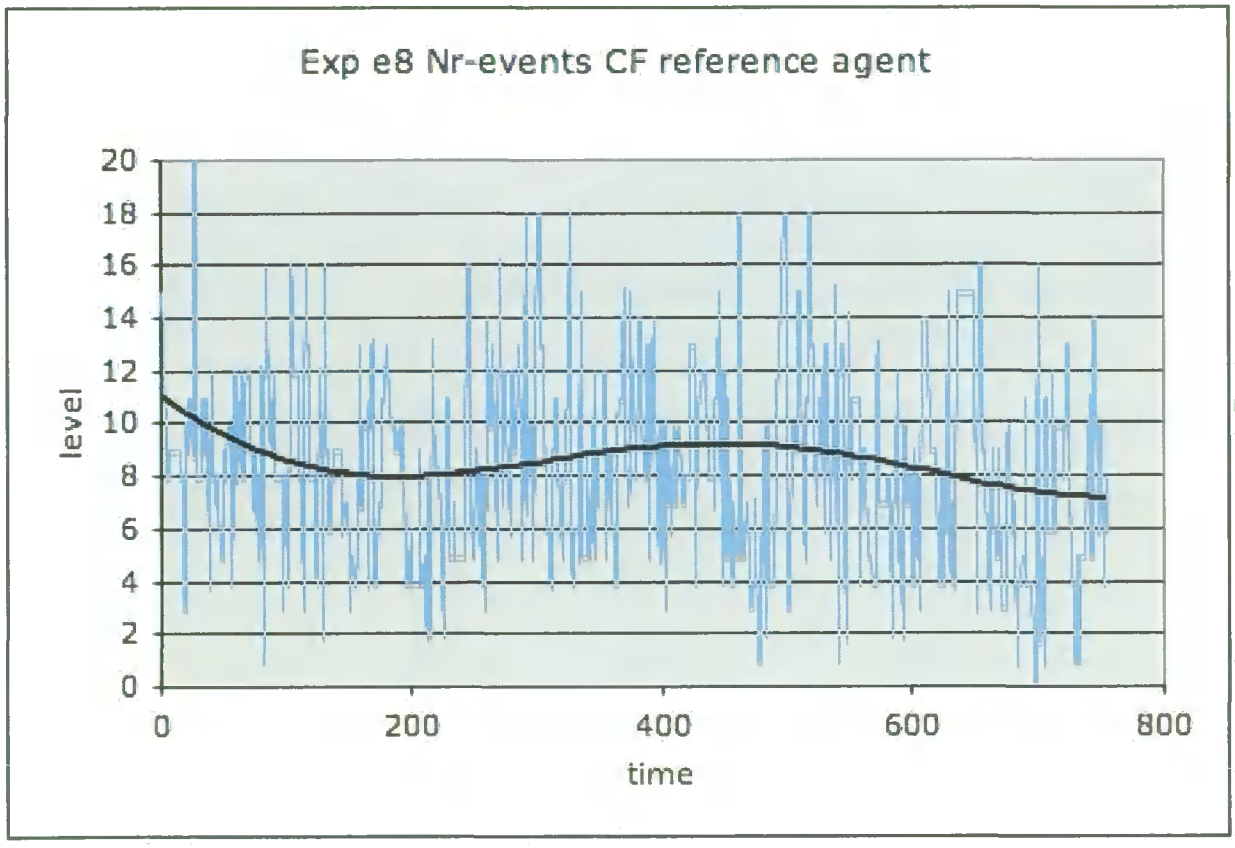


Figure 10.44: Number of events in the compound-function held by the reference agent in e8.

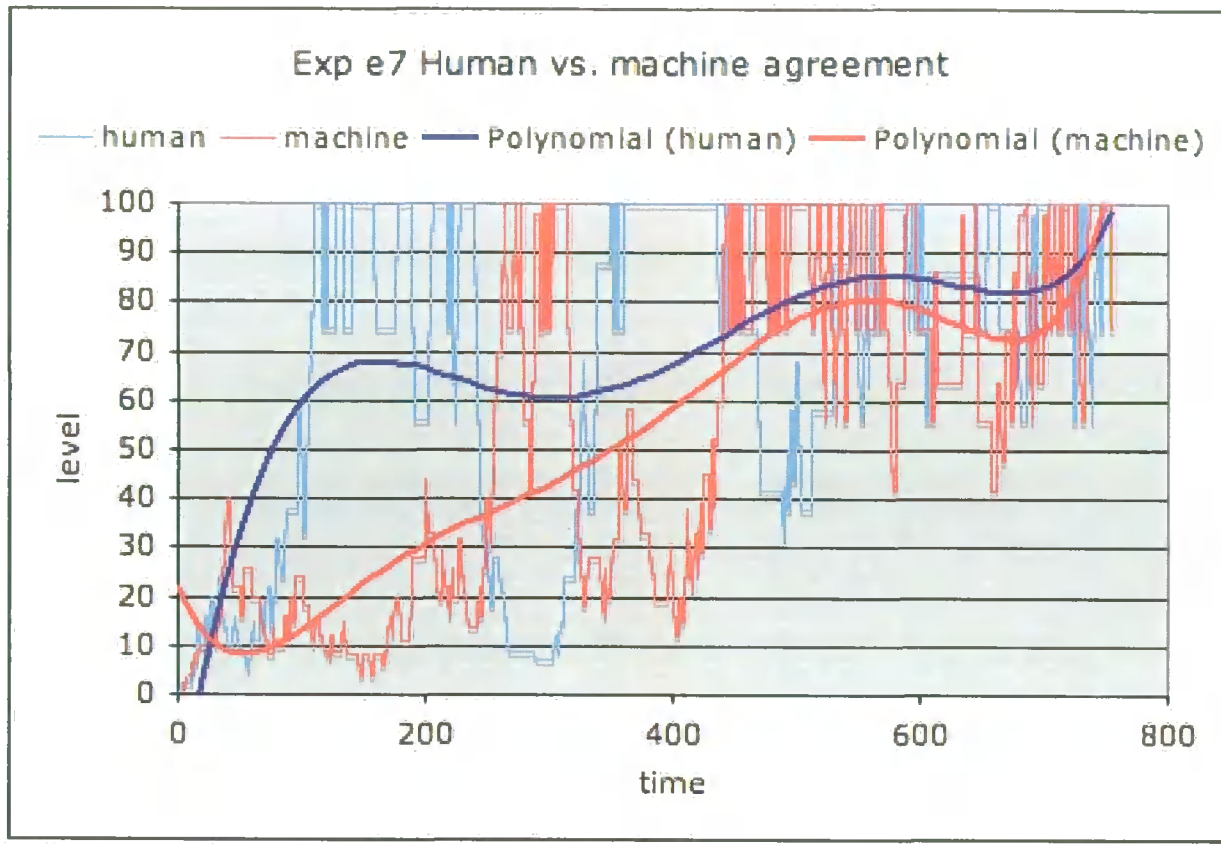


Figure 10.45: Human vs. machine agreement in e7.

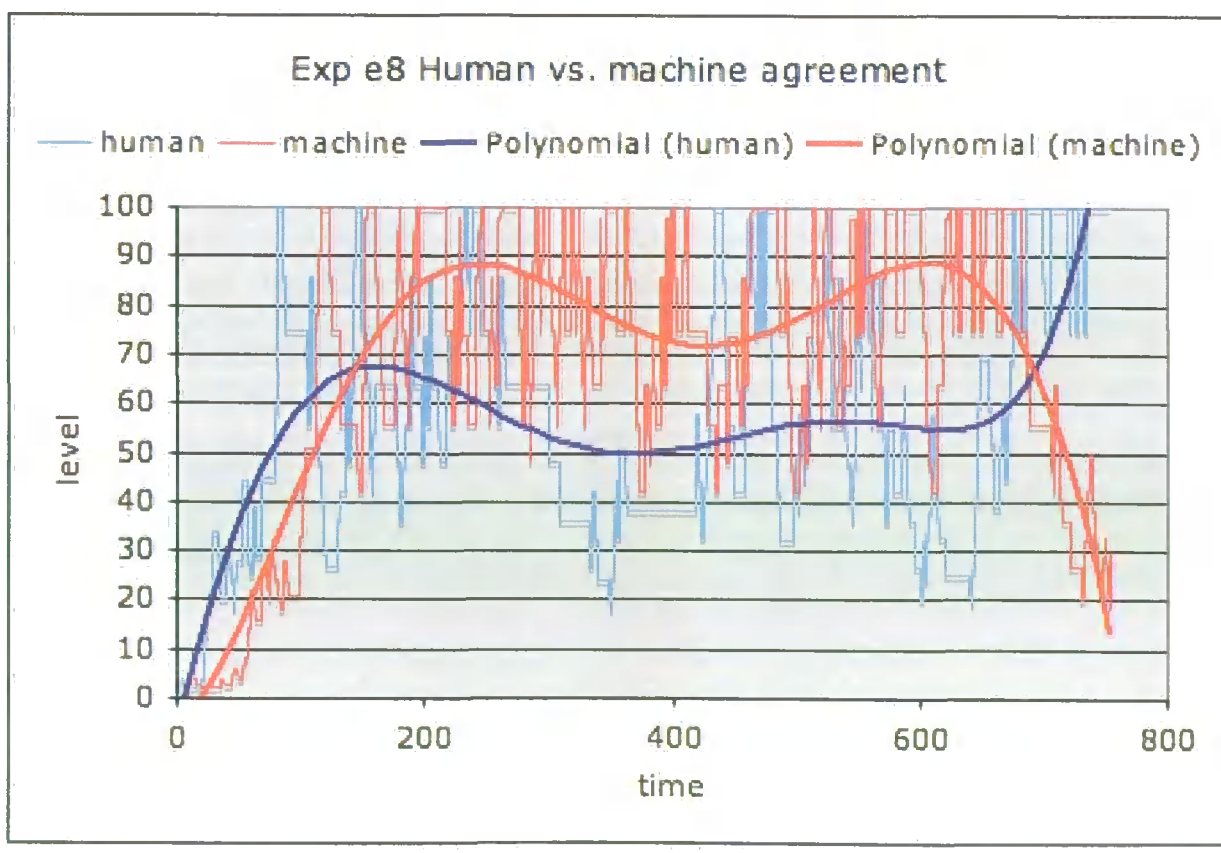


Figure 10.46: Human vs. machine agreement in e8.

10.4.3 Experiments e7 and e8: Visualisation of Evolution Data

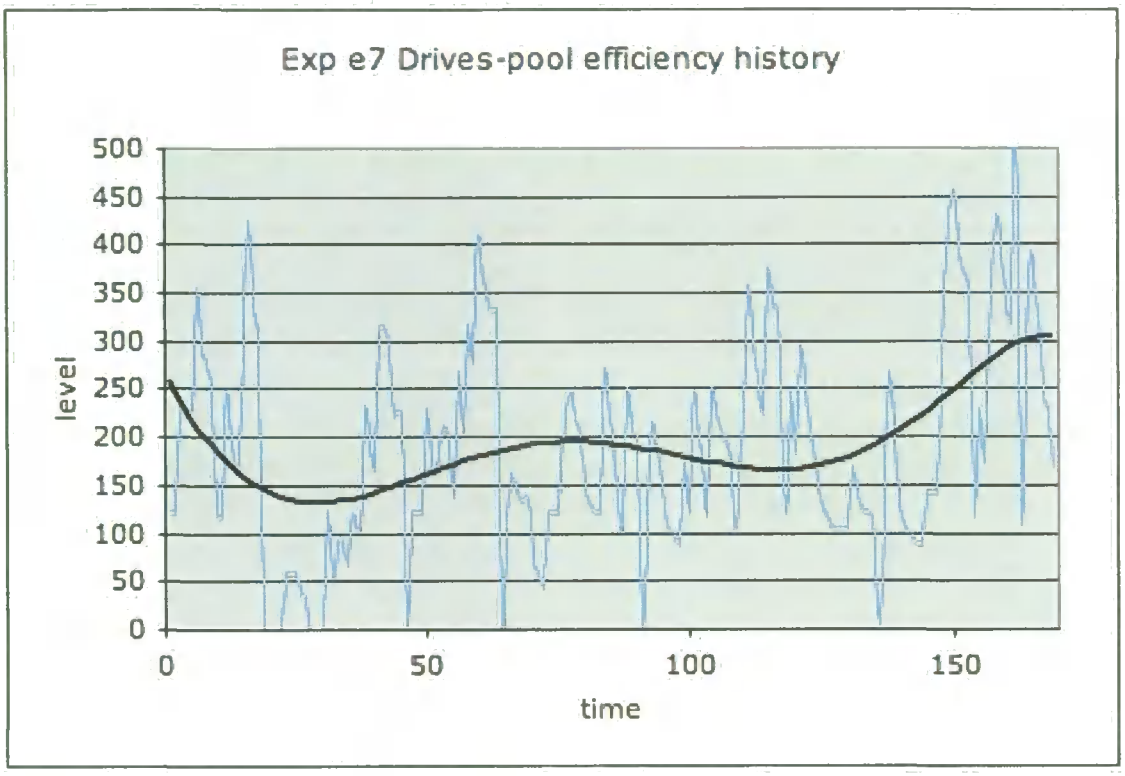


Figure 10.47: History of drives-pool efficiency in e7.

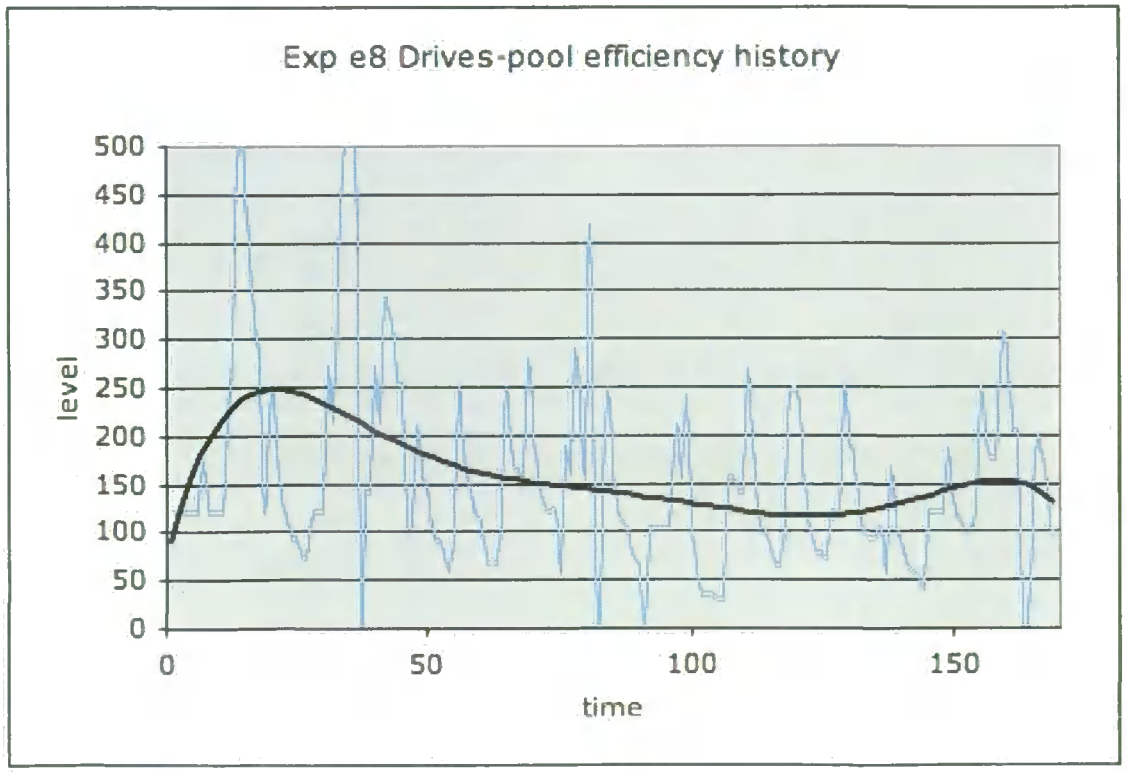


Figure 10.48: History of drives-pool efficiency in e8.

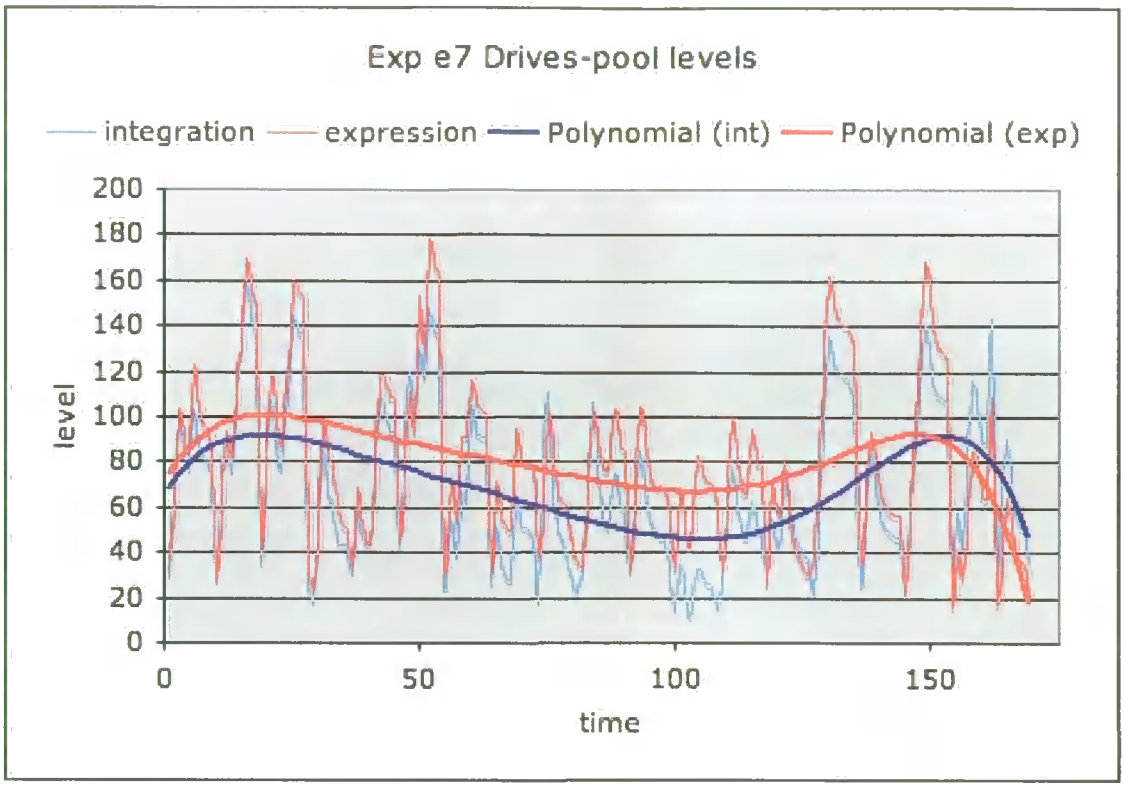


Figure 10.49: Drives-pool output levels in e7.

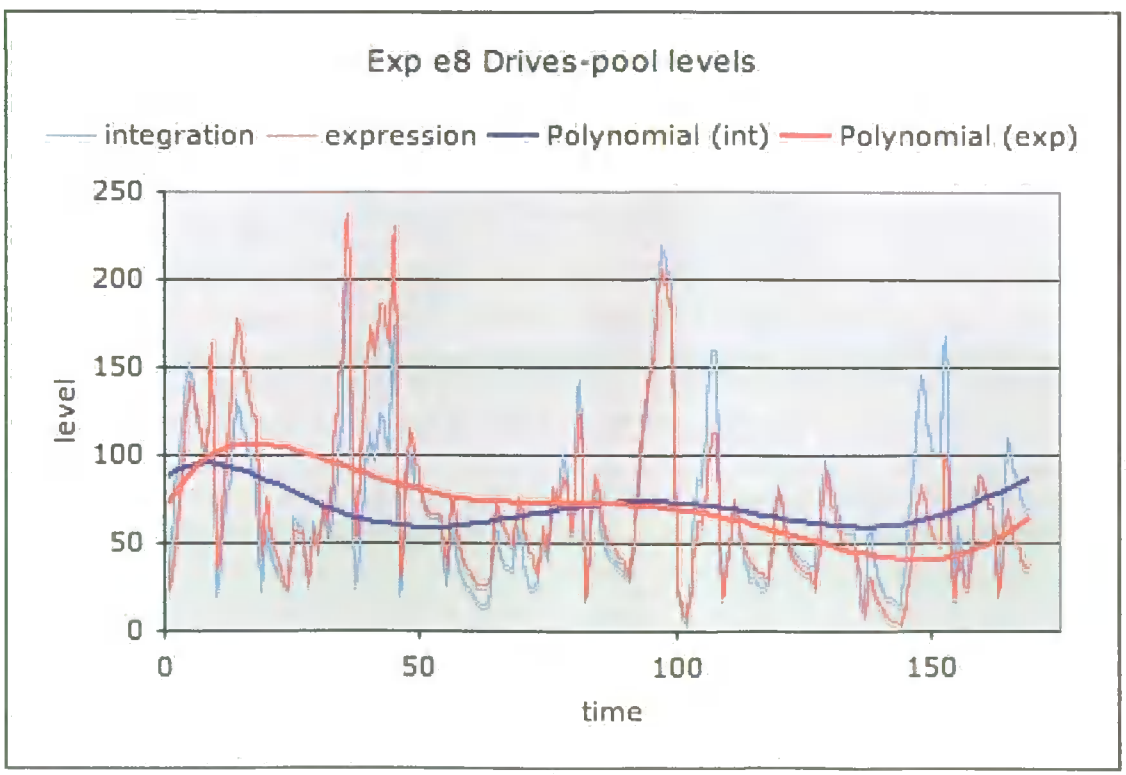


Figure 10.50: Drives-pool output levels in e8.

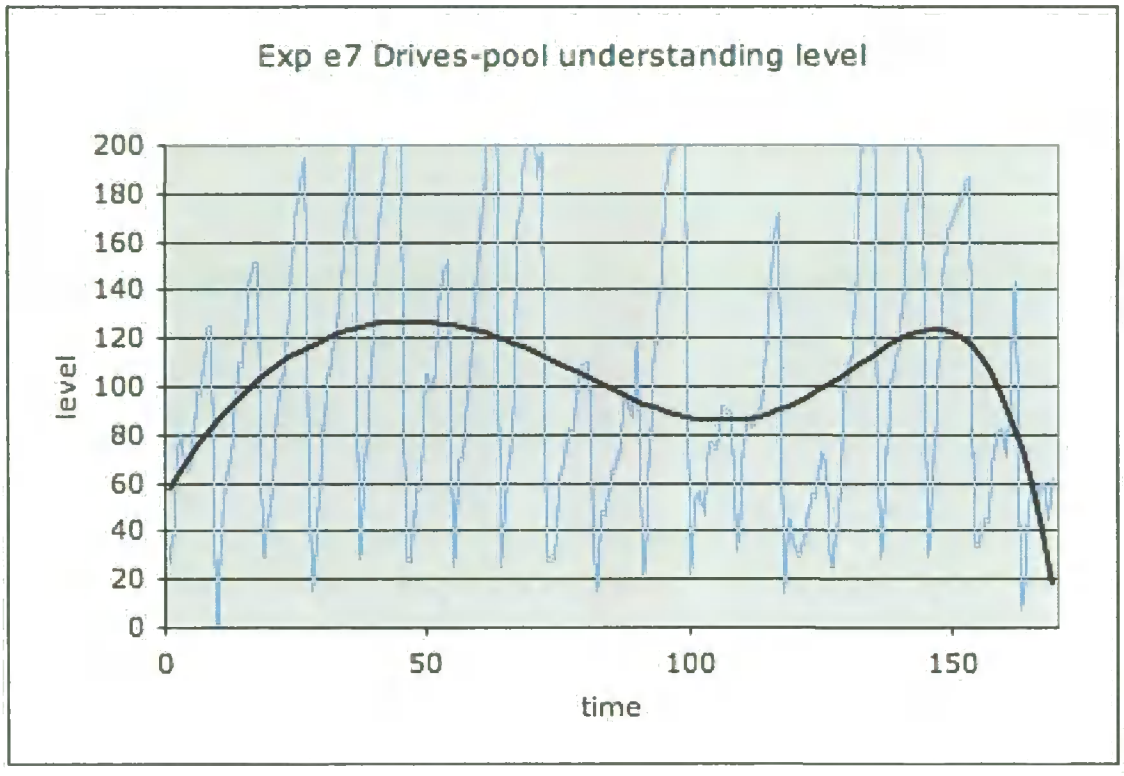


Figure 10.51: History of drives-pool understanding in e7.

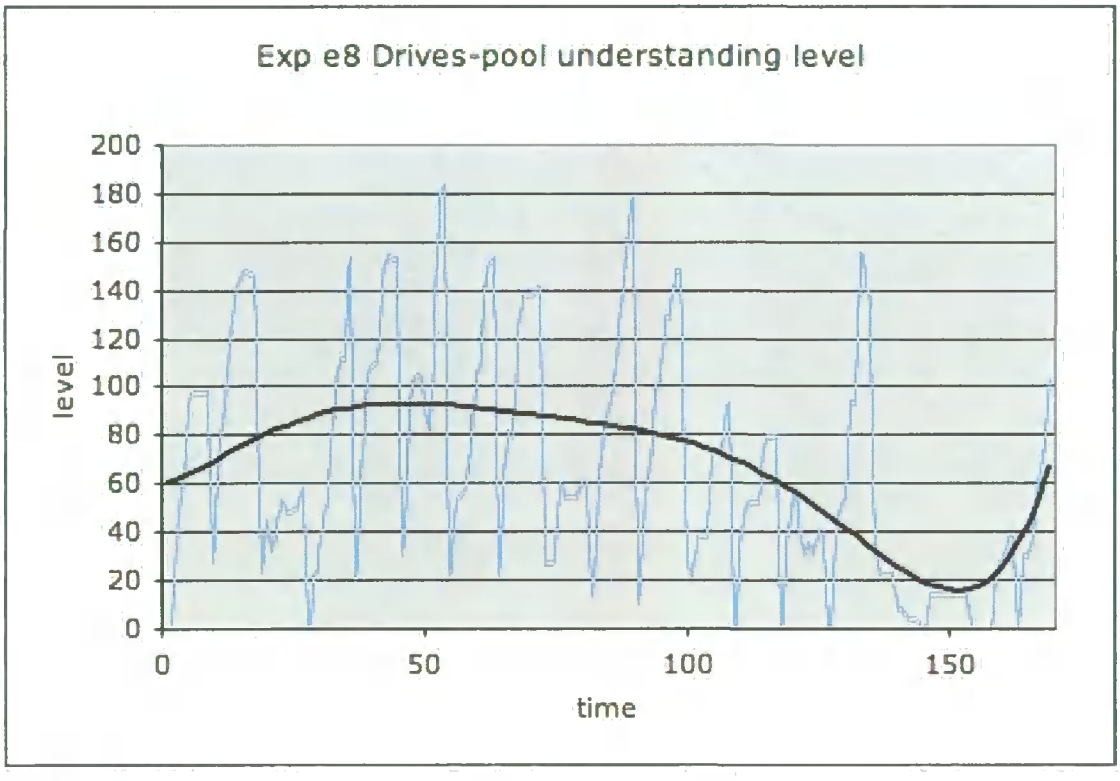


Figure 10.52: History of drives-pool understanding in e8.

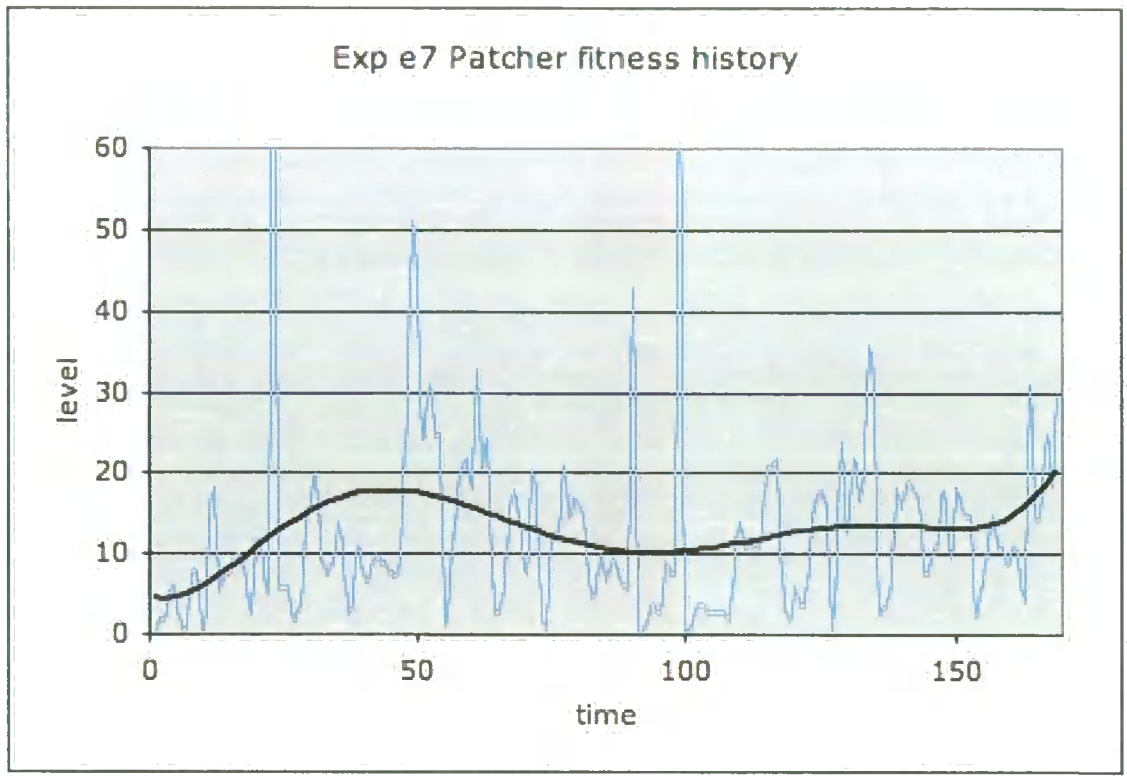


Figure 10.53: History of patcher fitness in e7.

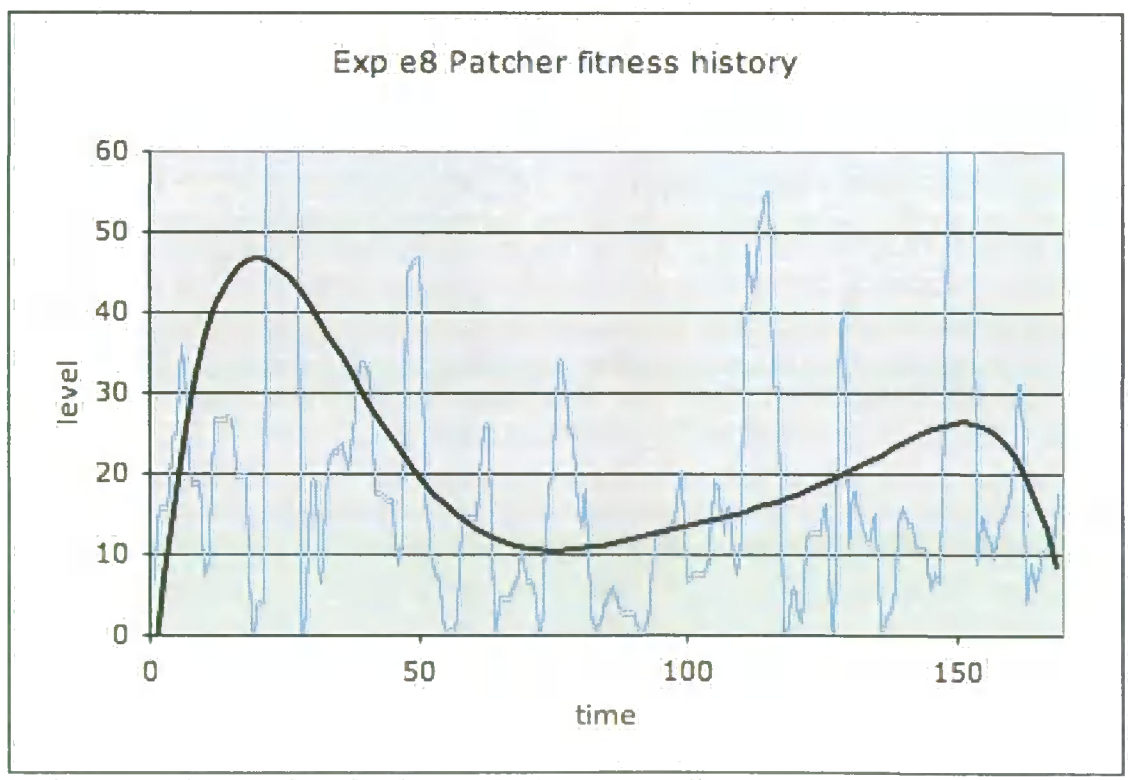


Figure 10.54: History of patcher fitness in e8.

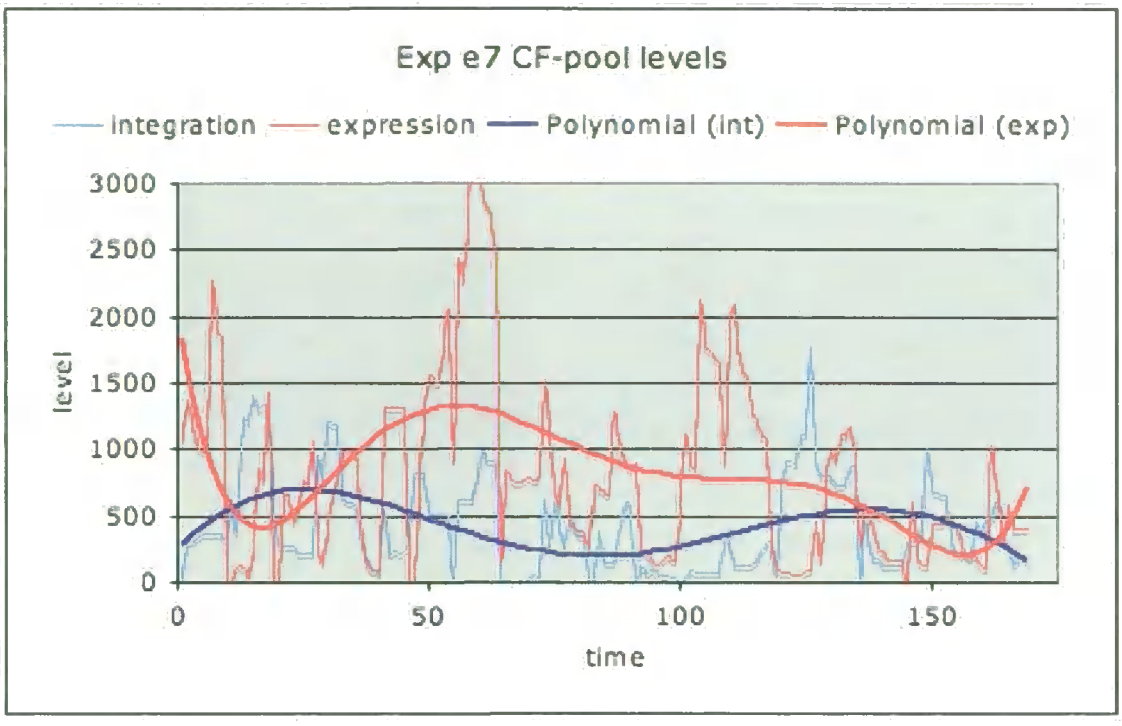


Figure 10.55: Compound-function pool fitness history in e7.

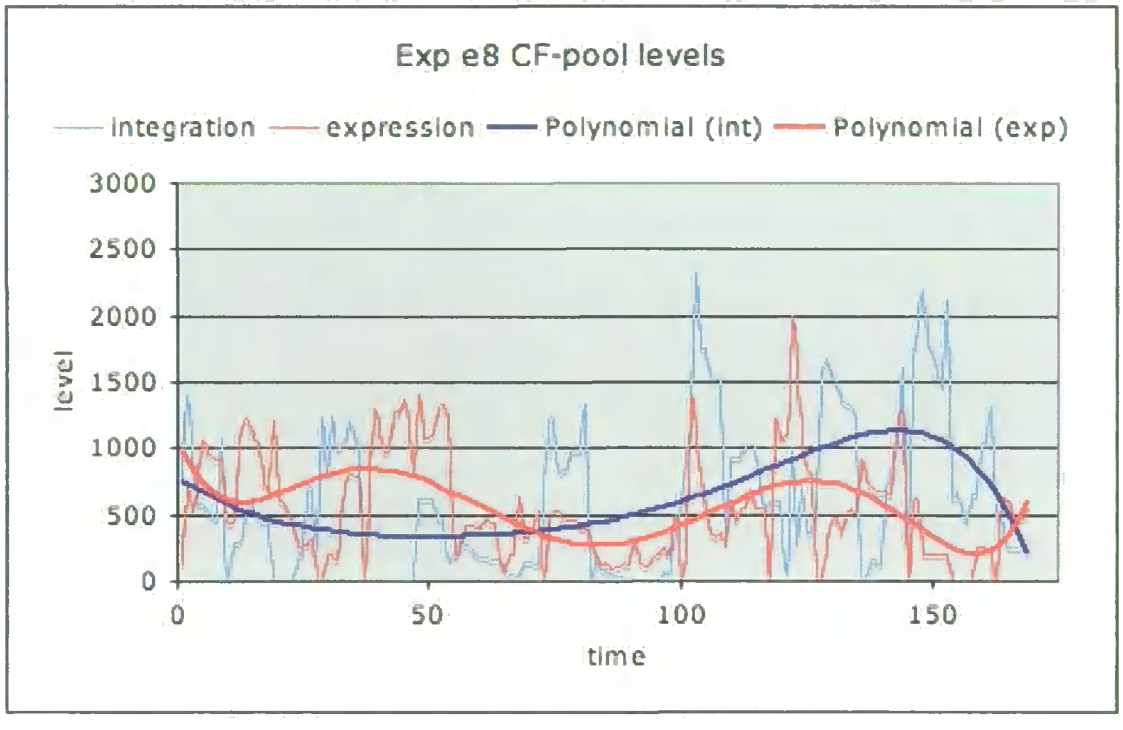


Figure 10.56: Compound-function pool fitness history in e8.

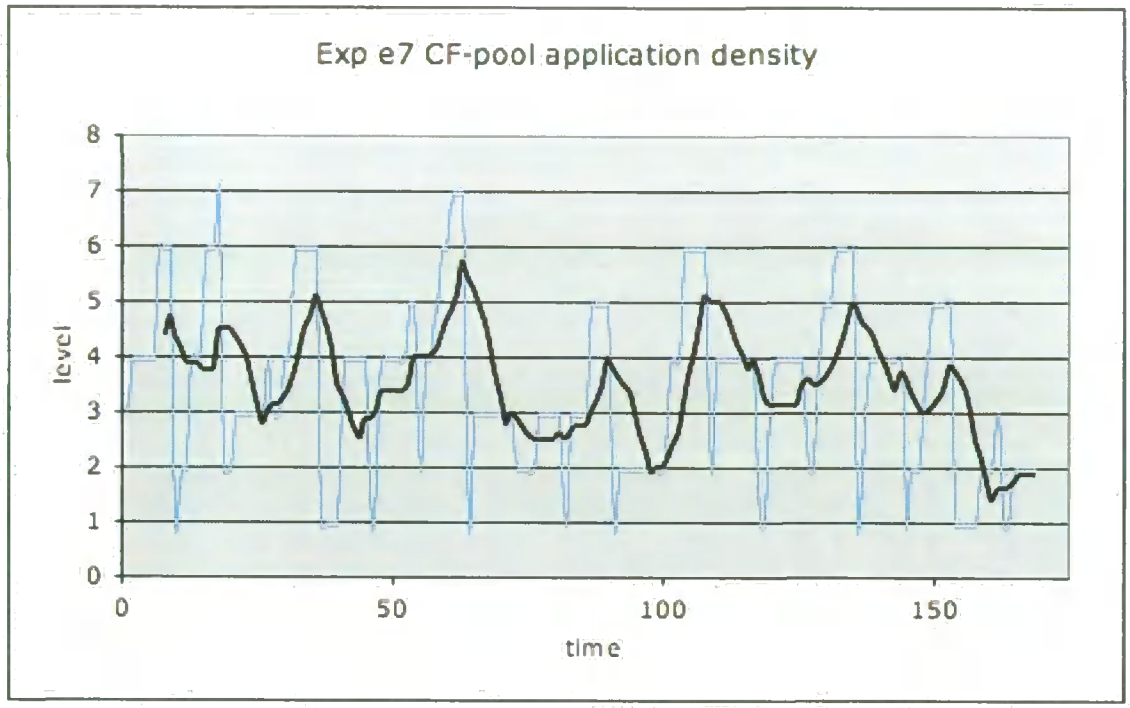


Figure 10.57: Compound-function pool application density in e7.

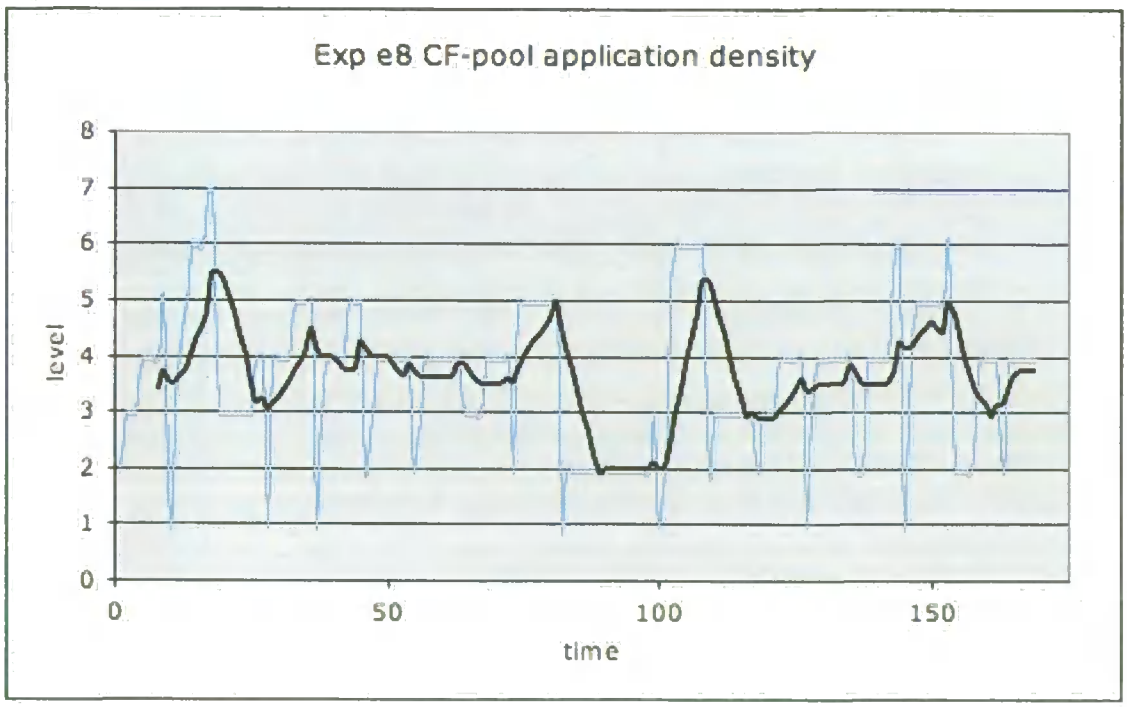


Figure 10.58: Compound-function pool application density in e8.

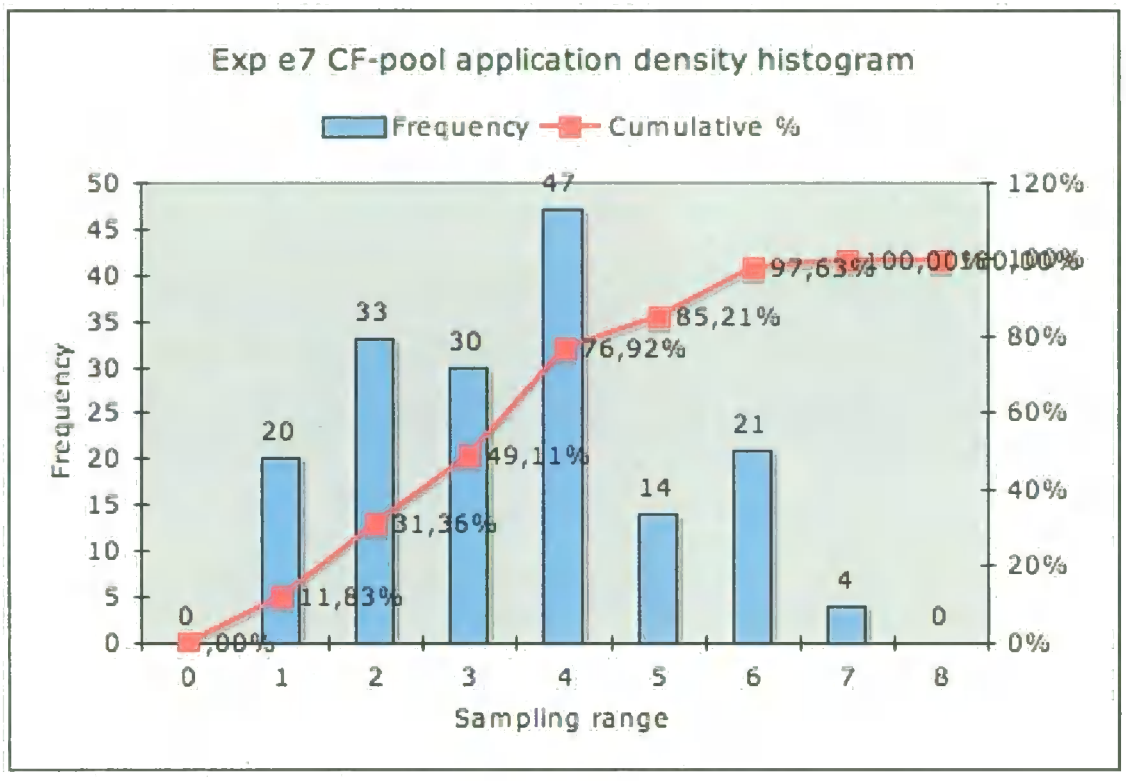


Figure 10.59: Compound-function pool application density histogram in e7.

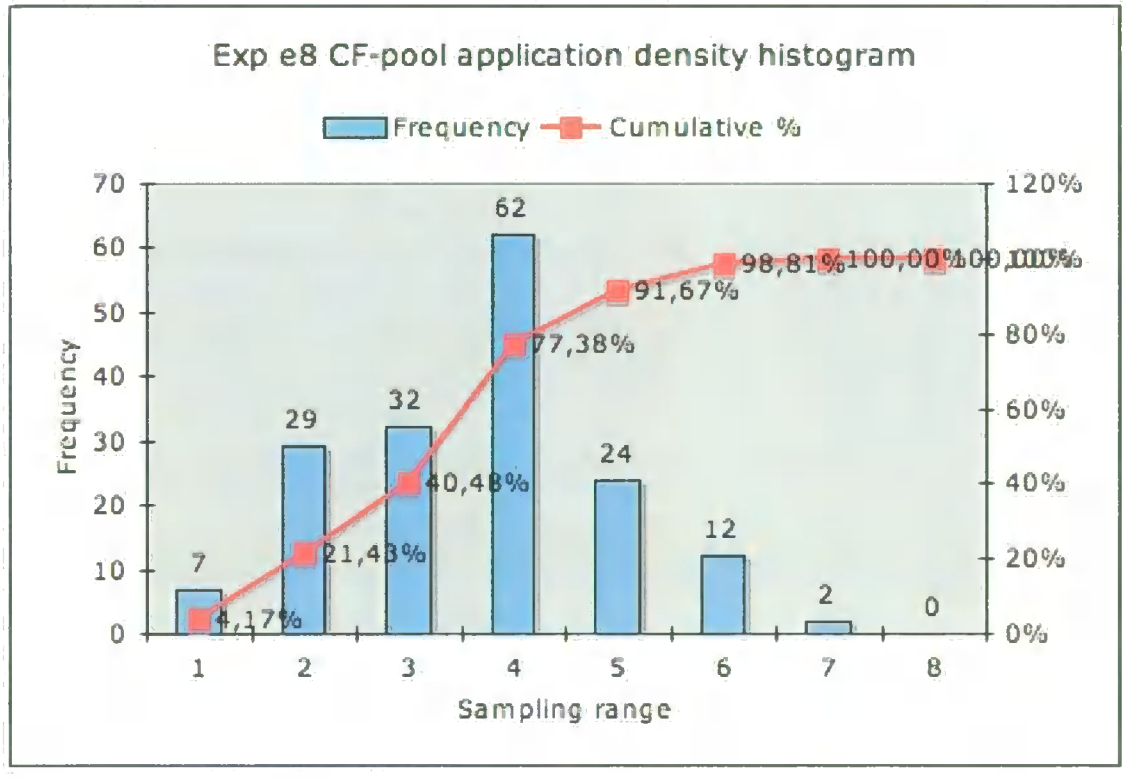


Figure 10.60: Compound-function pool application density histogram in e8.

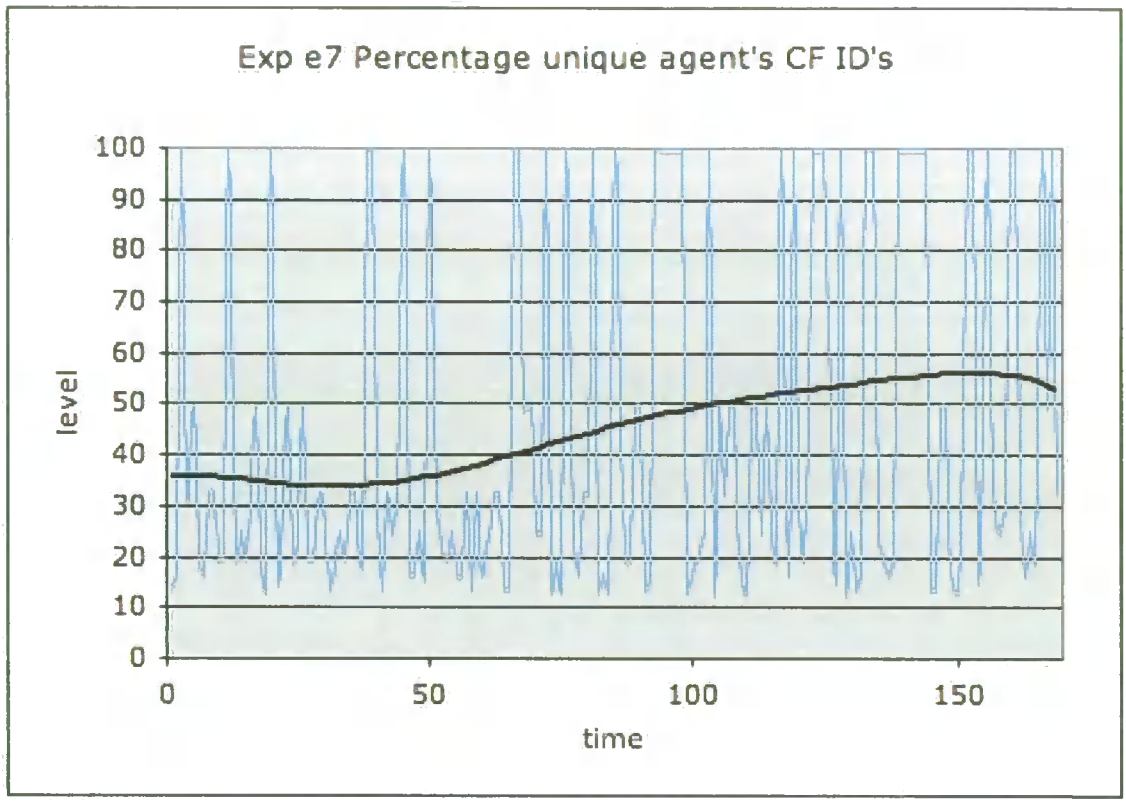


Figure 10.61: Percentage unique agent's CF ID's in e7.

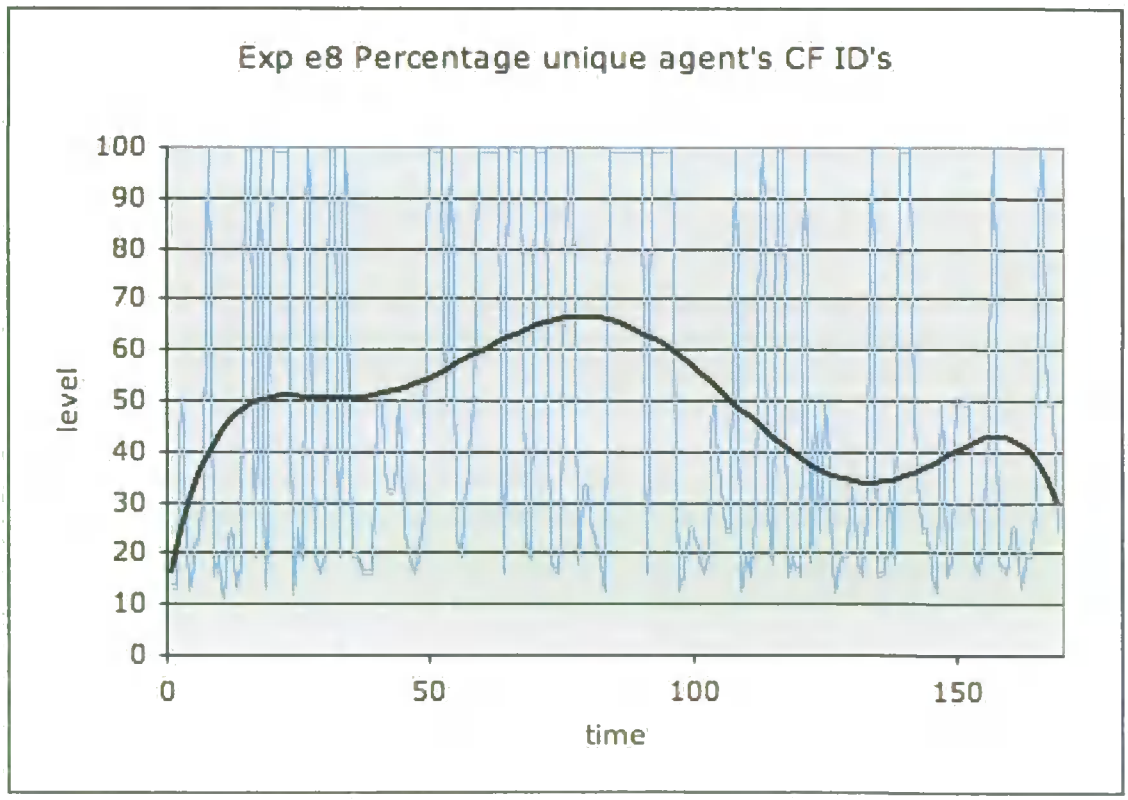


Figure 10.62: Percentage unique agent's CF ID's in e8.

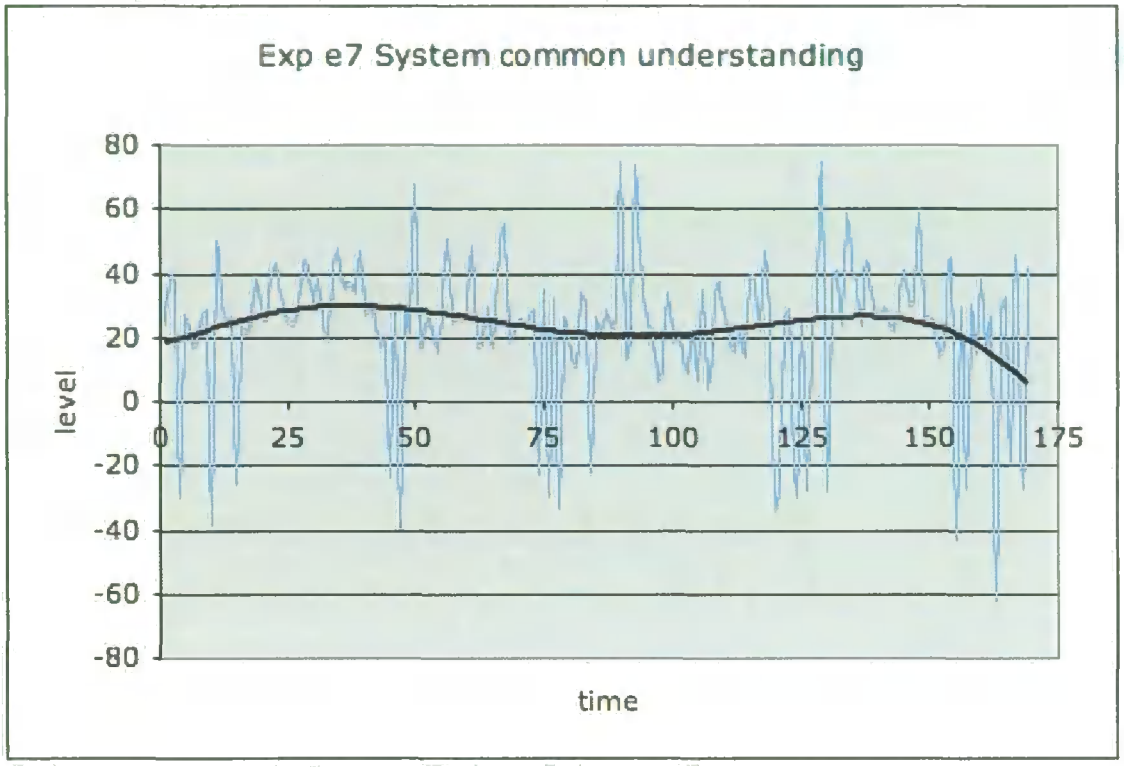


Figure 10.63: System common understanding in e7.

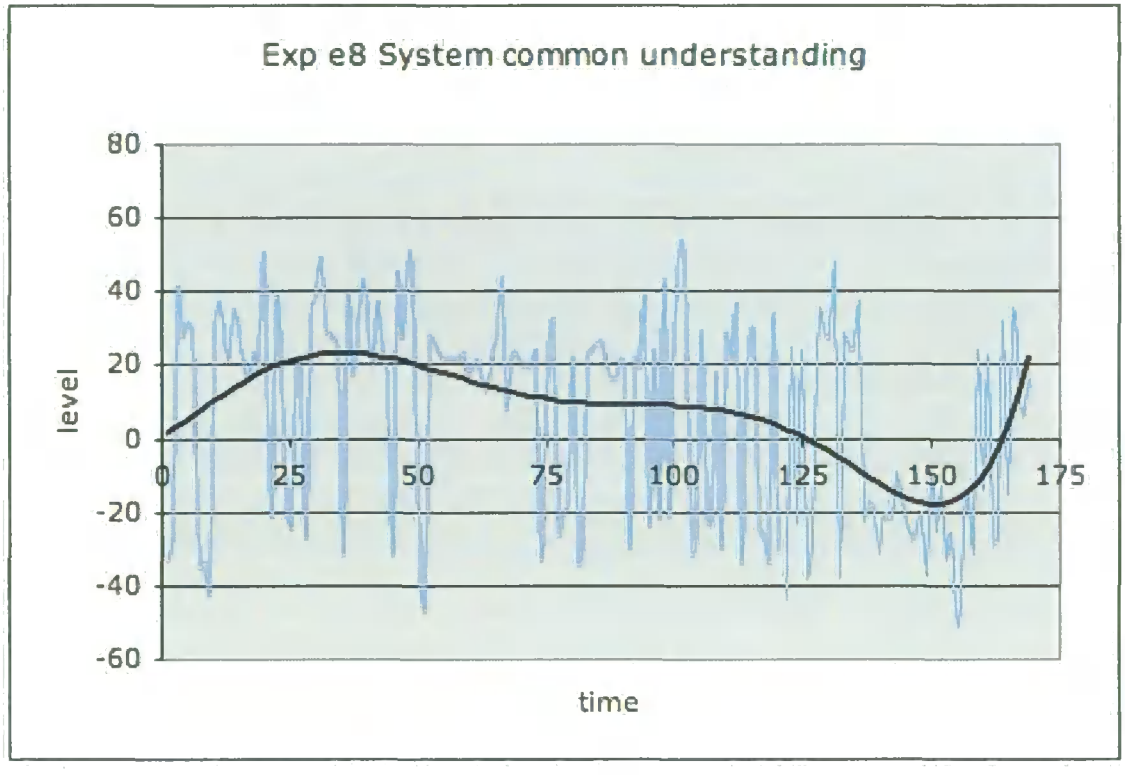


Figure 10.64: System common understanding in e8.

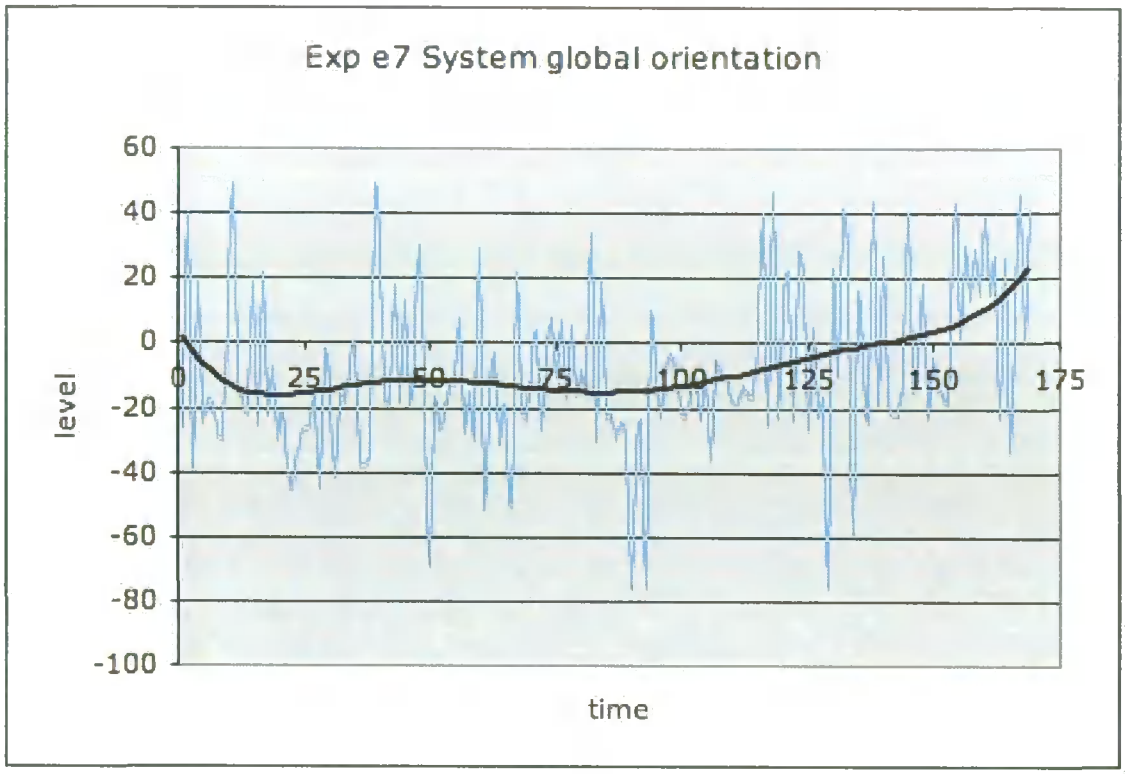


Figure 10.65: System global orientation in e7.

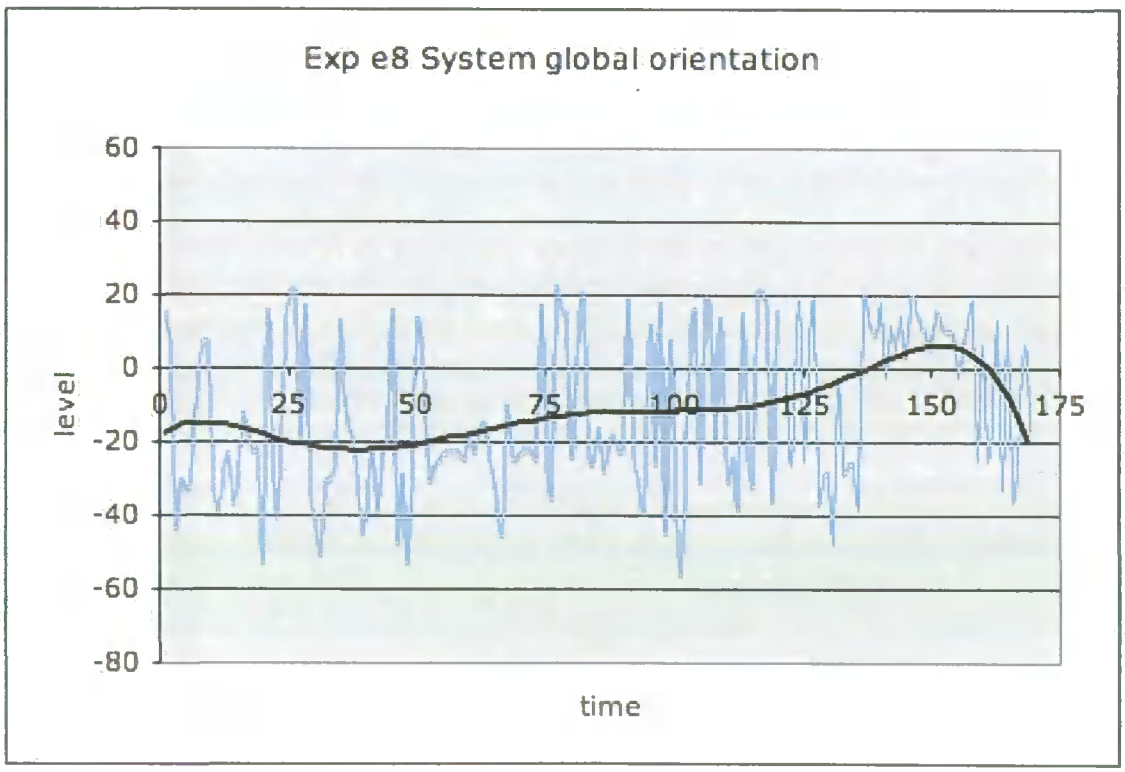


Figure 10.66: System global orientation in e8.

10.4.4 Experiments e9 and e10: Visualisation of Tracer Data

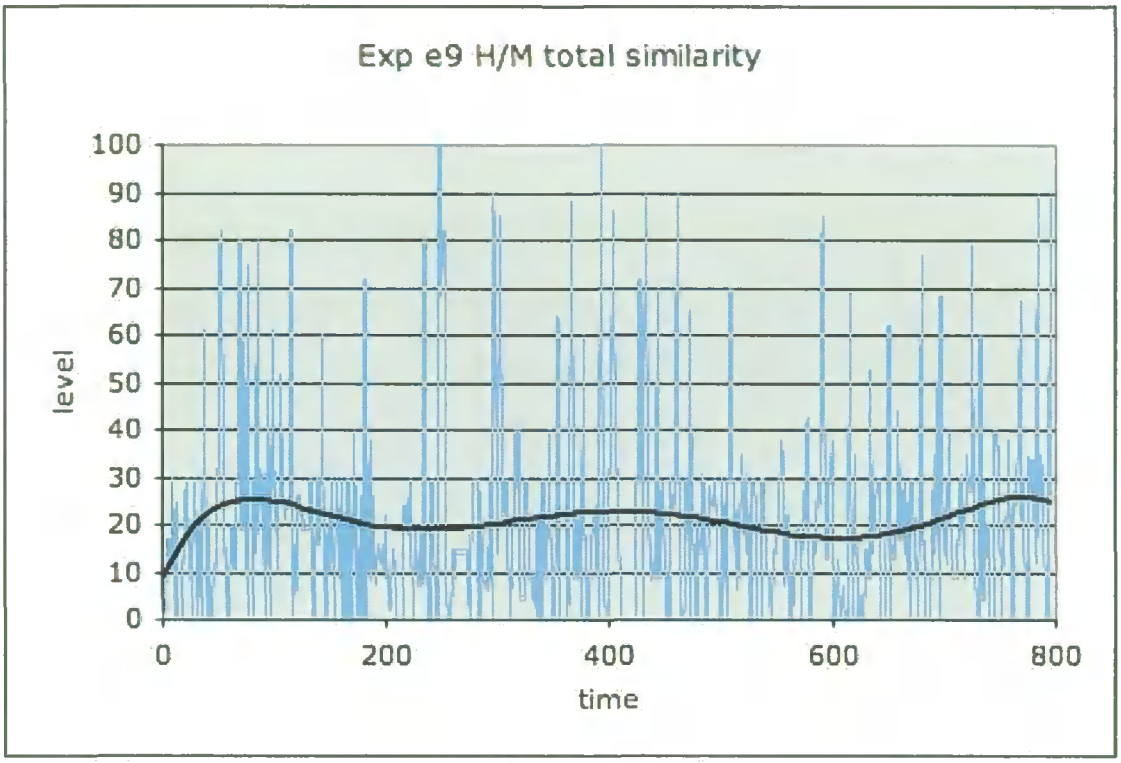


Figure 10.67: History of human-machine total similarity in e9

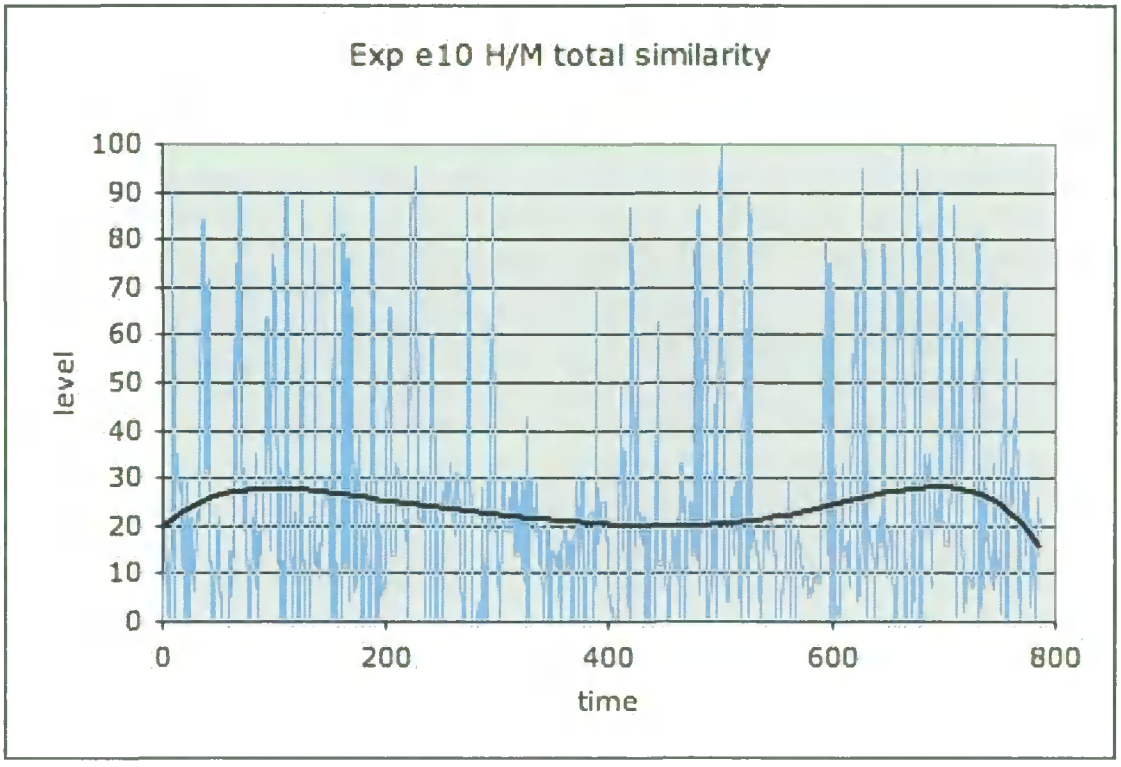


Figure 10.68: History of human-machine total similarity in e10.

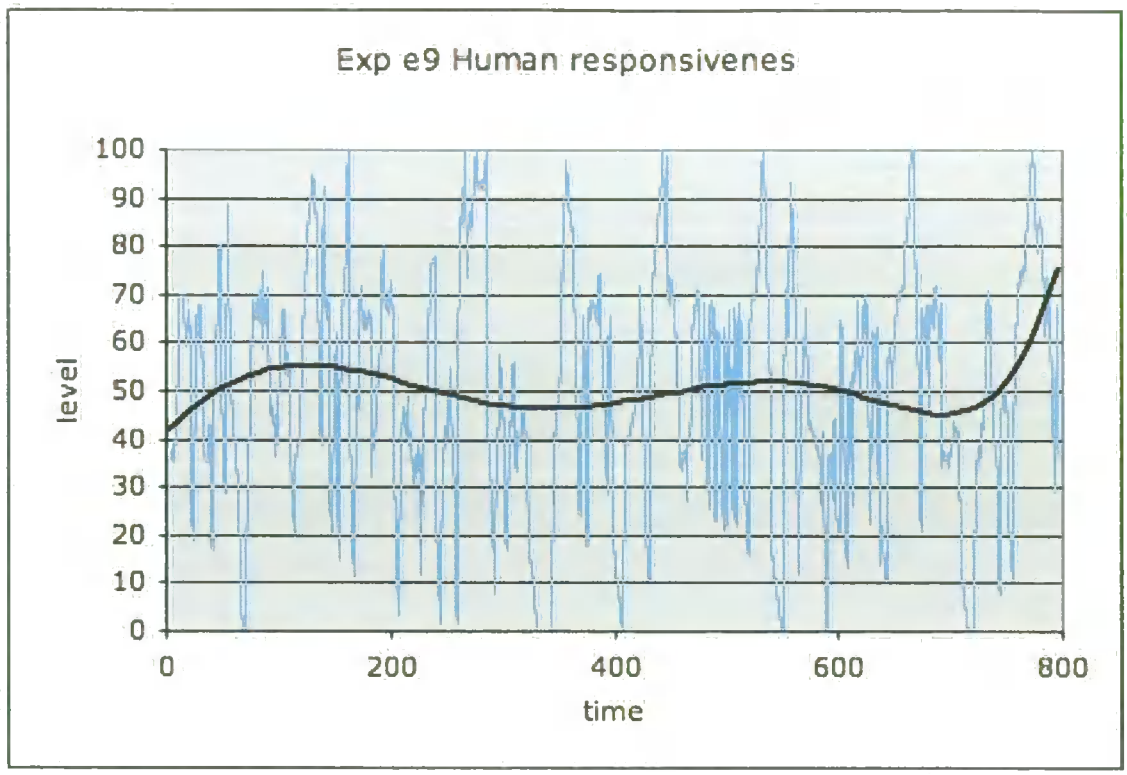


Figure 10.69: History of human responsiveness in e9.

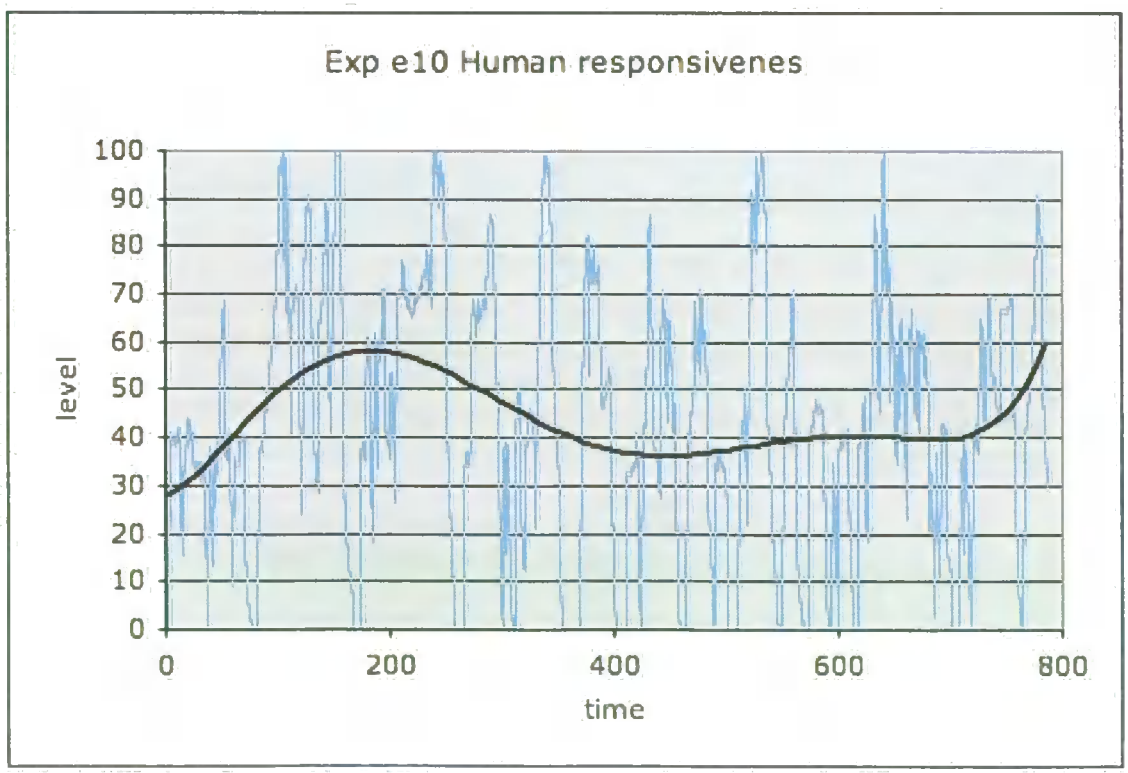


Figure 10.70: History of human responsiveness in e10.

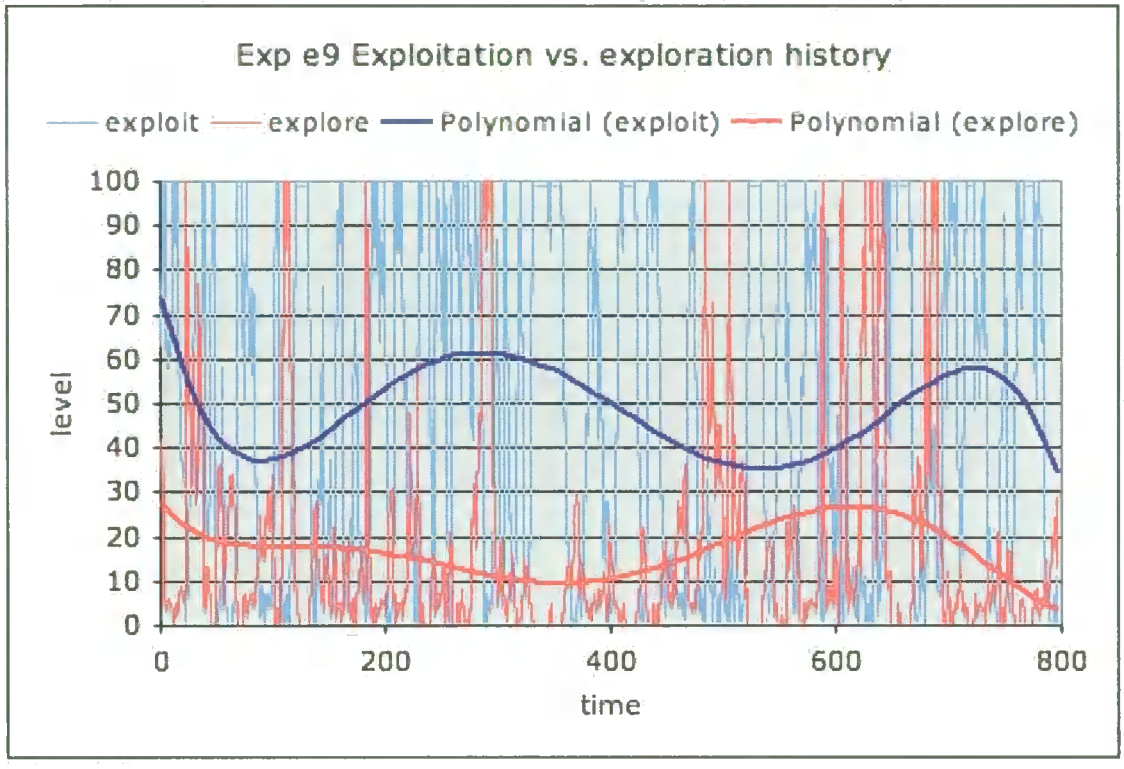


Figure 10.71: Exploitation vs. exploration pressures in e9.

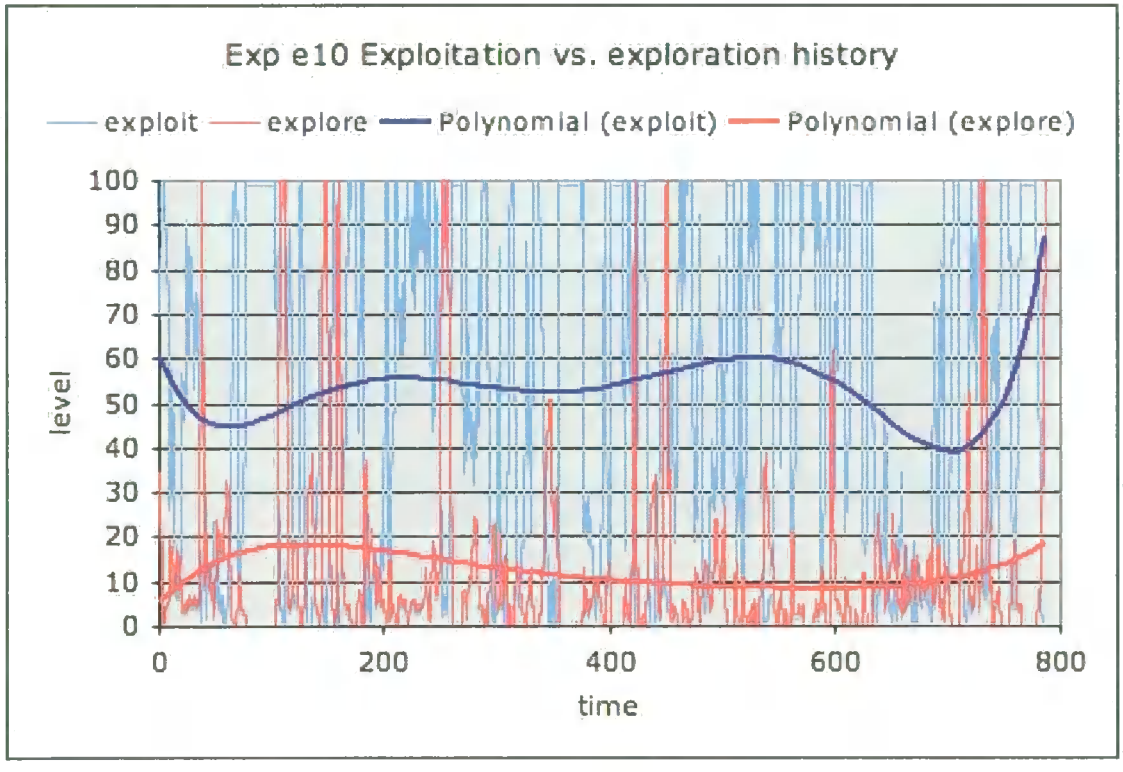


Figure 10.72: Exploitation vs. exploration pressures in e10.

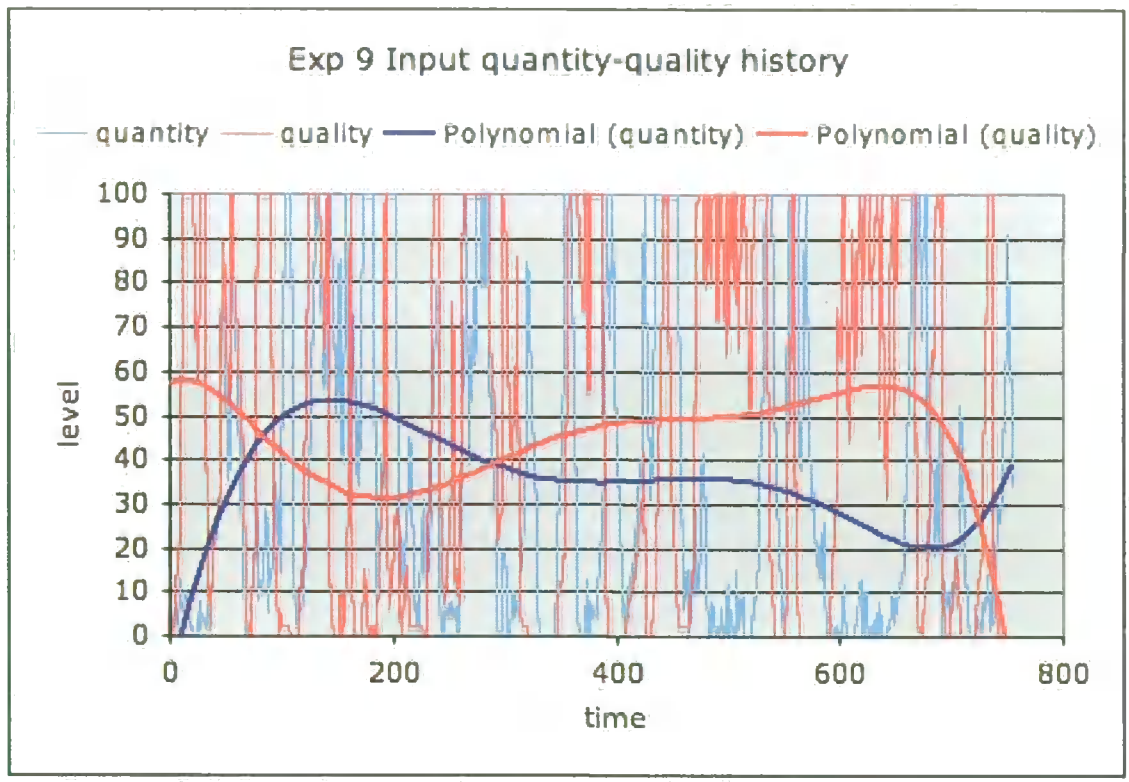


Figure 10.73: Quantity vs. quality of human input in e9.

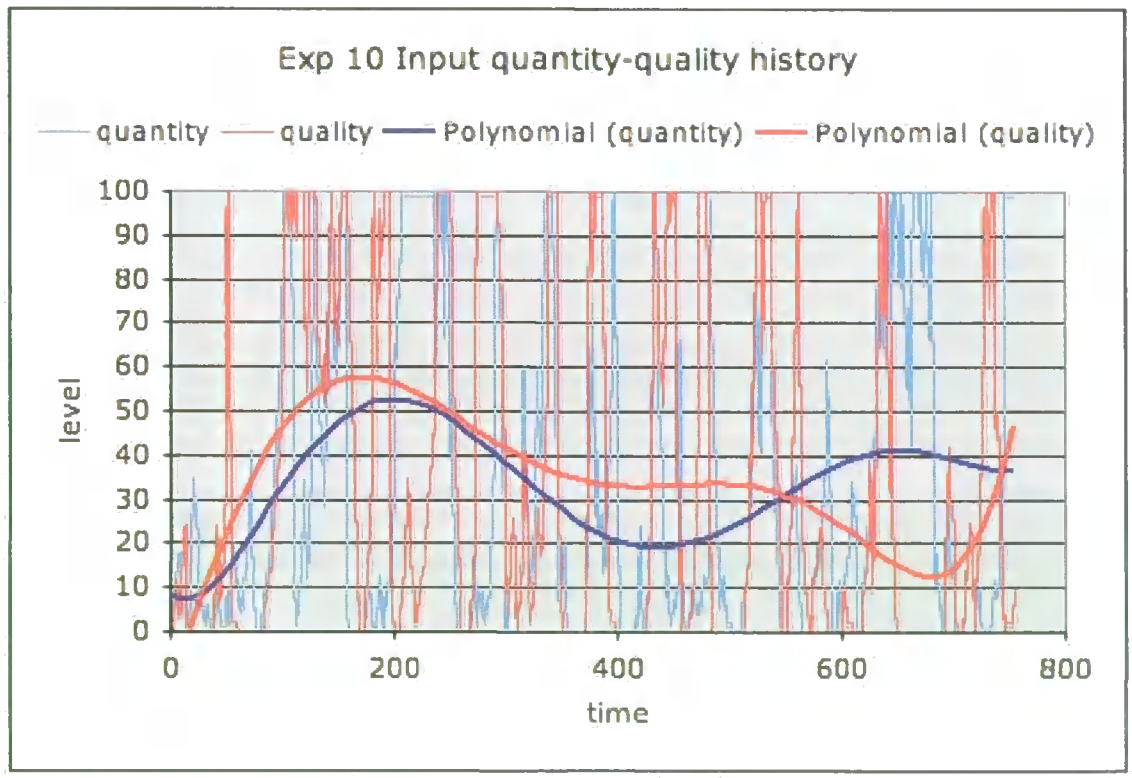


Figure 10.74: Quantity vs. quality of human input in e10.

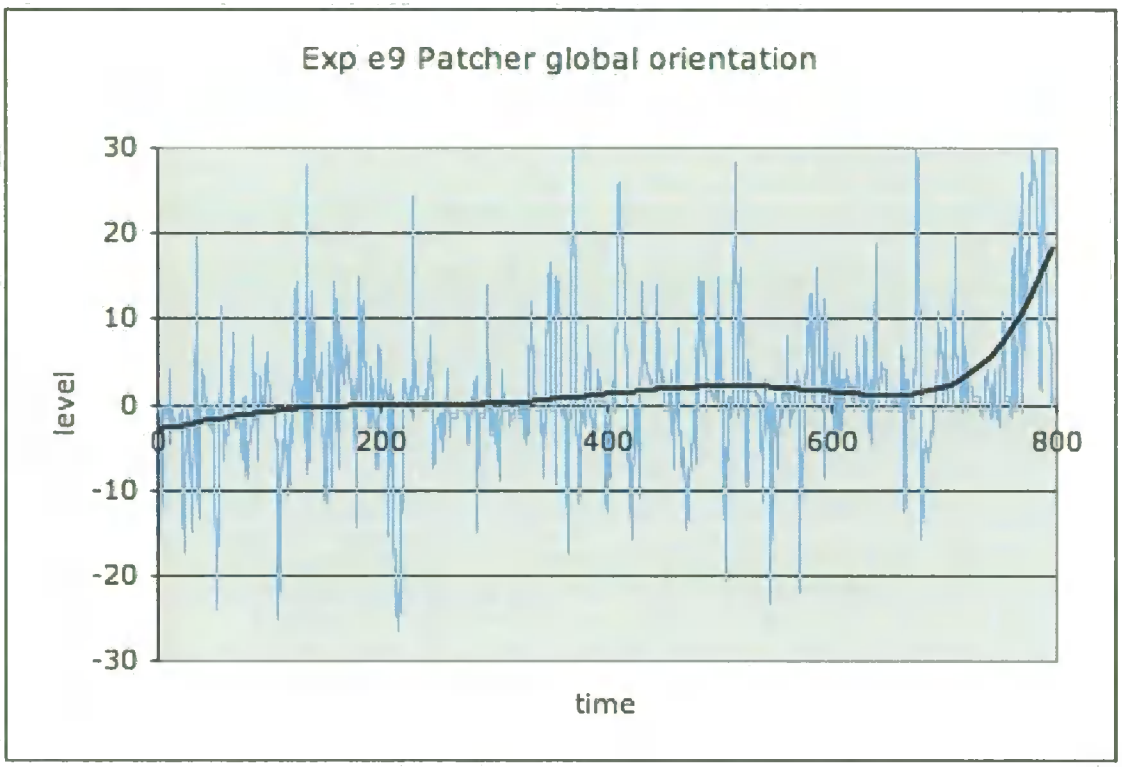


Figure 10.75: Patcher global orientation in e9.

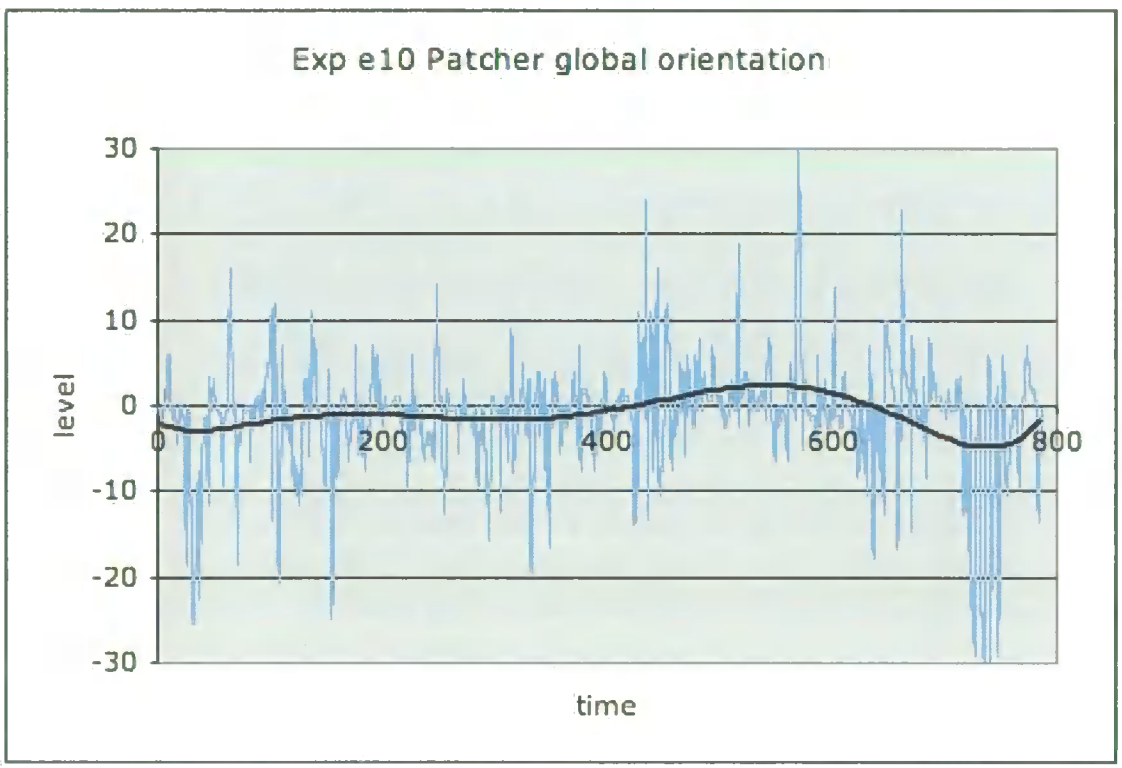


Figure 10.76: Patcher global orientation in e10.

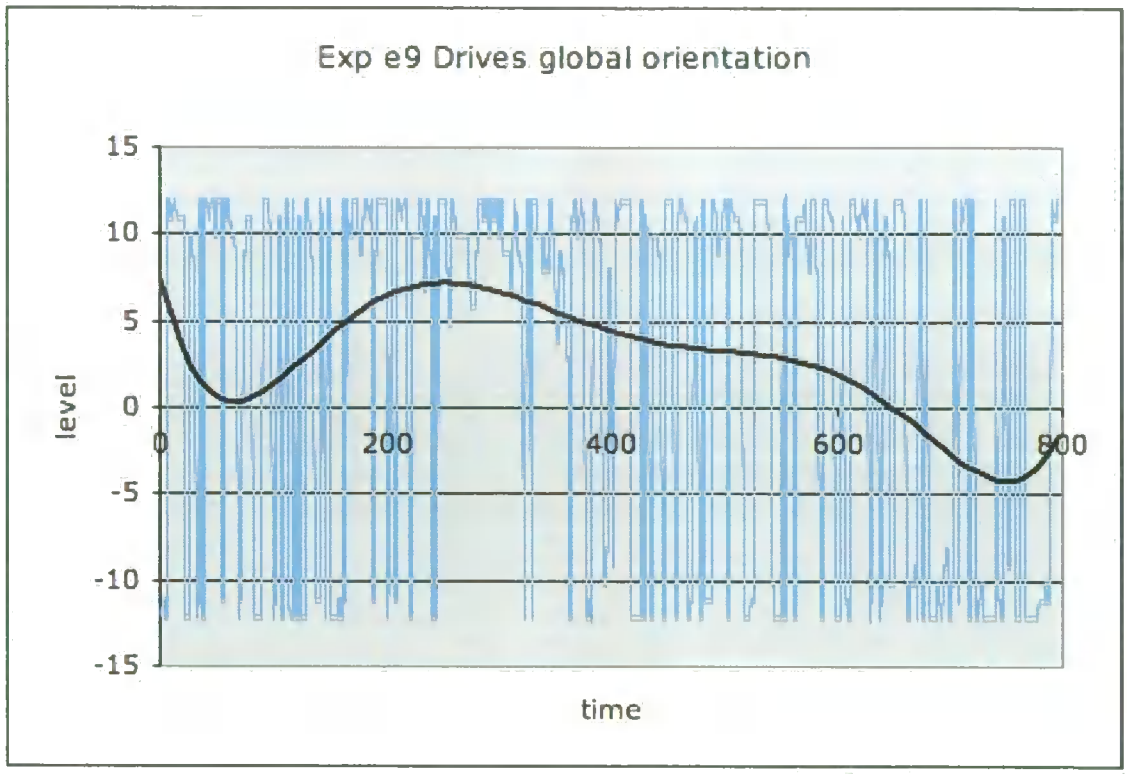


Figure 10.77: Drives-pool global orientation in e9.

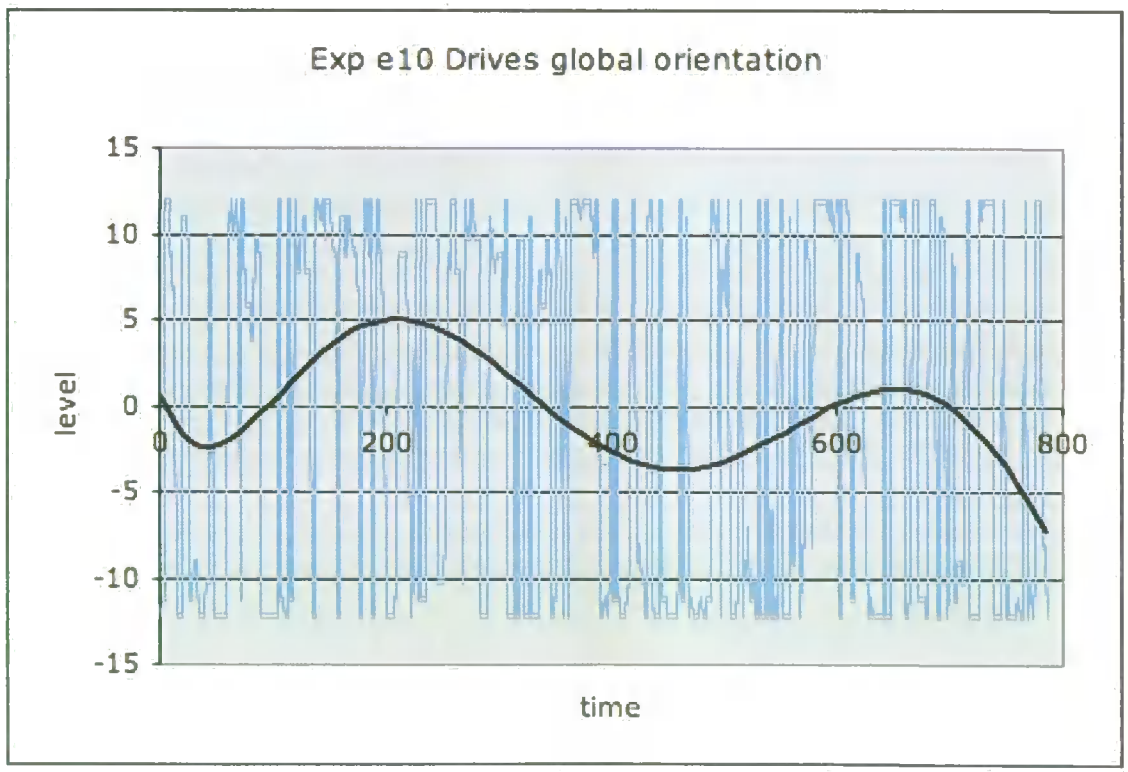


Figure 10.78: Drives-pool global orientation in e10.

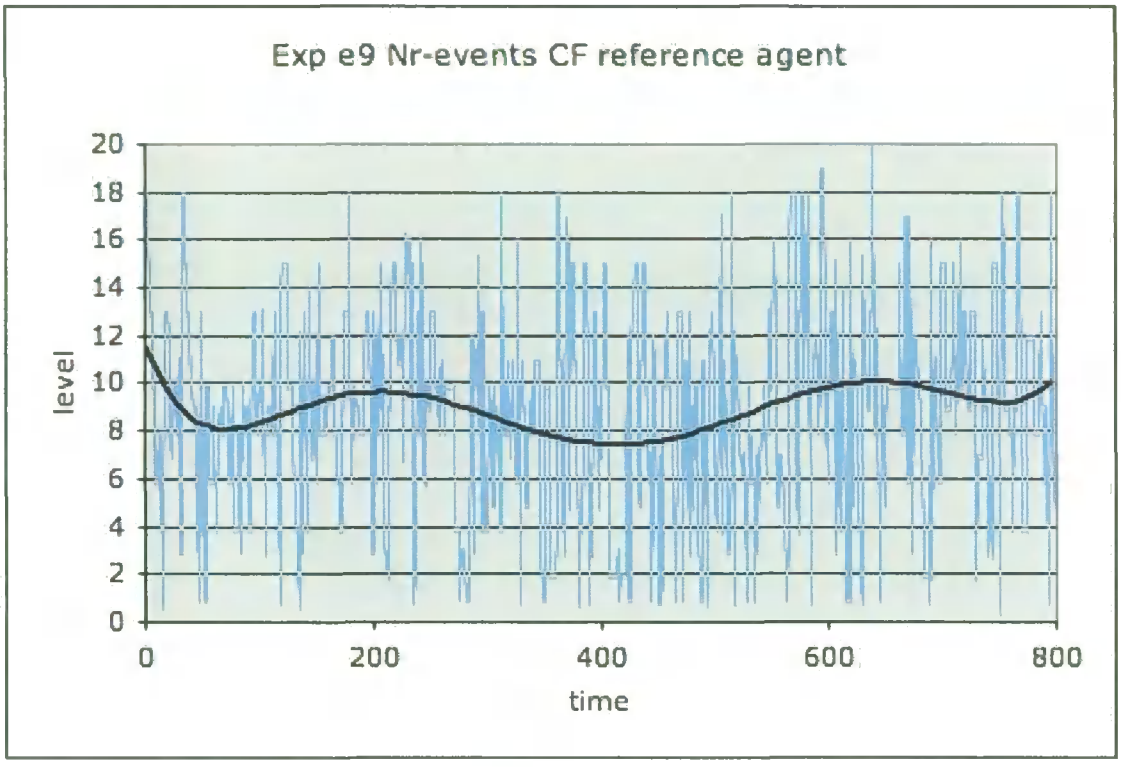


Figure 10.79: Number of events in the compound-function held by the reference agent in e9.

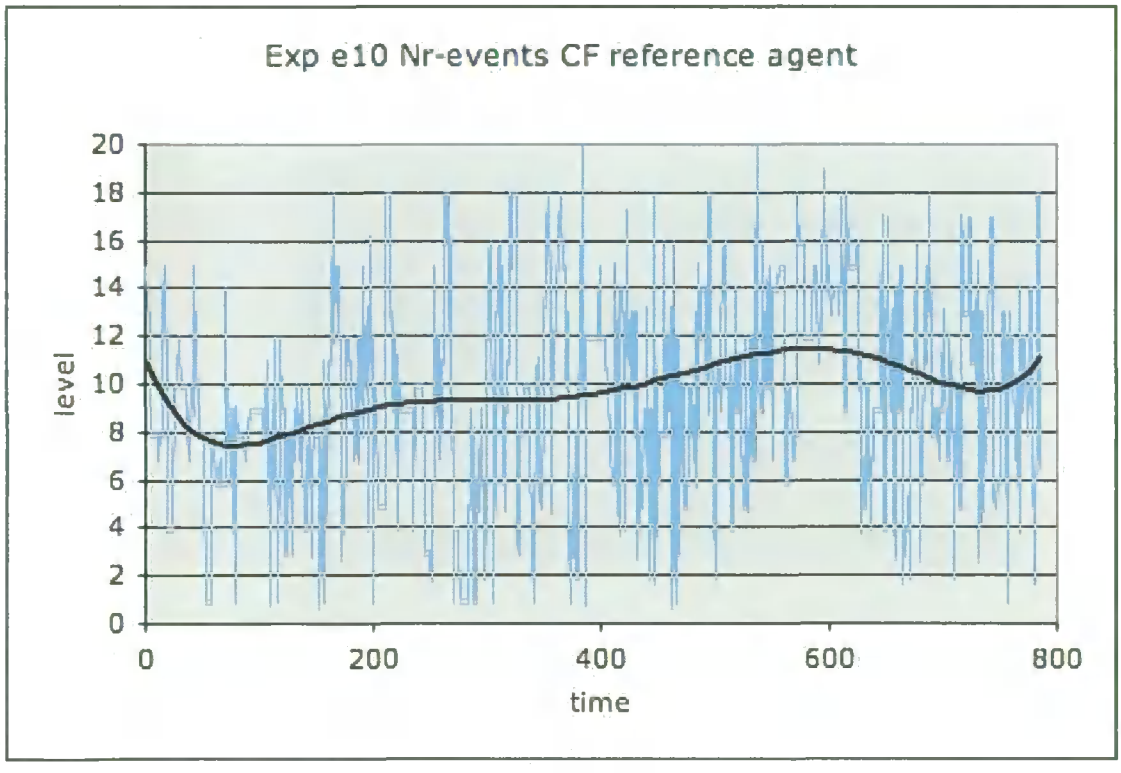


Figure 10.80: Number of events in the compound-function held by the reference agent in e10.

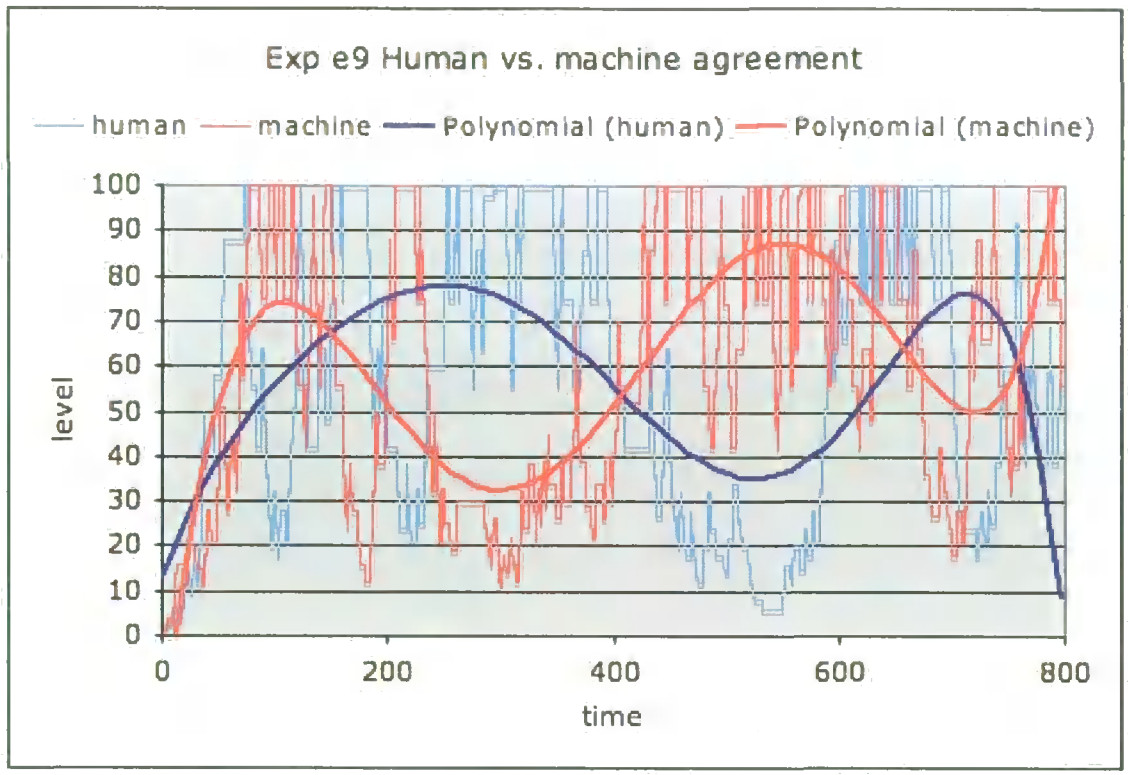


Figure 10.81: Human vs. machine agreement in e9.

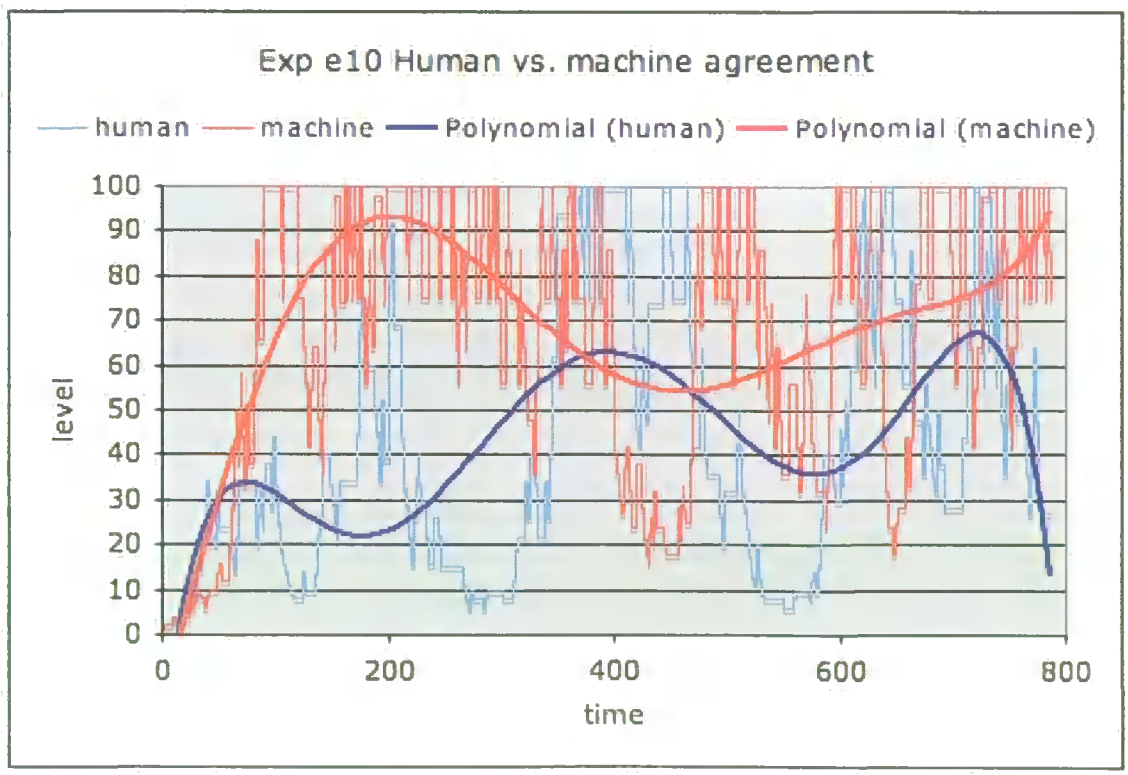


Figure 10.82: Human vs. machine agreement in e10.

10.4.5 Experiments e9 and e10: Visualisation of Evolution Data

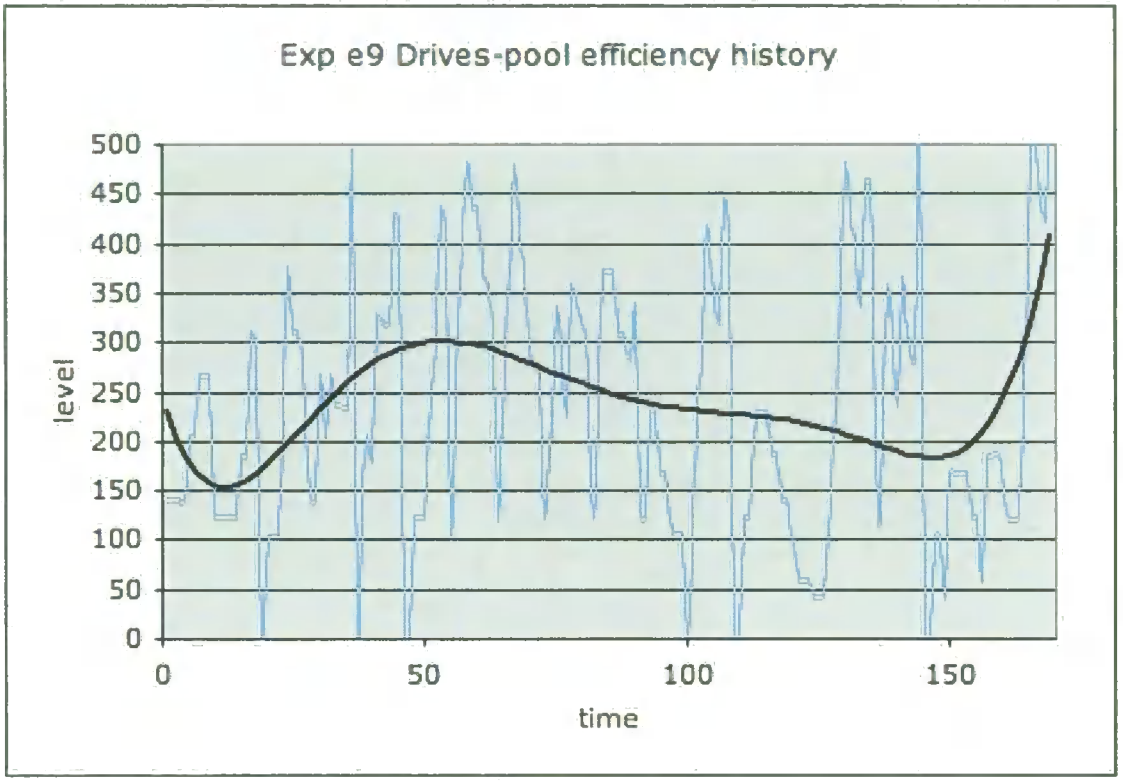


Figure 10.83: History of drives-pool efficiency in e9.

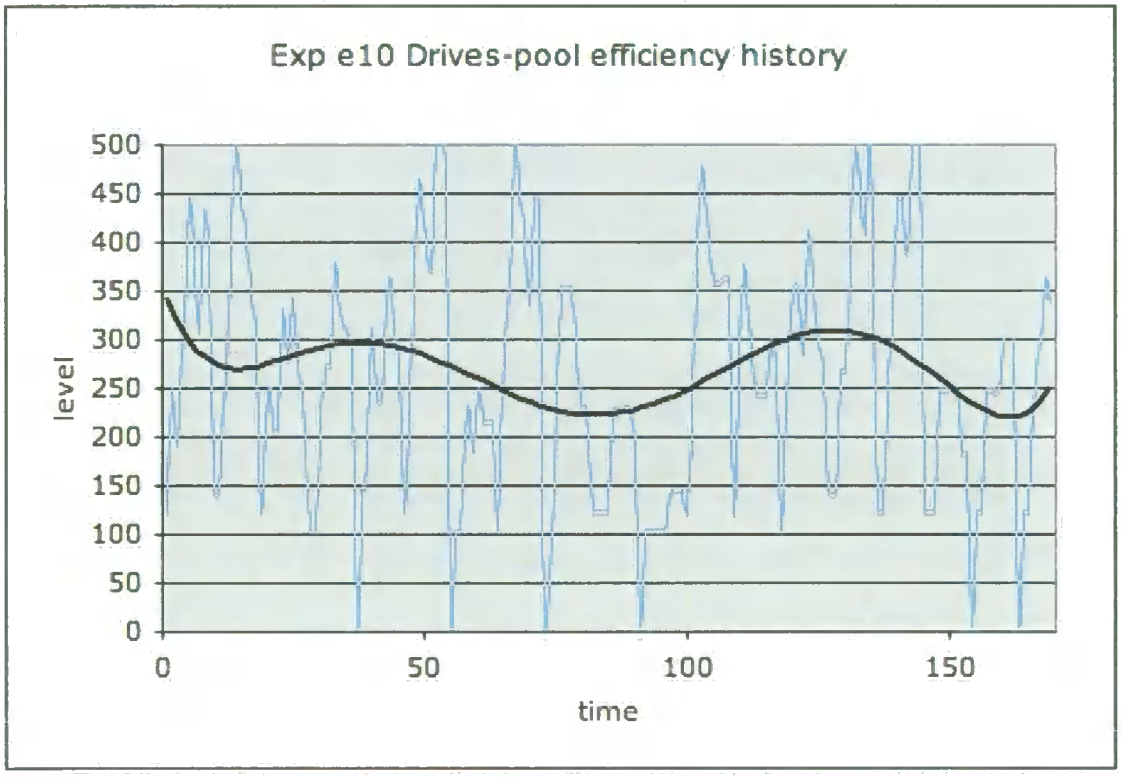


Figure 10.84: History of drives-pool efficiency in e10.

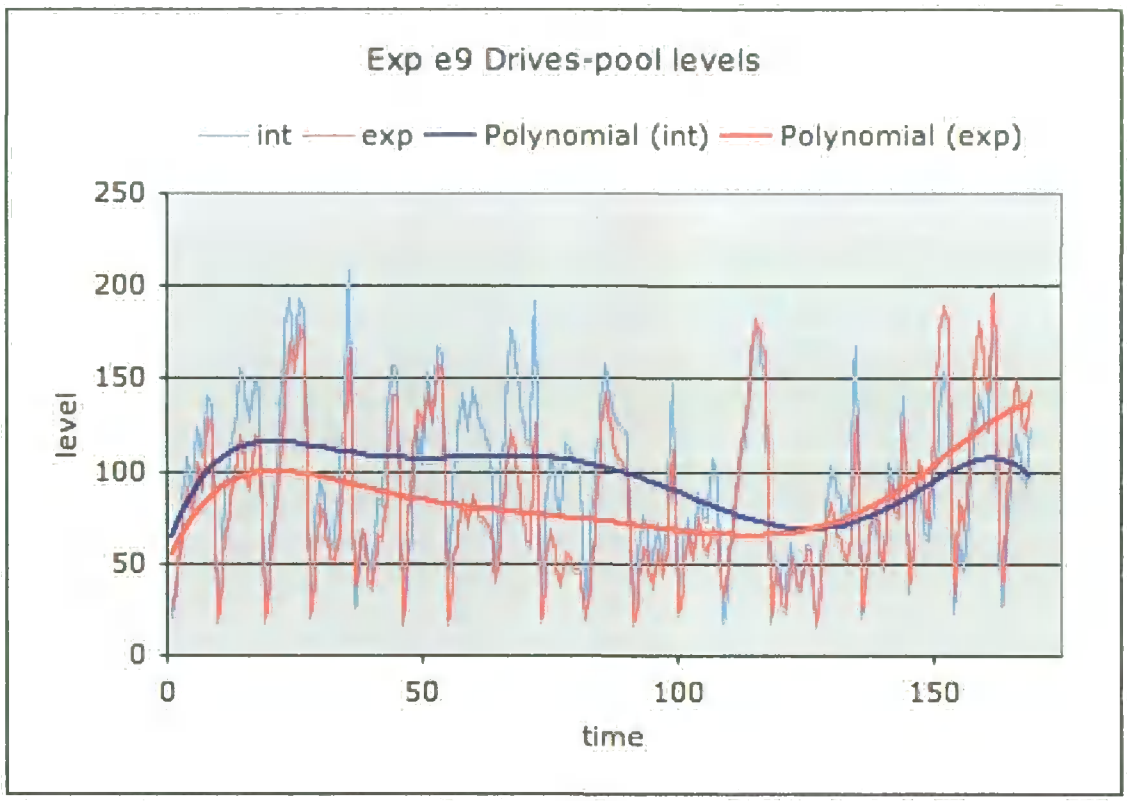


Figure 10.85: Drives-pool output levels in e9.

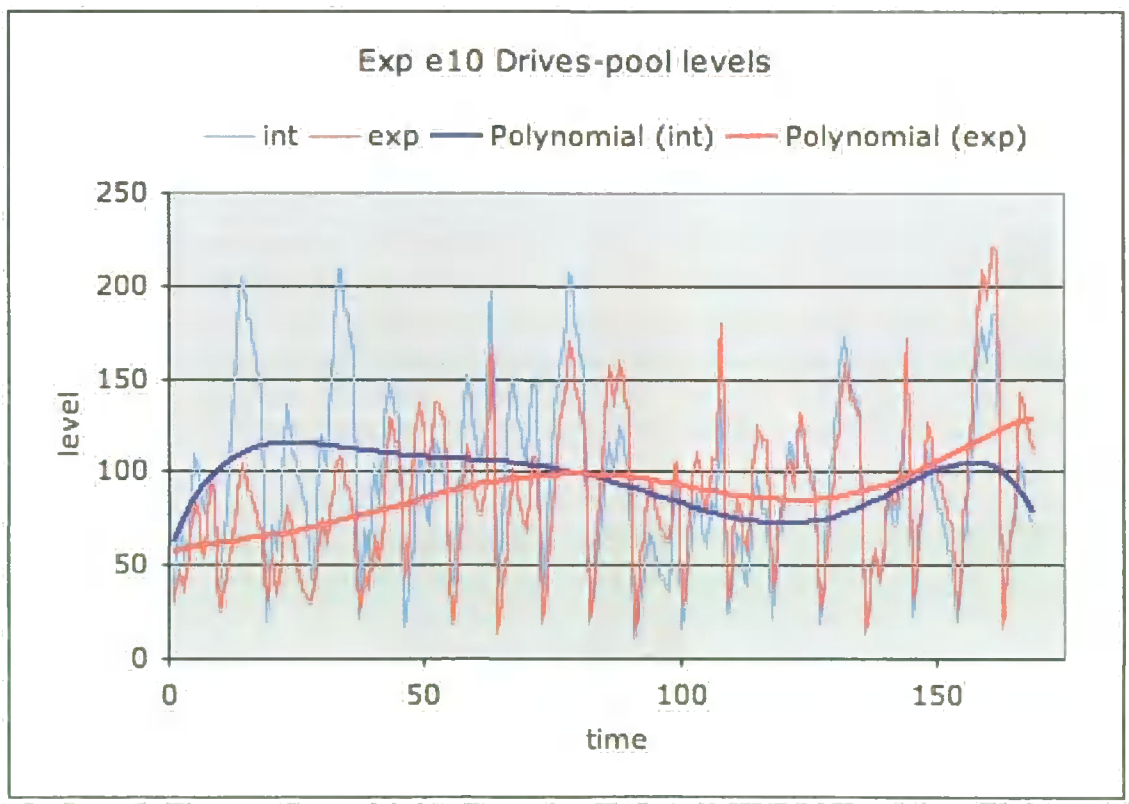


Figure 10.86: Drives-pool output levels in e10.

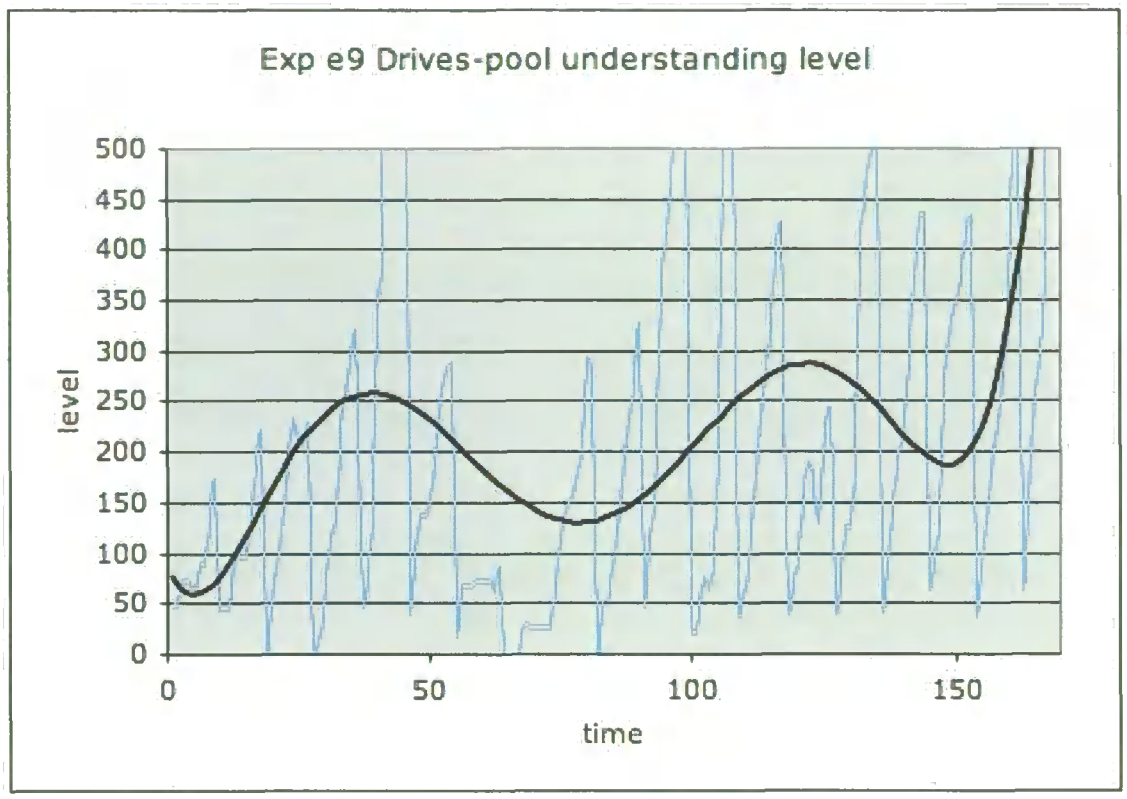


Figure 10.87: History of drives-pool understanding in e9.

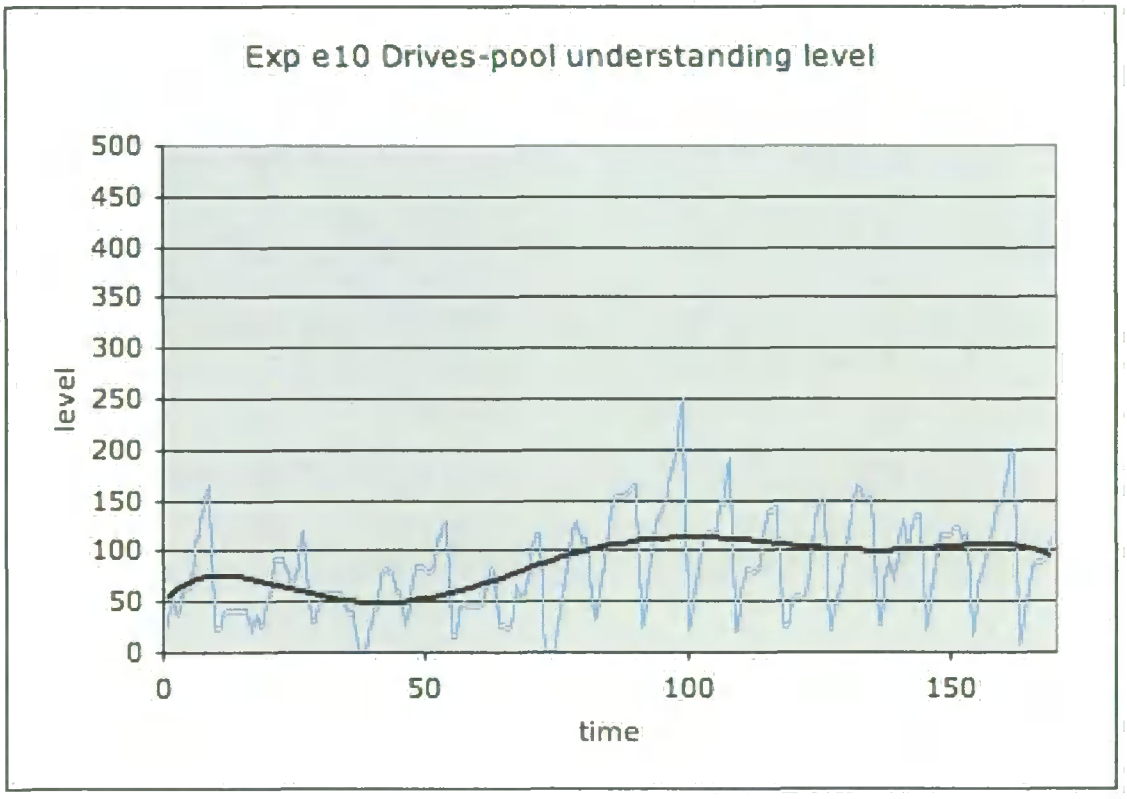


Figure 10.88: History of drives-pool understanding in e10.

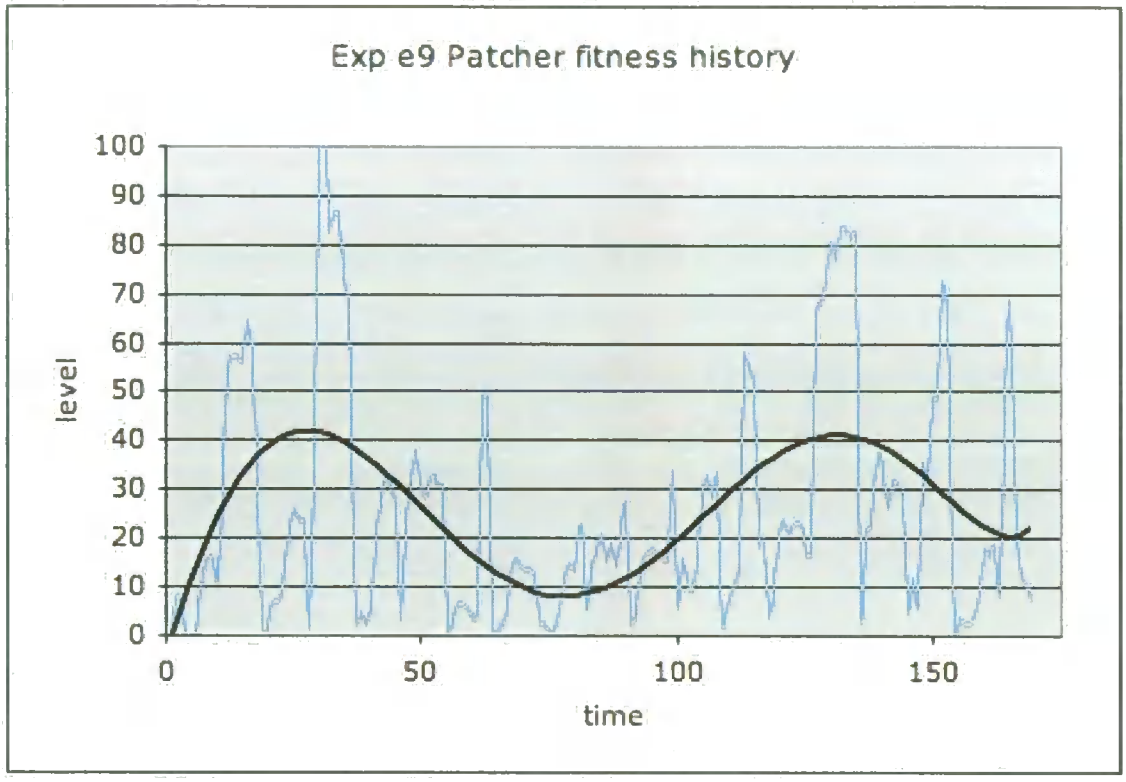


Figure 10.89: History of patcher fitness in e9.

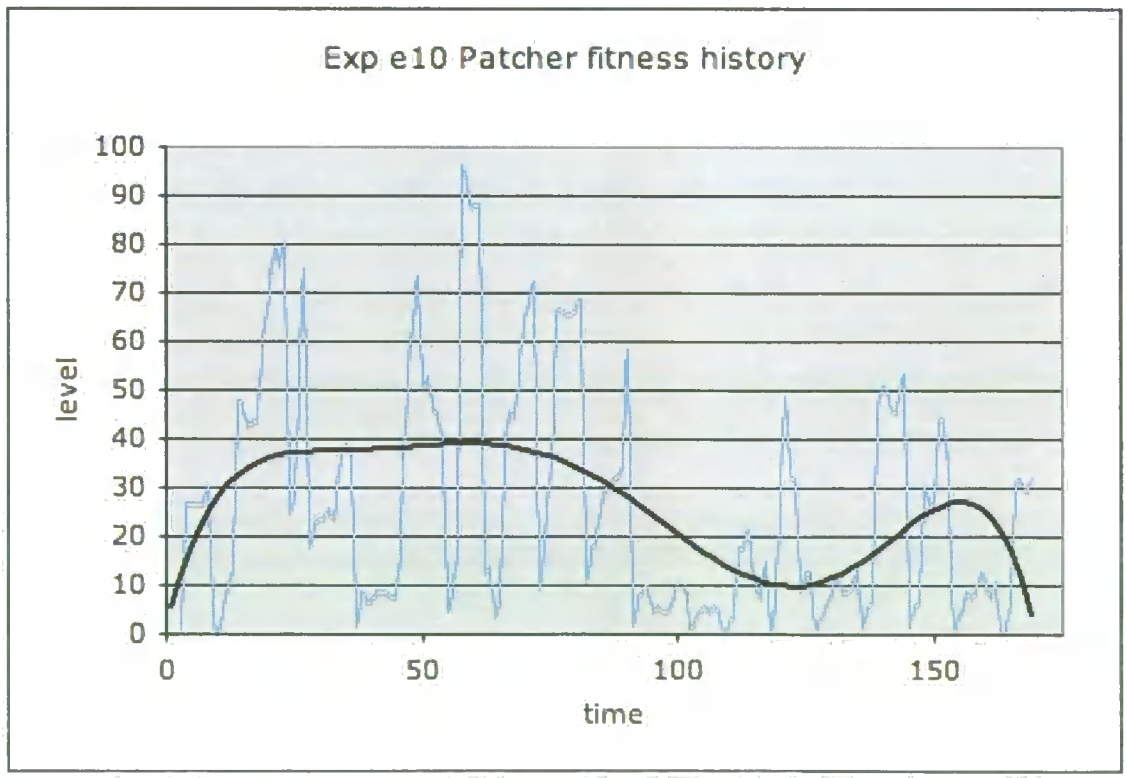


Figure 10.90: History of patcher fitness in e10.

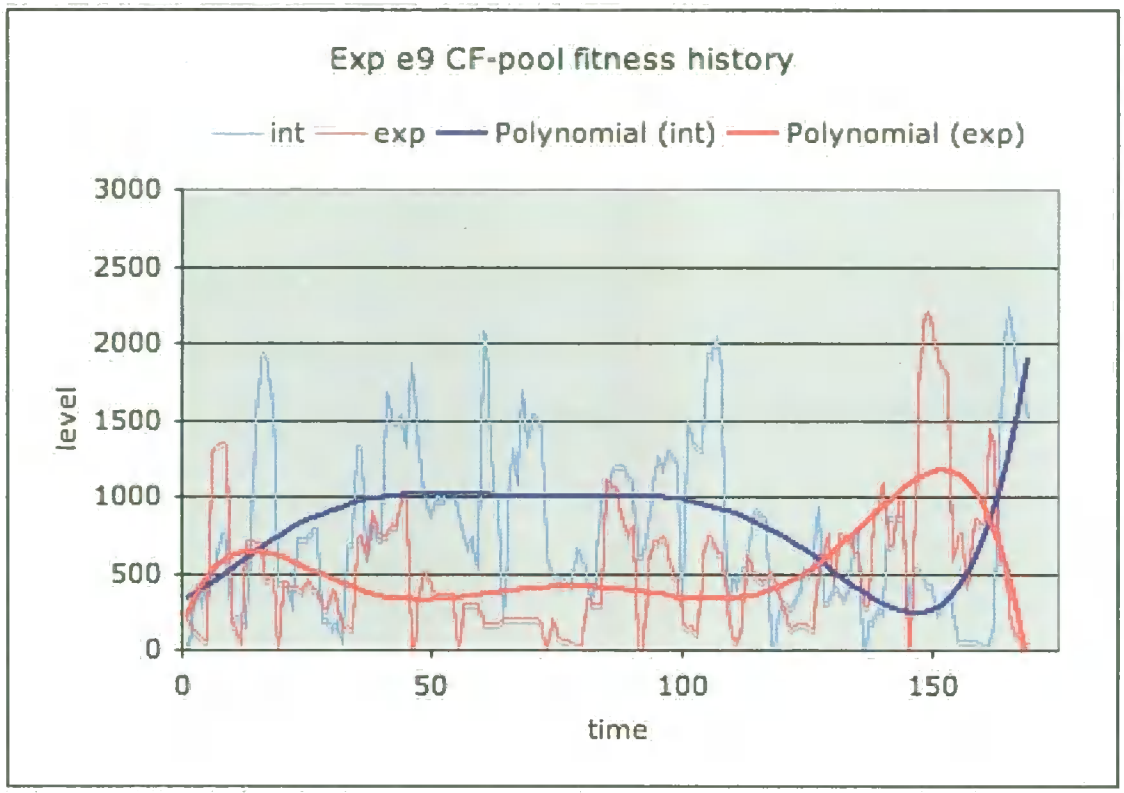


Figure 10.91: Compound-function pool fitness history in e9.

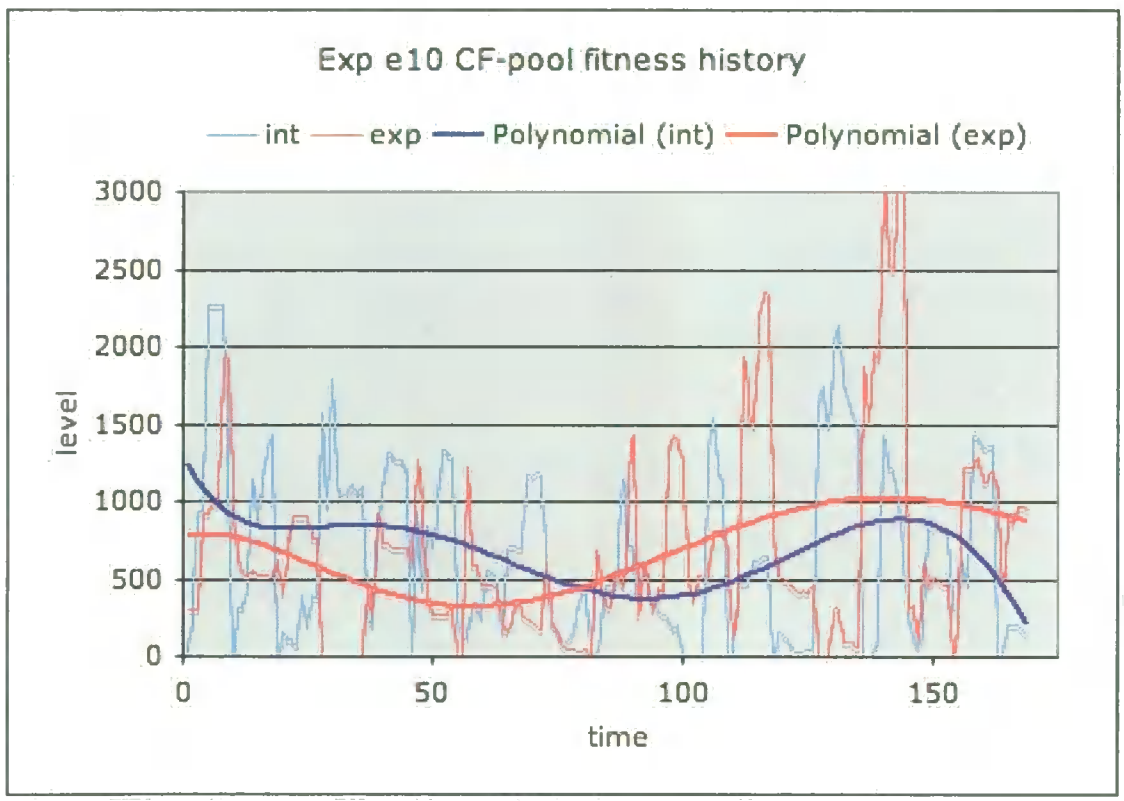


Figure 10.92: Compound-function pool fitness history in e10.

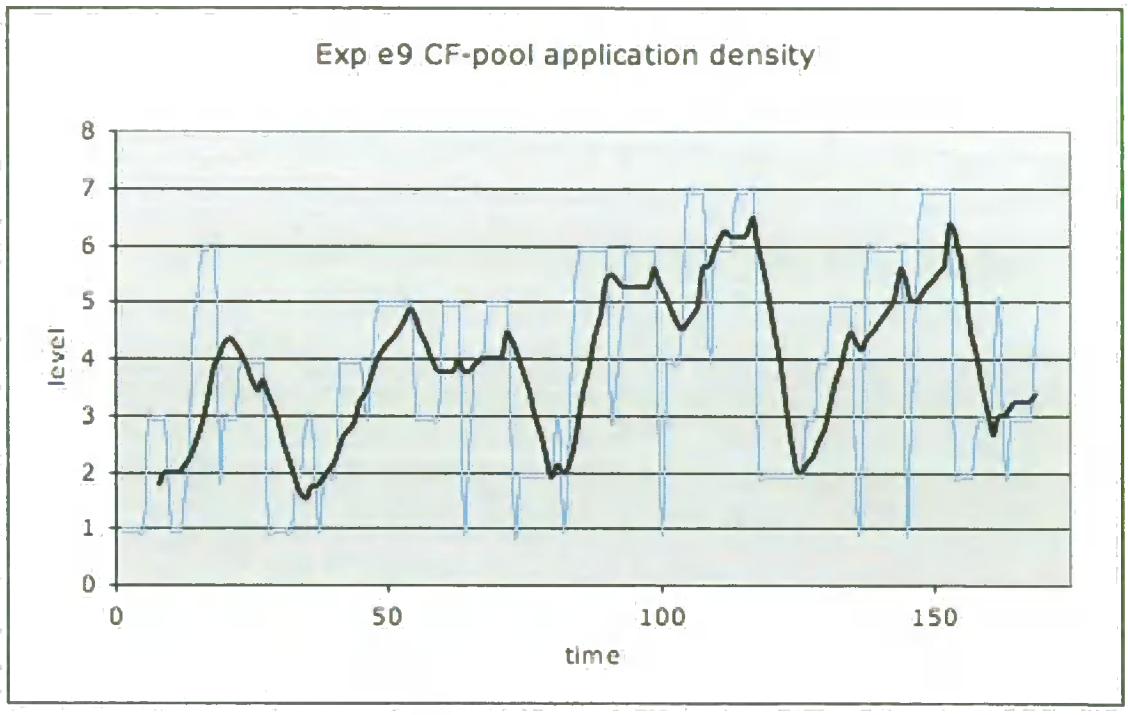


Figure 10.93: Compound-function pool application density in e9.

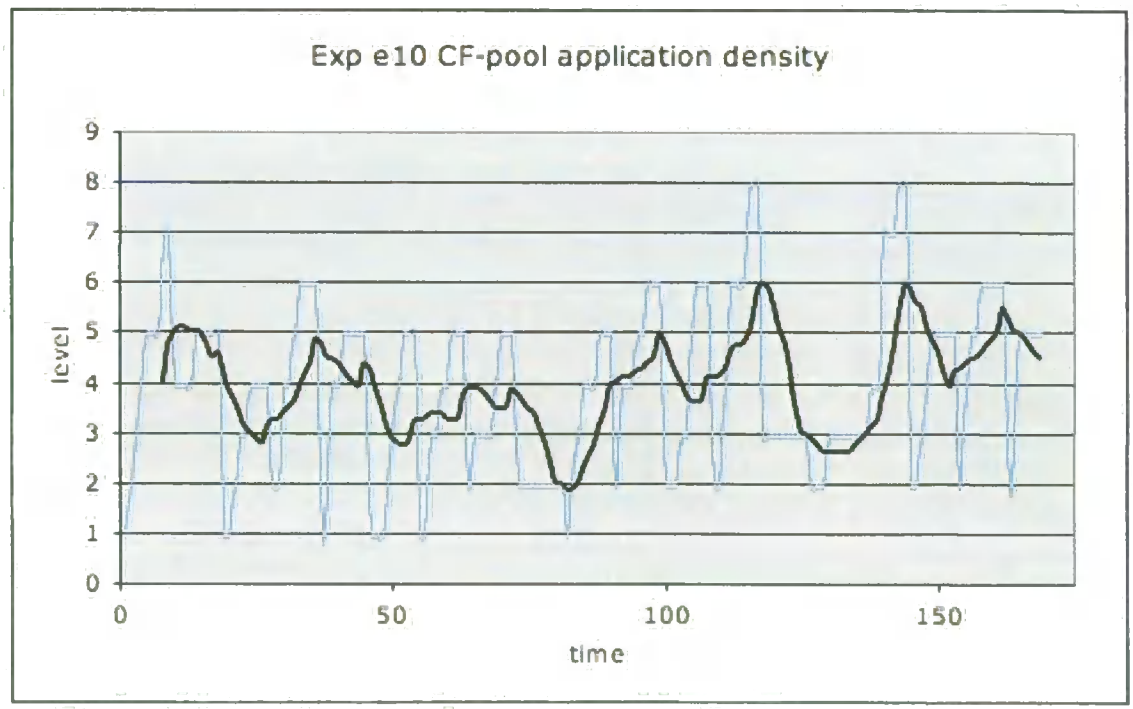


Figure 10.94: compound-function pool application density in e10.

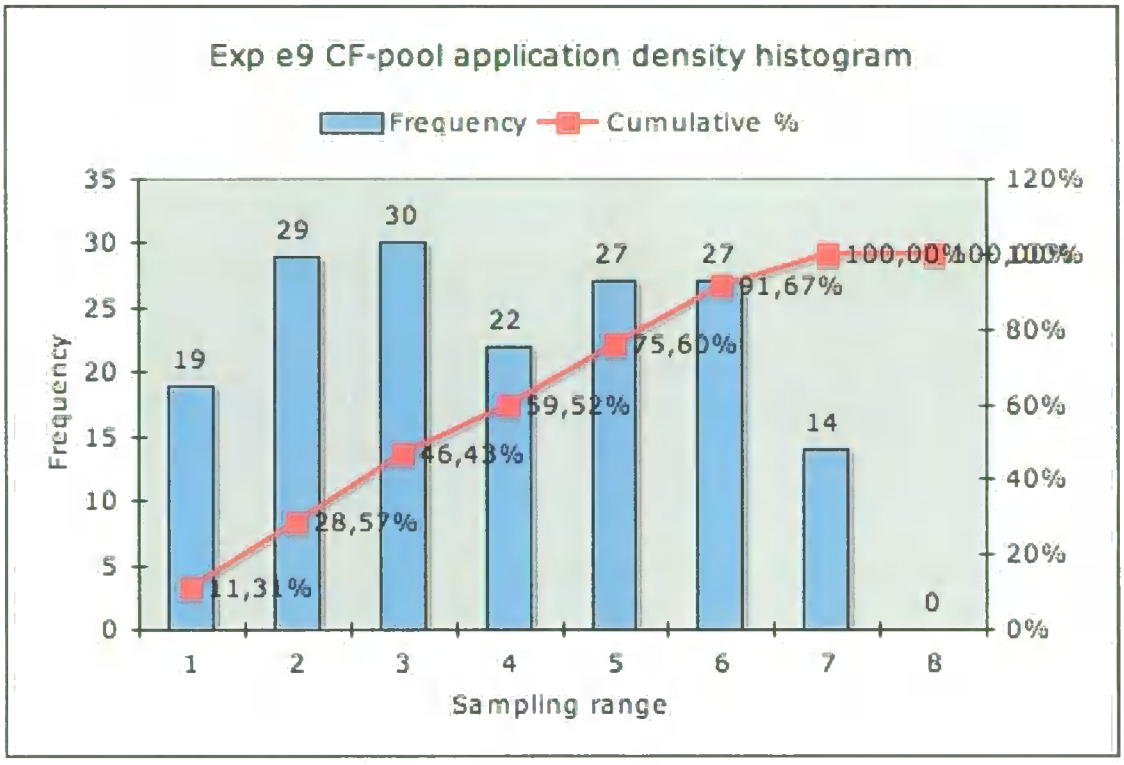


Figure 10.95: CF pool application density histogram in e9.

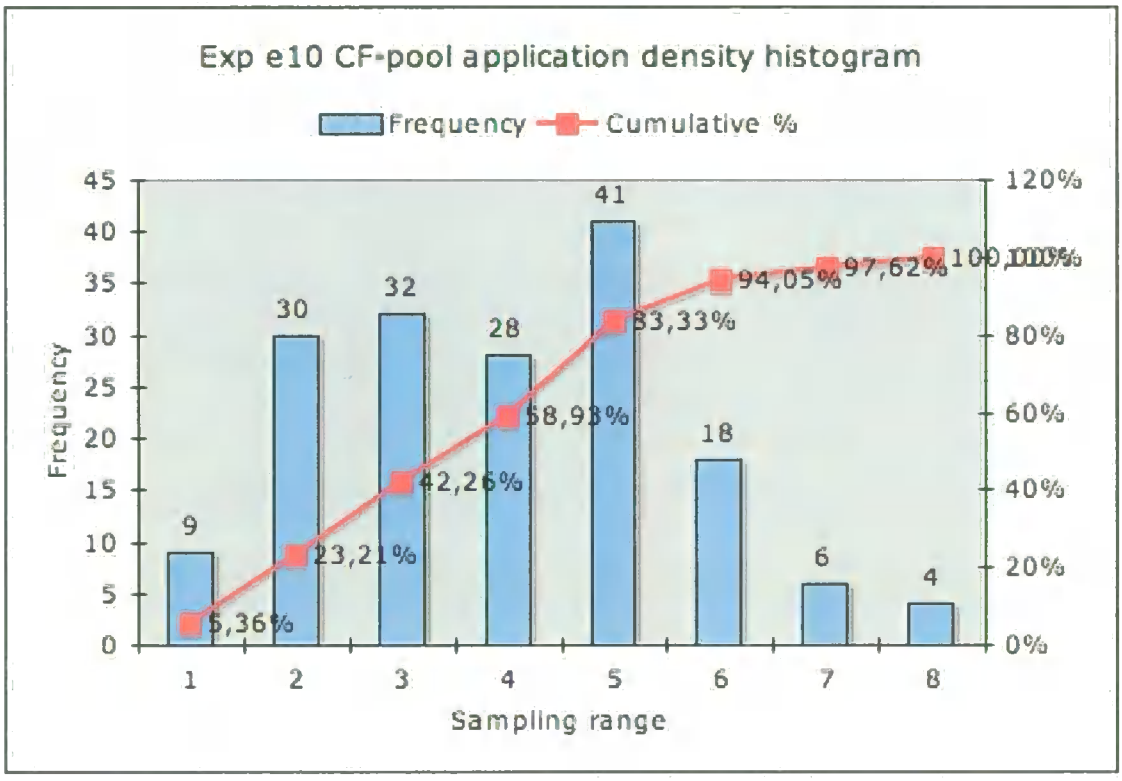


Figure 10.96: CF pool application density histogram in e10.

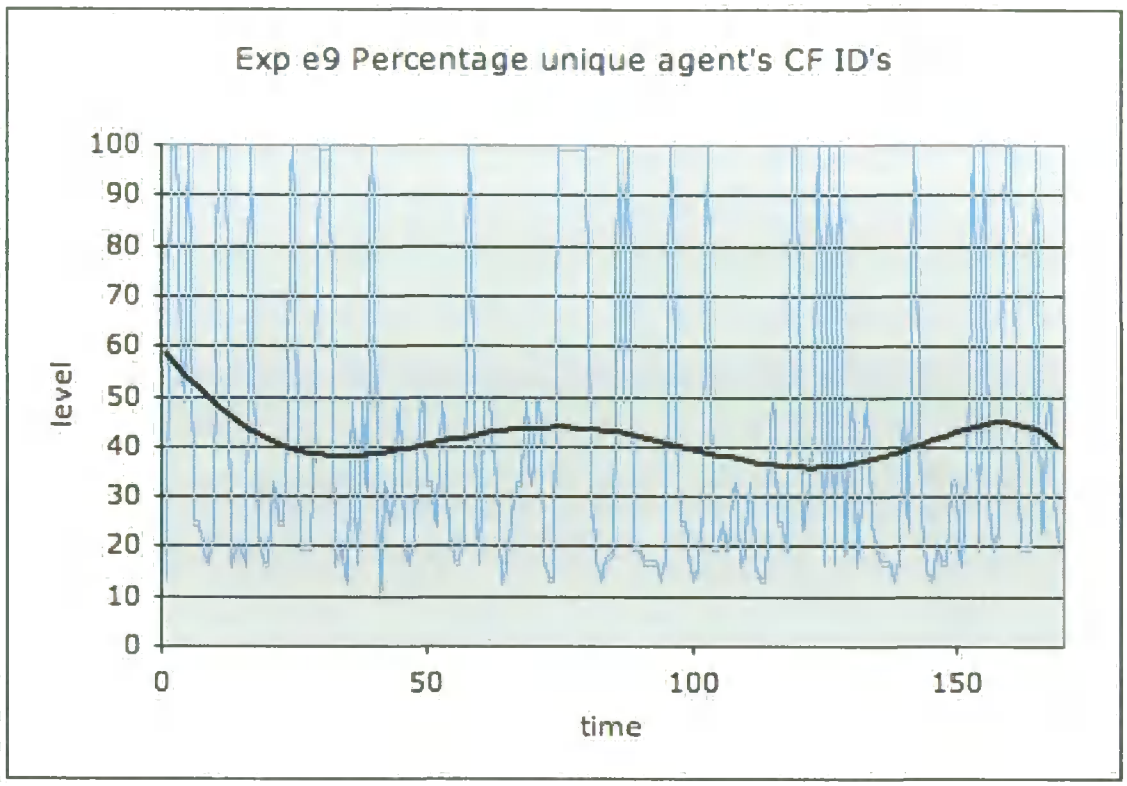


Figure 10.97: Percentage unique agent's CF ID's in e9.

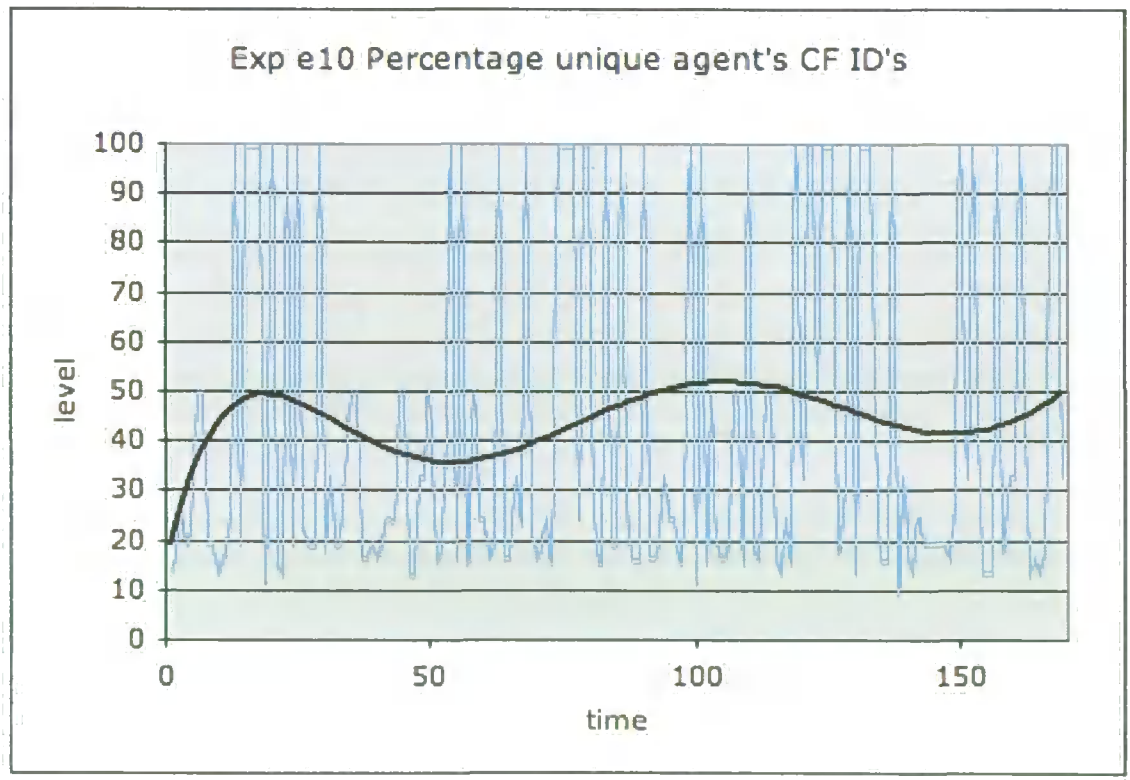


Figure 10.98: Percentage unique agent's CF ID's in e10.

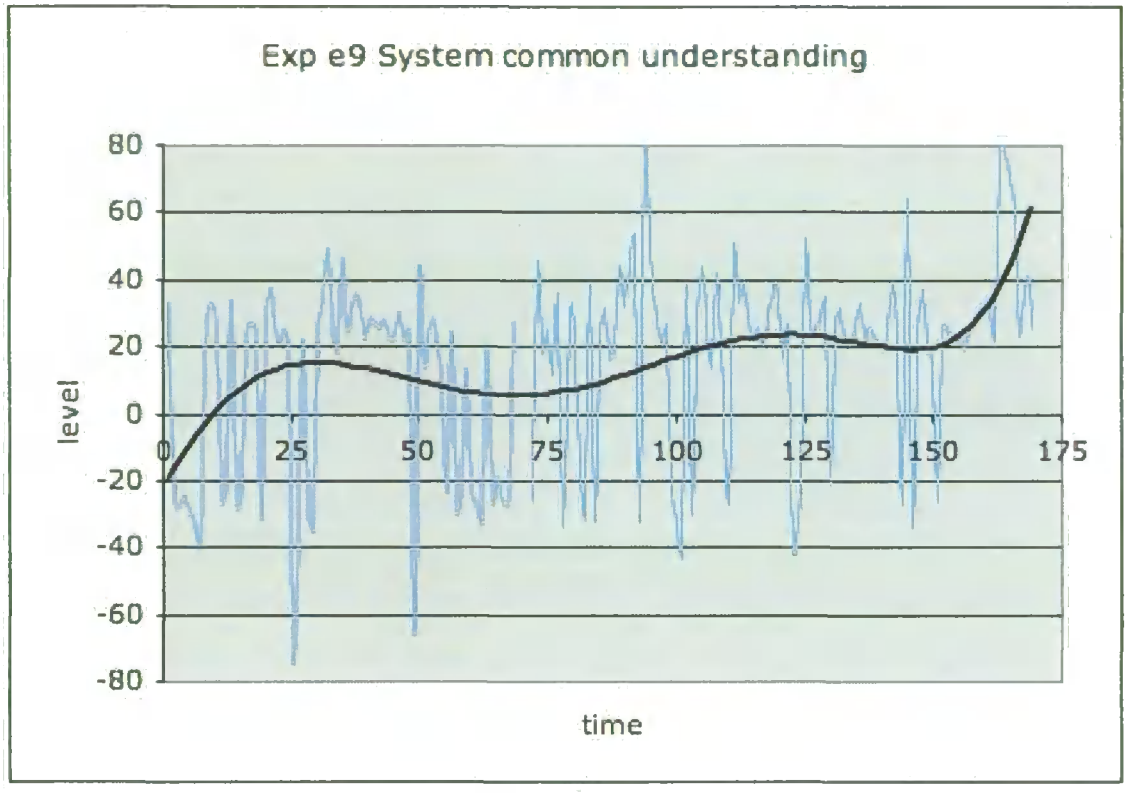


Figure 10.99: System common understanding in e9.

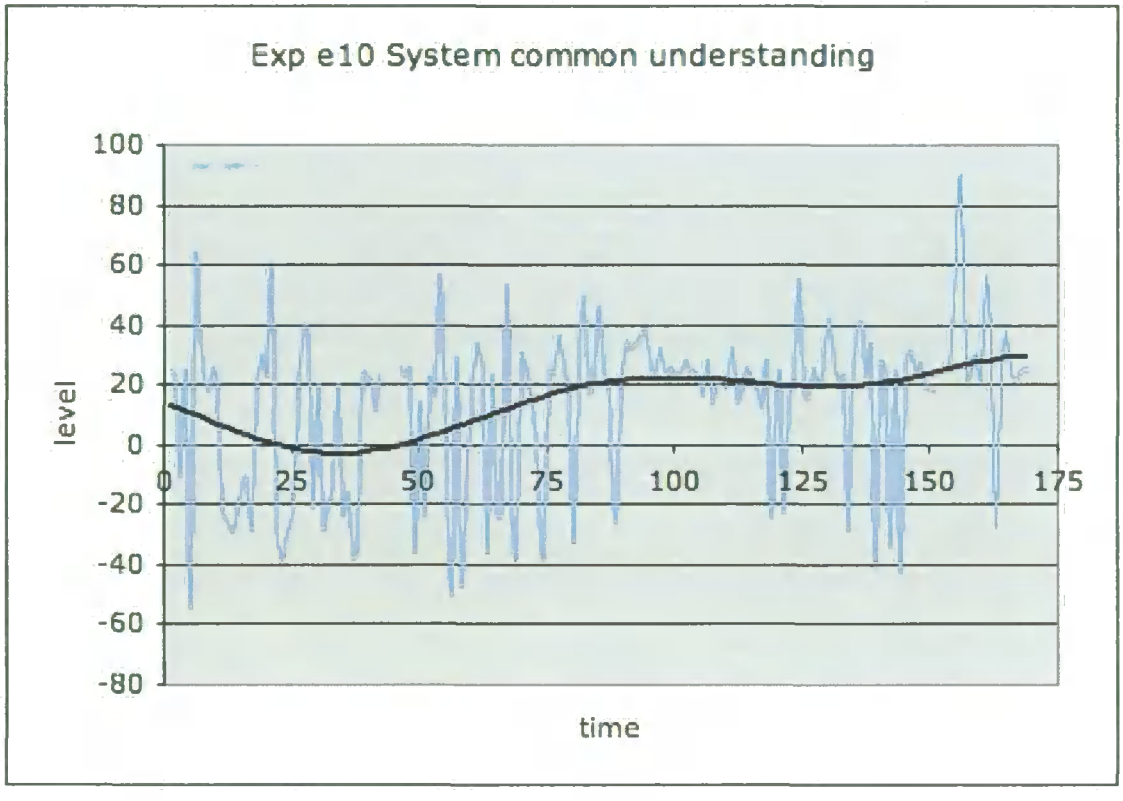


Figure 10.100: System common understanding in e10.

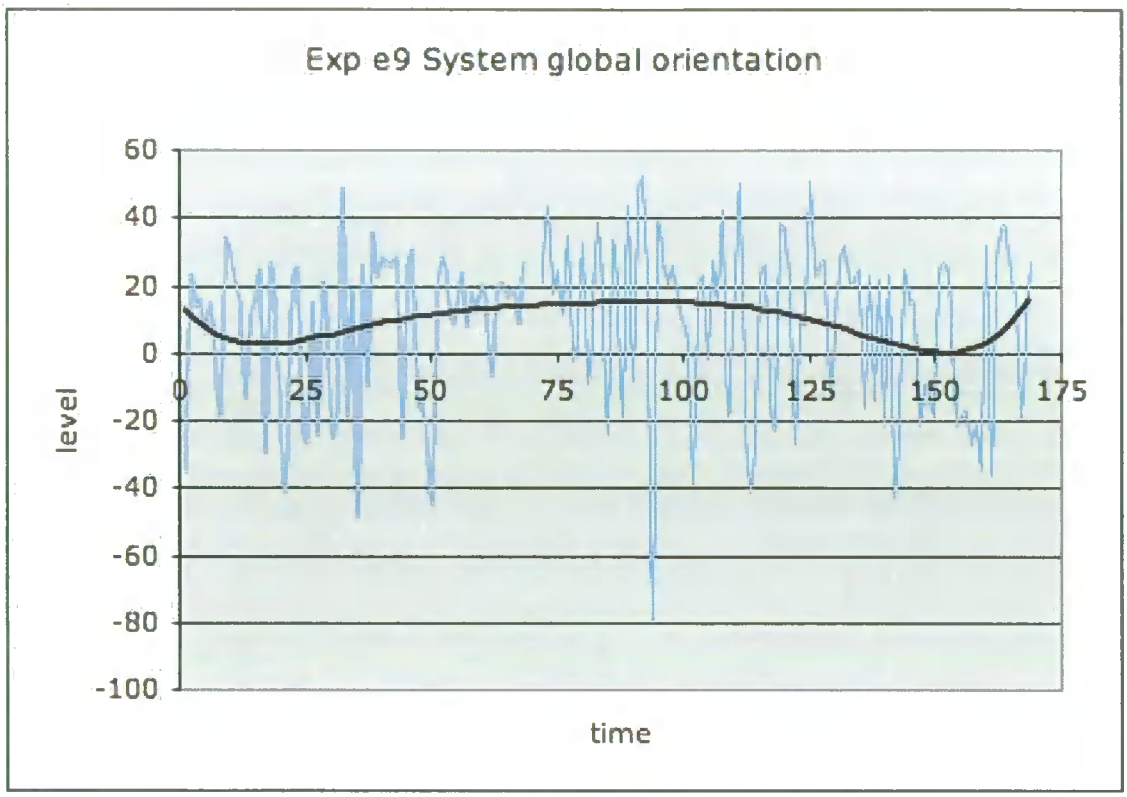


Figure 10.101: System global orientation in e9.

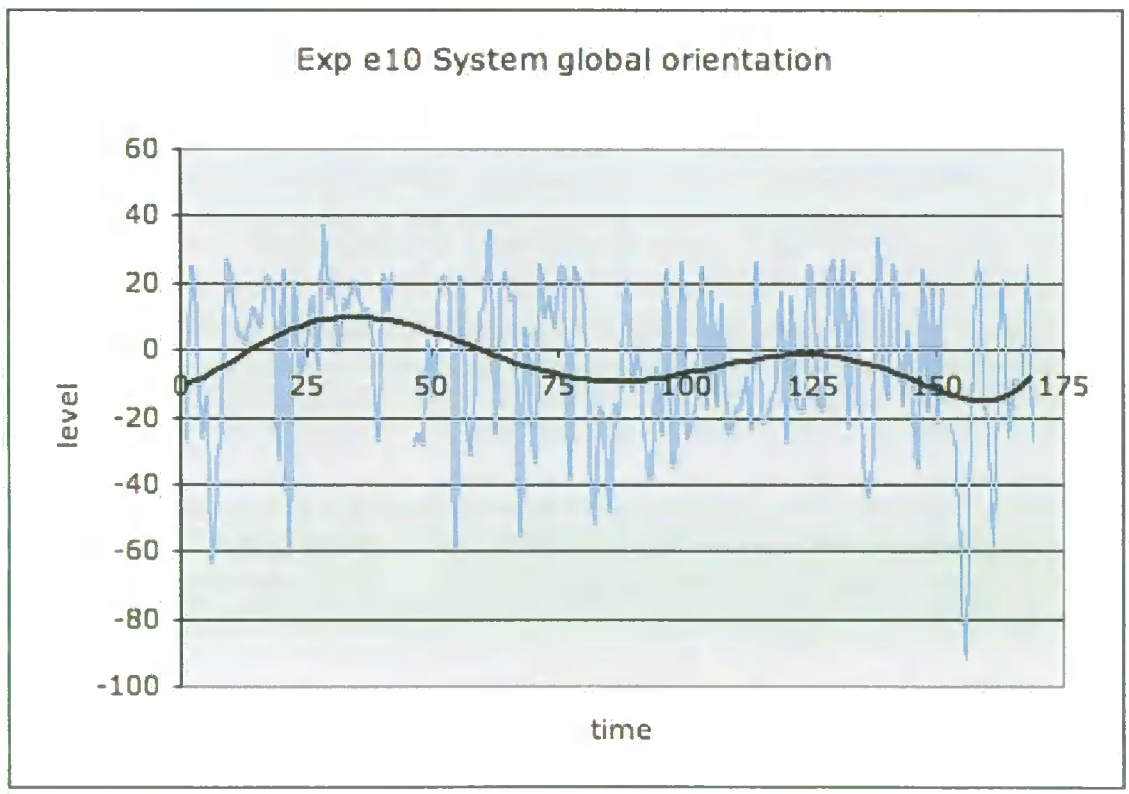


Figure 10.102: System global orientation in e10.

10.4.6 Overview and Interpretation of Data of Experiments e7 to e10

History of man-machine total similarity: figures 10.31, 10.32, 10.67 and 10.68.

Figures nr. 10.31, 10.32, 10.67 and 10.68 show the history of man-machine total similarity. All figures display spiking behaviour with occasional zones of more stability. The erratic pulsing of the similarity between the last input sequence provided by the human performer and the last contents of the reference agent's compound-function (CF, also a melody object) reveals a relative weakness of CF. Selecting another CF from the current CF-pool introduces new musical processing functions that differ from the current ones in significant ways. Thus large steps are taken in the state space defined by the CF. In other words, the musical operators in the CF are too discrete, one would wish for more continuity. In addition, noise is introduced because the real-time segmentation algorithm does not always make a correct decision. In such cases, the most recent human input is not entirely captured correctly.

The polynomials disclose slowly undulating behaviour; this proves that man and machine are actually involved in a process mutual influence. Considering figure 10.31, the dip around sample 650 coincides with the dip at sample 160 in figure 10.55. This observation generates evidence of the inability of the CF-pool to develop fit processing functions facing unpredictable and rapidly changing human input.

History of human responsiveness: figures 10.33, 10.34, 10.69 and 10.70, history of the exploitation vs. exploration pressures generated by human input: figures 10.35, 10.36, 10.71 and 10.72 and history of the quantity/quality ratio of human input: figures 10.37, 10.38, 10.73 and 10.74.

As a reminder, human responsiveness is computed by tracking the strength and frequency of the activity of the human performer; it is proportional to the quantity and quality of the human input and inverse proportional to the current-no-input-gap. The exploration/exploitation ratio is updated according to the amount and sign of the changes in human responsiveness. For example, if human responsiveness decreases, exploitation pressure increments using a multiplier. Consequently, exploitation and exploration interact in non-linear ways while the exploitation/exploration ratio only conditions the selection of compound-functions and drives.

Considering figures 10.34 and 10.36, the incremental tendency of human responsiveness – indicating an aggressive human interactor – is paralleled by a sharp decrease in exploitation level. Only exploration pressure remains positive. The data in figure 10.38 confirms a significant boost of the quality of human input. The strong waving quantity polynomial echoes the powerful dynamics of the human performer.

Taking figures 10.69 and 10.71 into consideration; relative stability with minor oscillations (around a centre value of 50) at sample 500 of human responsiveness entails a narrowing distance between exploration and

exploitation, meaning equal pressures, or a relative impasse for this behavioural activity. This observation demonstrates a successful continuous, (rather than discrete) coupling between features of a real-time MIDI input stream and the consequent appropriate adaptation of internal system pressures.

Patcher global orientation and drives-pool global orientation: figures 10.39, 10.40, 10.41, 10.42, 10.75, 10.76, 10.77 and 10.78.

Since patches and drives are functionally complementary, both are addressed simultaneously. In figure 10.39, the patcher orientation polynomial slowly oscillates between positive (integration) and negative (expression) values. In figure 10.40, expression dominates throughout which is congruent with a *selfish* systems inclination. However, the exceptional character of the data solicits an alternative explanation. It might be that relationships inside most patches were inappropriate at instantiation time. Therefore, the patcher was unable to develop integration pressure. However, a gradual increase towards value zero is also clearly observed. Strikingly, such exceptional results were only encountered in experiment e8. The data in figure 10.75 gradually develops a tendency for integration, again congruent with a *social* systems inclination.

The drives in figures 10.41 and 10.42 show an ascending data profile while the data in figures 10.77 and 10.78 display a descending profile. This might indicate that when learning is off, the drives develop negative values; i.e., a tendency for expression.

Number of events in the compound function that resides in the reference agent of the machine players' agency: figures 10.43, 10.44, 10.79 and 10.80.

The number of events of a machine-generated response depends on the current values of the input-pressures-vector inside the current reference agent, more precisely on the contrast between the vector's three values (see chapter 6, section 6.6.2). At any time, the CF creates a sequence of between 1 and 20 events, clearly oscillating around a centre value of 10.

Human vs. machine agreement: figures 10.45, 10.46, 10.81 and 10.82.

These figures monitor the relationship between two independent values; they are modified respectively when the human and the machine just finished playing their most recent sequence. The values are updated according to the new man-machine melodic distance; i.e., the interval in difference between both sequences. All values are initially zero and increment as soon as the interaction gets going. Remarkably, figure 10.45 and figure 10.46 (somewhat less so) demonstrates a steady increase in agreement for both man and machine. Figures 10.81 and 10.82 show oscillating values. When both values are highly similar *and* of a high value there is evidence that both parties managed to develop musical functionality to perform in a common effort with shared objectives. In other words, human and machine expose adaptive behaviour. In conclusion, these observations reveal emergent goal directedness when learning is on.

Activity in the drives-pool: figures 10.47 to 10.52 and figures 10.83 to 10.88.

Let us consider the drives-pool efficiency levels of all four experiments as shown in figures 10.47 and 10.48 (experiments include learning) and figures 10.83 and 10.84 (experiments exclude learning). Since the drive efficiency level echoes how well the relationships inside a given drive actually contribute to the fulfilment of its current objective; i.e., integration or expression – changes in that efficiency level might reveal complex oscillatory behaviour. The complexity follows from positive and negative feedback inside the motivation network and how it interacts on a macroscopic level in the global systems architecture. Now, the drives endowed with learning produce a quite constrained, sawtooth-like pattern. Drives without learning create much wider and irregular oscillations. Learning seemingly helps to guide the evolutionary process – in particular given the incremental trend of the data collected in figure 10.47.

Figures 10.49, 10.50, 10.85 and 10.86 document drives-pool output levels. When learning is on (figures 10.49 and 10.50) both competing levels remain more or less in the same relationship, in figure 10.49, integration pressure persistently dominates though both figures also reveal the steady rhythm of the evolutionary process. The polynomials disclose a slowly undulating data profile of low amplitude. The picture is slightly different in figures 10.85 and 10.86; the difference between integration and expression levels is more

differentiated. Again, learning seems to impose a stabilizing effect on the levels' competition.

In addition, the data in figures 10.85 and 10.86 evolves from primarily integration towards mainly expression at the end. When taking this information in relation to the ascending data profile depicted in figures 10.99 and 10.100; i.e., the incremental nature of the system common understanding levels, expression levels and understanding clearly demonstrate positive correlation. Therefore, it is concluded that human *and* machine opt for expressive behaviour towards the end of experiment e9 and e10.

The drives-pool understanding levels (figures 10.51, 10.52, 10.87 and 10.88) show a relatively flat data profile, except for figure 10.87. The latter shows a strong incremental data curve, which coincides with the incremental nature of the data in figure 10.99, the system common understanding record. The correlations in the other experiments – relatively flat, low frequency polynomials for understanding-level and system common understanding – confirm this conclusion. In other words, the data in both figures should be congruent because the understanding level of the drives-pool is scaled up or down according to the degree of common understanding (see chapter 8, section 8.6).

A clear correlation is also observed between the drives pool output levels and the compound-function pool fitness histories documented in figures 10.55, 10.56, 10.91 and 10.92. The polynomials in the respective data sets are

complementary, for instance, the crossings of red and blue polynomials in figures 10.85 and 10.86 (drives-pool levels) and the swings in the polynomials of respectively figures 10.91 and 10.92 are complementary. In all figures, the tendency is towards more expression and the turning points (for instance, at generation 80 in experiment 10) are perfectly synchronized.

Patcher fitness history: figures 10.53, 10.54, 10.89 and 10.90.

Patcher fitness data shows very irregular patterns with occasional peaks. The positive feedback inside the listening network in combination with the firing activity in sensors and neurons introduces quite unpredictable output; more precisely, it proves extremely difficult to uncover correlations with other datasets.

Data levels in the compound-function-pool: figures 10.55 to 10.60 and figures 10.91 to 10.96.

The evolution of fitness in the CF-pool follows a regime of clustering preferences. For instance, in figure 10.55, the CF-pool develops areas of very high expression fitness; i.e., musical processing functions that work very well for the purpose of expression, at the beginning of the interaction. Figure 10.91 shows an evolution from integration towards mainly expression. The clustering nature of the data entails a high amount of consistency in machine responses. A social machine inclination (figures 10.55 and 10.91) reveals more consistency than selfish inclination; the latter seemingly fostering more

competition between integration and expression fitness causing more erratic level changes.

Observe the exploration boost at generation 600 (tracer data) in the exploration-exploitation pressures data (figure 10.71) to coincide with an evident transition in the CF-pool fitness history (figure 10.91) from a flat integration regime to a sudden peaking winning fitness for expression at generation 150 (evolution data). This observation is further confirmed by a comparative transition from integration to expression in the drives-pool levels (figure 10.85).

Figures 10.57, 10.58, 10.93 and 10.94 document the oscillations in CF-pool application density; i.e., how many compound-functions are distributed to the players' agency – this number also reflects the number of agents inside the current agents cluster. Clusters of seven agents are extremely rare. Also, coherent micro-oscillations are spotted suggesting self-organised behaviour constrained by temporal regularity in the agents' pattern formation. The wave-like outline of the black curve (the floating average grouping eight data samples) suggests a similar regularity of spatiotemporal structures on a larger scale. This observation echoes the fractal-like nature of the general networked architecture proposed in chapter 1, section 1.2.

Figures 10.59, 10.60, 10.95 and 10.96 show CF-pool application density histograms; how many times every CF in the pool is actually put to work, a number that also reveals information on the relative divergence in usefulness (fitness) of all CF in the pool. A sharper profile emerges given that learning is

on (figures 10.59 and 10.60), compound-function nr. 4 is the more popular one. The data shows a Gauss-like distribution. Not so in figures 10.95 and 10.96 where learning is off. The data shows far less contrast; in particular figure 1095 shows a nearly flat distribution, also, function nr. 8 is never applied. Learning thus promotes idiosyncratic machine behaviour because it develops temporal biased preferences for the selection of specific musical processing functions – another form of emergent functionality.

Number of unique compound-function ID's in the players' agency: figures 10.61, 10.62, 10.97 and 10.98.

Specificity of CF selection is conditioned primarily by three factors: (1) the indented orientation to take next (instructed by the current drive), (2) the quality of the contents of the CF-pool; i.e., the current contrast and proportion of numerical availability between the functions fit for expression and those fit for integration and (3) the current exploration-exploitation-ratio (for details, please see chapter 9, figure 9.11). Specificity of 100% results when only a single candidate function exists to be distributed to *every* agent. Zones of regular spiking of maximum specificity demonstrate sustained interactions between the motivation network and the players' agency.

Global output: figures 10.63 to 10.66 and figures 10.99 to 10.102.

This group of figures documents system behaviour at the highest level of generalization. System common understanding shows areas of relative

stability and areas of oscillations constrained between specific upper and lower levels. Figures 10.63 (learning is on) maintain a remarkable interaction climate characterized by man-machine *agreement*. From figure 10.65 it is discovered that system global orientation values remain primarily negative throughout, thus both man and machine are engaged in an extended process of mutual *expression*. A similar picture shows up for experiment e8.

Data profiles for experiments e9 and e10 are in sharp contrast to the data in the previous experiments; system common understanding equally develops agreement, however, given an orientation of *integration* throughout (experiment e9) and an orientation oscillating around zero (experiment e10).

A strong correlation exists between the incremental nature of the drives-pool understanding level (figure 10.87) and a similar profile in system common understanding level (figure 10.99).

System-global-orientation level and the level of man-machine understanding receive further comparative examination in the next section.

10.4.7 Consideration of Global Comparative Results

10.4.7.1 Comparative Visualisation of Results

The *system-global-orientation* level and the level of *system-common-understanding* are important indicators of Oscar's global behaviour. This section provides a comparative analysis by way of histograms.

Figures 10.103 to 10.110 are organised in two columns. The left column compares all four experiments while making the sum of all data samples that are positive (light blue) and the absolute sum of all negative samples (red). The right column addresses the same samples however counting the number of positive and negative samples. Both values are averages taken over the data in all eight objects in the current population.

First of all, the activity in the patcher and the drives-pool is taken up because both collections of objects reflect the activity of respectively the human interactor and the synthetic performer, Oscar itself. This information is further employed to compute the two top-most behavioural data: (1) the system global orientation and, (2) the degree of system-common-understanding.

Remember, a patch and a drive both contain relationships whose continuous action results in the formulation of two competing output variables, one holding positive, and the other holding negative values. These values are interpreted as competing pressures; positive values aim for human-machine integration, negative values reflect the expression level. The system-global-orientation is computed from the ratio between the orientations of patcher and drives-pool. Finally, the degree of human-machine understanding – the system common understanding – is the amount of

contrast between levels of agreement and conflict. For a complete discussion, please refer to chapter 8, section 8.6.

The system-common-understanding represents the top-most impression of the global systems behaviour. As introduced in chapter 1, the implicit intention of Oscar is to develop networks and musical processing functions that are optimised towards generating *agreement* between human and machine as musical partners. Agreement implies that both man and machine are competent to develop functionality that contributes to sustain the current *system-global-orientation*, irrespective of whether it is integration or expression. We shall see that the experiments reported here show strong evidence that the current systems architecture manages to support such functionality successfully.

Let us try to infer the effect of (1) learning and (2) system inclination from inspection of the data exposed in these figures. In order to develop a formal ground to compare data pairs, a measure of contrast between the respective values is computed [4].

[4] Dividing the absolute value of the difference of the two values by their sum and multiplying the result by 100 computes the amount of contrast, in percent.

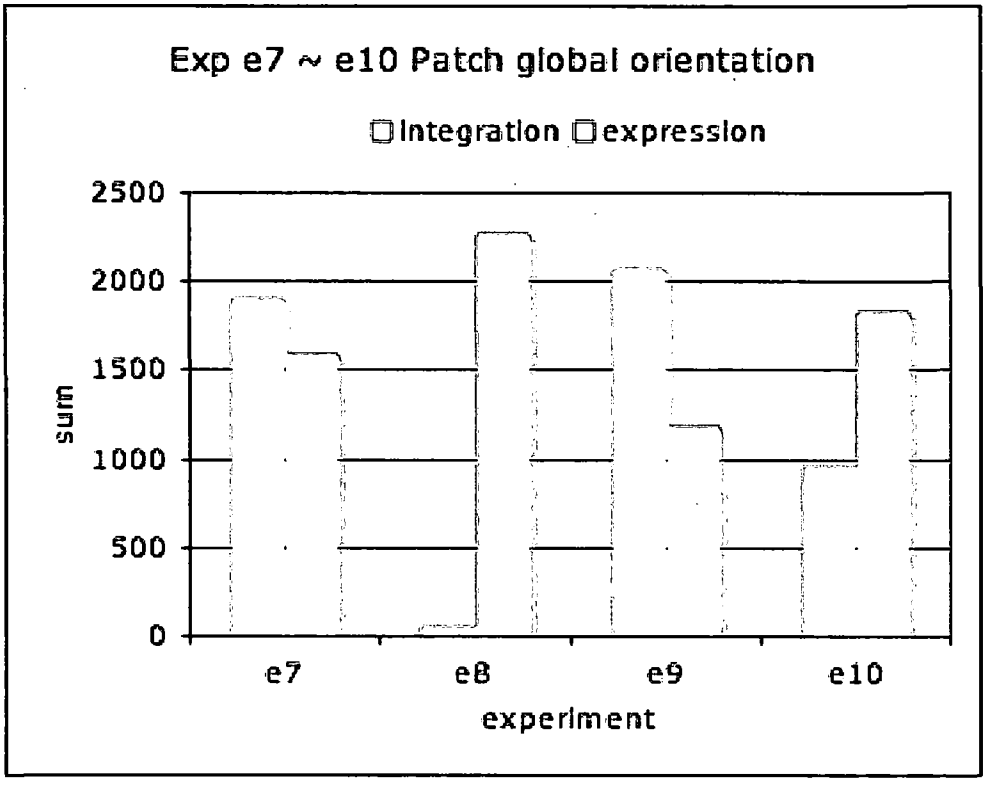


Figure 10.103: Sum of patch pressures in e7 to e10.

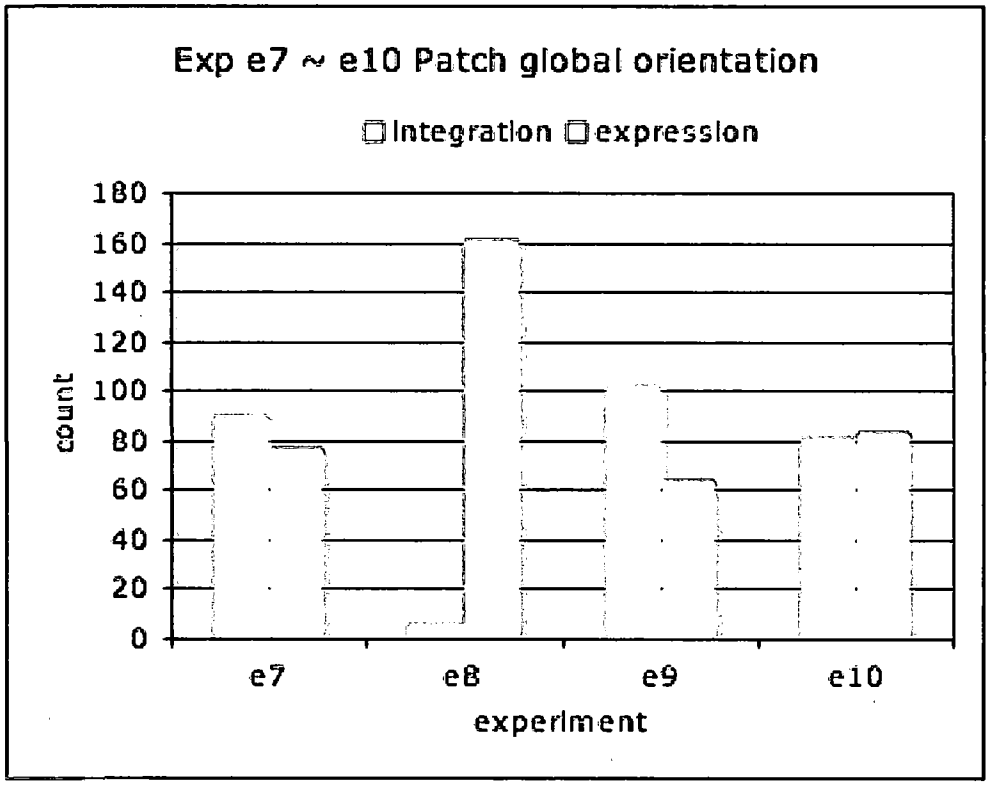


Figure 10.104: Count of patch pressures in e7 to e10.

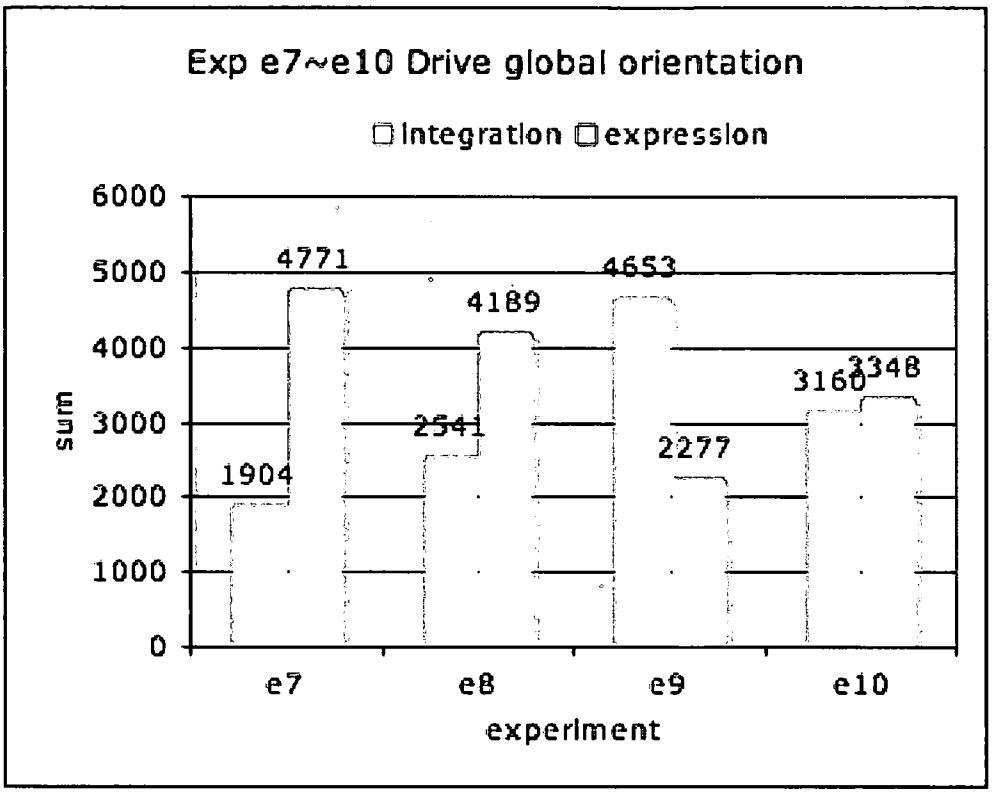


Figure 10.105: Sum of drive activation in e7 to e10.

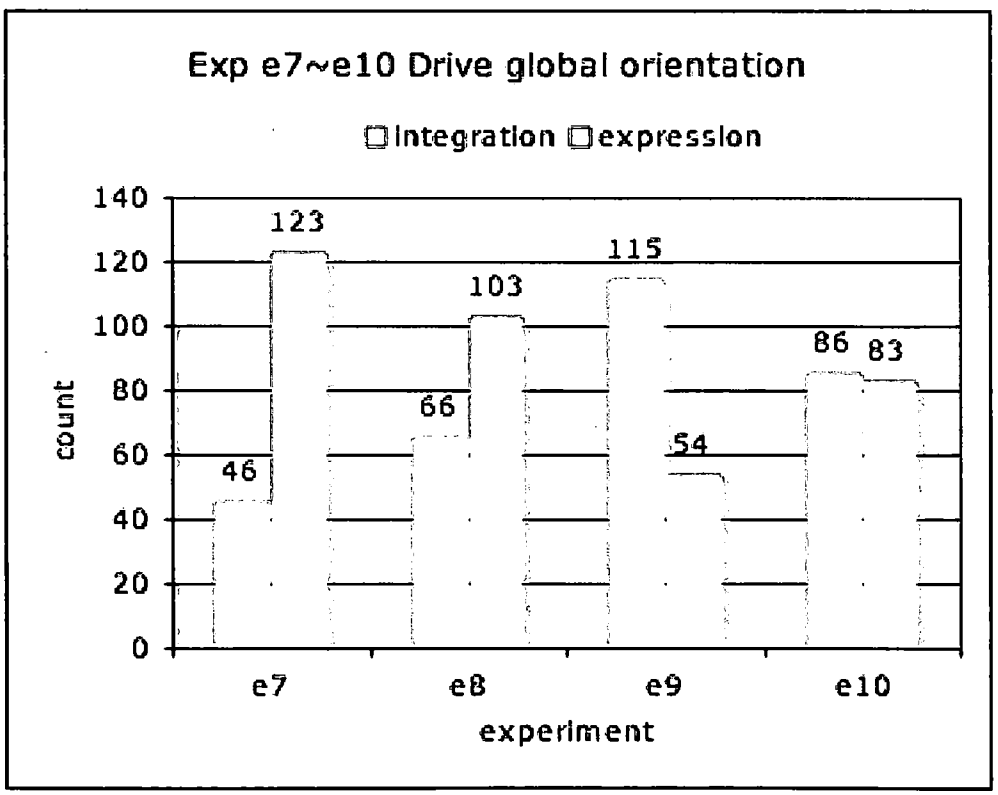


Figure 10.106: Count of drive activation in e7 to e10.

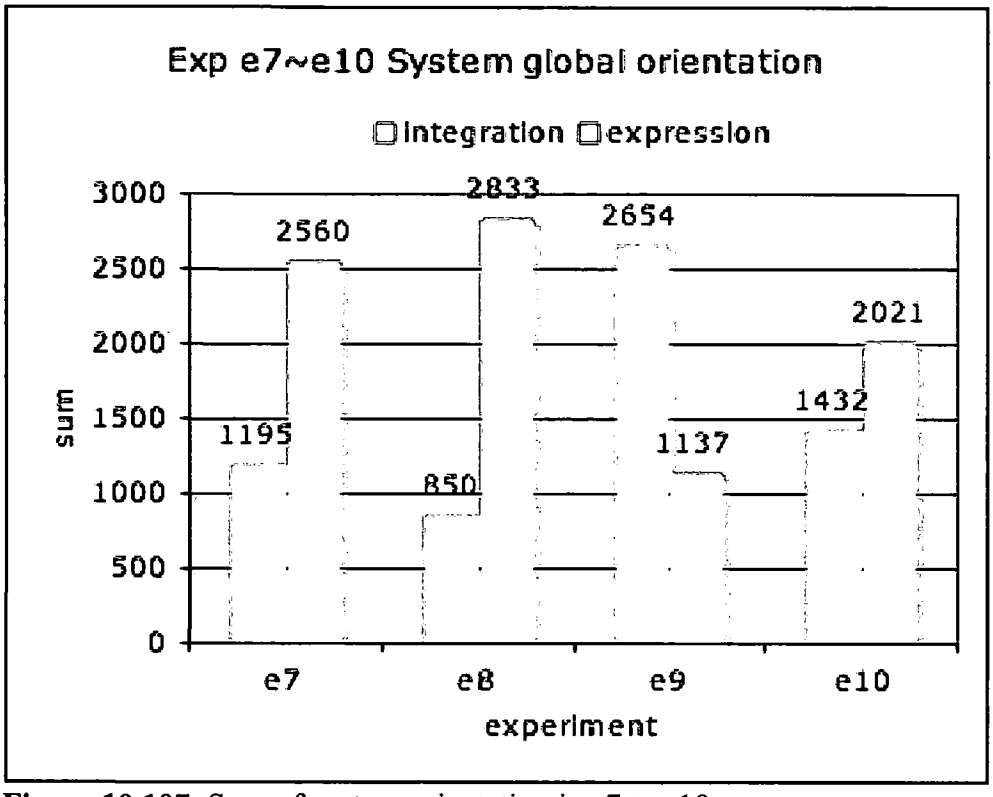


Figure 10.107: Sum of system orientation in e7 to e10.

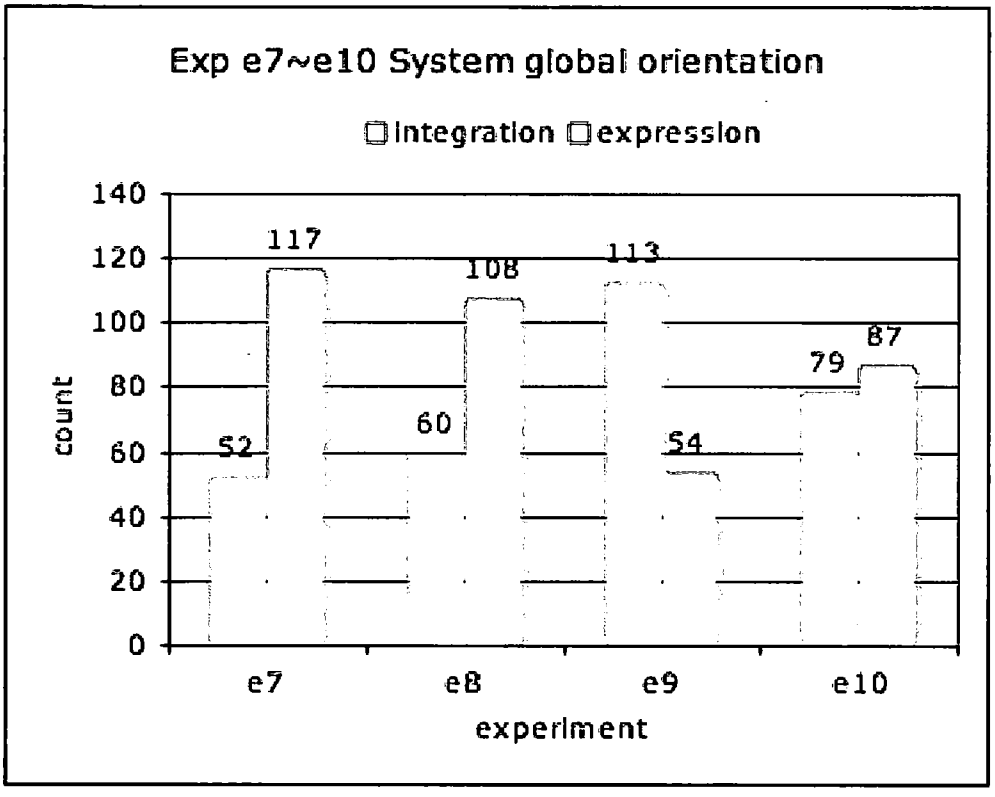


Figure 10.108: Count of system orientation in e7 to e10.

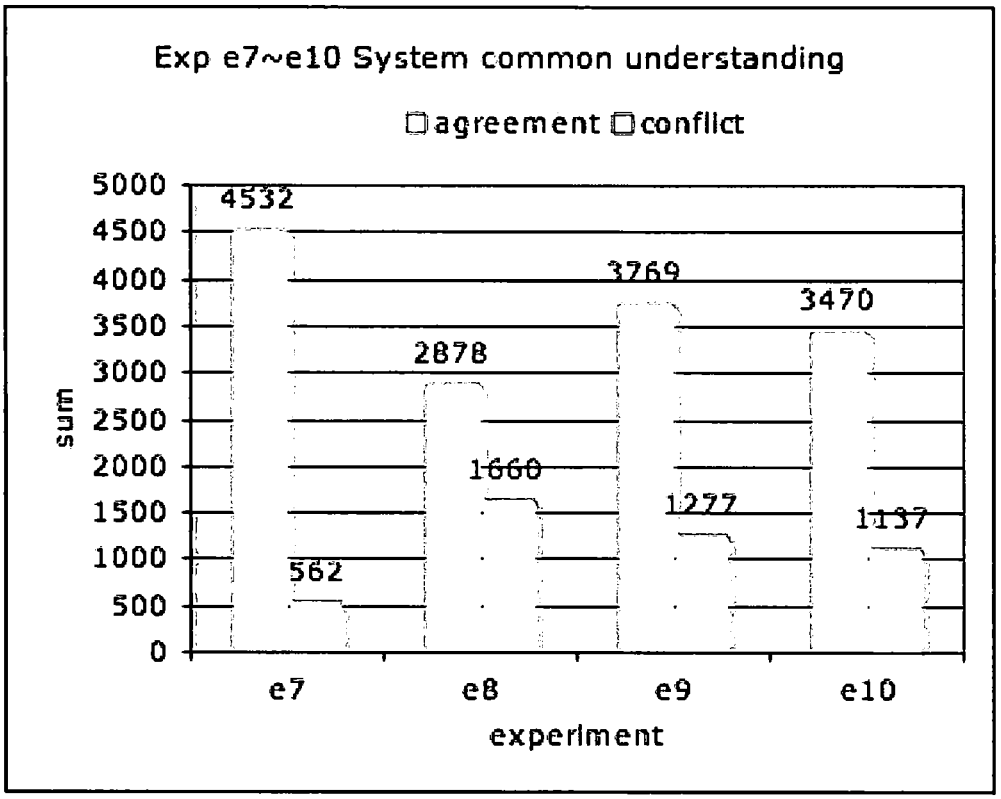


Figure 10.109: Sum of common understanding in e7 to e10.

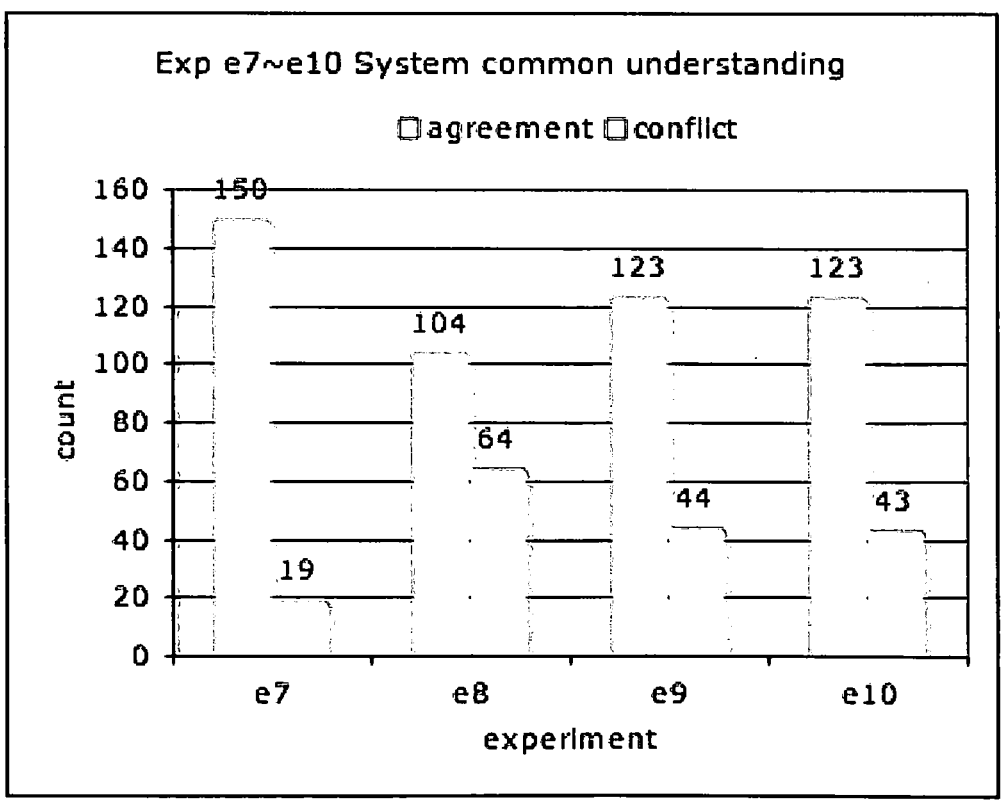


Figure 10.110: Count of common understanding in e7 to e10.

10.4.7.2 Evaluation of Results

Figures 10.103 to 10.110 document global results obtained from the four systematic experiments e7 to e10. Every figure compares two numerical values reflecting the status of a given measurement, given the results from the four experiments. All numerical values are evaluated against a relative scale; all data samples are scaled by a factor 10, 100 or 1000 in order to derive consequent values prepared for visualisation.

Figure 10.103: Patcher global orientation, observing the sum of all patch pressures.

The results from experiments e7 and e8 are highly dissimilar; they reveal the complete inability of the patcher to develop integration pressure when the system is configured as *selfish* (e8). Only 64 samples (2, 75%) provide a positive value and 2270 samples (97,25%) are negative. When configured as *social* (experiment e7), integration pressure supersedes expression pressure according to the following ratio: 54,4% positive samples, 45,6% negative samples.

Important, a patch object is not in any way directly linked to the system inclination parameter. Only the *adjust-fitness* method associated with the compound-function-pool addresses the system inclination parameter (chapter 6). Remarkably, the individual activity in the different system components develops an emergent peaking of negative values – the conclusion is that expression pressure wins which seems logical given a selfish systems orientation. This outcome is not programmed in any way; in this case, it proves that the motivation network (incorporating the CF-pool) manages to favourably propagate information to the perception network (chapter 1). In

addition, this proves the ability of a patch to develop favourable relationships in time, relationships that implicitly contribute to the improvement of man-machine understanding in the long run. Globally speaking, the effect is the development of distinctive systems behaviour.

Learning is disabled in experiment e9 and e10. Both experiments offer divergent results, however, less contrasting than experiments e7 and e8. Experiment e9 with *social* system inclination develops more integration than expression pressure. Experiment e10 with *selfish* inclination, it is just the other way round. Both observations are congruent with the understanding that a social inclination should generate listening networks that offer more potential to integrate than express.

When considering contrast in figure 10.103, the patch global orientations contrast for experiments e7 to e10 are respectively, 8.6%, 94.5%, 27.2% and 31.1%. In experiment e8, the human performer remains helpless to develop positive patch levels; since patch objects do not feature learning, the evolutionary chain clearly escalates into an *only expression* point attractor regime. Interestingly, for experiment e8, the highest patch level contrast coincides with the lowest contrast (yet still substantial global agreement) in system common understanding (26.8%) – as displayed in figure 10.109. This proves that the listening and the motivation networks do effectively interact *and* that the system, *as a whole*, manages to accommodate the unsolicited effect of point attractors successfully, clearly another instance of emergent

functionality. In other words, the networked architecture developed in this thesis fosters internal spontaneous cooperation amongst system components.

Figure 10.104: Patcher global orientation, observing the *count* of all positive and negative patch pressures.

The results in figure 10.104 follow the data in figure 10.103. Considering experiment e10, the number of positive and negative samples is nearly identical, 82 to 84; i.e., a difference of only 0.6 percent. The divergence for experiment e10 in figure 10.103 is 960 (sum of positive samples) to 1830 (sum of negative samples) or a ratio of 35% to 65%. Therefore, a patch features few strong negative peaks as opposed to many but weaker positive ones.

Figure 10.105: Drives-pool global orientation, observing the *sum* of all pressures.

Considering figure 10.105, the drives competing orientation levels – what is the impact of learning? In experiment e7 and e8, expression supersedes integration given a contrast of respectively 42.9% and 24.5% – learning thus promotes the development of expression rather than integration, irrespective of system inclination.

Experiment e9 – with social inclination and learning disabled – integration is the winning force with a contrast of 34.3%. This profile looks similar to the data of e9 in figure 10.103; the pressures in the patch and the drive produce comparable levels, again the effect of coupling between patch and drive

surpasses the effect of learning. In experiment e10 – with selfish system inclination and without learning – the system does not develop a clear attitude; contrast is only 2.9 %, signalling an impasse in the drives pool.

Generally speaking, contrast between positive and negative samples is more prominent in a patch than in a drive.

Figure 10.106: Drives-pool global orientation, observing the *count* of all pressures.

The data in figure 10.105 is confirmed. However, the data in experiment e10 seems out of balance; the *count* positive/negative ratio is 86/83 while the *sum* positive/negative ratio is 3160/3348. Therefore, more attempts to integrate were undertaken as opposed to attempts to express – however, the amplitude of the former being inferior to the strength of the latter. Thus, selfish inclination with no learning leaves the drives in a global symmetry.

Figure 10.107: System global orientation, observing the *sum* of all pressures.

The contrast between integration and expression is maximal in experiment e7, e8 and e9. Only in experiment e9, (social inclination, no learning) integration supersedes expression, explained by the cumulative effect of patcher and drives-pool. The strongest global orientation occurs in experiment e8, a ratio of 850/2833 or 23% integration and 77% expression. Experiment e10 (selfish inclination, no learning) features the least contrast: a ratio of 1432/2021 or 41% integration and 59% expression.

Given a selfish inclination (data pairs e8 and e10), expression is the winning force with a contrast of respectively 53.8% and 17%. However, the condition of social inclination and learning enabled (e7) also generates a system orientation focussing on expression with a contrast of 36.4%. Again, the effect of learning remains unclear.

Figure 10.108: System global orientation, observing the *count* of all pressures.

The general tendencies of figure 10.107 are reflected here as well. However, the contrast between integration and expression is less prominent than in figure 10.107. Only experiment e9 develops a clear preference for integration. Experiment e10 shows approximately equal values; this echoes the values in figures 10.104 and 10.106.

Figure 10.109: System common understanding, observing the *sum* of all pressures.

This figure provides system information at the highest level of abstraction; how well man and machine adapt to one another in order to maximize mutual agreement. Maximum agreement is manifest in experiment e7 and all four experiments show evidence that Oscar exhibits a capacity to manipulate its system components in order to agree with the human interactor. Figure 10.109 illustrates the following agreement/conflict ratios (in percent):

Experiment	e7	e8	e9	e10
Agreement	89	63	75	75
Conflict	11	37	25	25

Table 10.9: Levels of human-machine agreement and conflict in experiments e7 to e10.

Experiment e7 (combining social system inclination and learning) features the highest score. Contrast in experiment e7 is 77.9%. Consequently, social system inclination and the inclusion of learning seem to provide the basis for successful human-machine interaction, where the system as a whole aims to optimise towards maximum *agreement.*

The lowest contrast is spotted in the second data pair (experiment e8); 26.8% contrast between the levels of agreement and conflict given a selfish system inclination. The difference in contrast of experiment e7 vs. experiment e8 makes it clear that a selfish attitude – which aims to optimise global system performance only considering the competing levels in the drives – is less suited to evolve fruitful interaction.

A social inclination exploits the momentary relationships between the outputs of patch (human pressure) and drive (machine pressure) in order to compute a global orientation. The result updates the fitness of the current compound-function – expressing a preference towards the musical material currently produced by the machine performer. A selfish inclination (experiment e8) aims to optimise the drive fitness by only taking *its* current orientation into account. So it seems logical to expect more agreement in the first case.

The results for experiment e9 and e10 are of nearly identical contrast; respectively 49.4% and 50.6%. This means that the constructive impact of a social orientation is no longer operational when learning is disabled. In addition, it shows that the inclusion of learning plays a much more significant role than a particular specification of the system inclination.

Figure 10.110: System common understanding, observing the *count* of all pressures.

Again the results of the previous figure are confirmed. Strikingly, the results for experiment e9 and e10 are virtually identical. It means that the system as a whole settles itself in a given behavioural niche when learning is disabled – let us refer to this as the capacity of Oscar to develop a functional behavioural identity. Keep in mind, this figure reflects the history of the fluctuations of agreement and conflict levels inside the system – the dynamics of this process are visualised in figures 10.63, 10.64, 10.99 and 10.100. At every step in time, these are computed by counting all positive and negative understanding levels. Figure 10.110 shows the actual total sum of these values gathered over a time span of approximately 40 minutes of man-machine interaction. Therefore, it is not easy to explain the unusual correlation between the values in experiment e9 and experiment e10. A general conclusion is that the inclusion of learning is instrumental to keep the system from an orientation impasse *and* helps to develop contrast between social and selfish behaviour.

In conclusion, the data in the figures 10.103 to 10.110 is often obscure to understand and difficult to interpret. However, there is ample evidence that learning plays a less vital role than the consequences of the non-linear couplings between system components and in a more general sense, between Oscar and the human interactor (chapter 1, figure 1.1). Further correlation analysis, detailed next, aims to shed light on the kind of coupling between the drives global orientation and the level of system common understanding.

10.4.8 : Correlation Analysis of Experiments e7 to e10

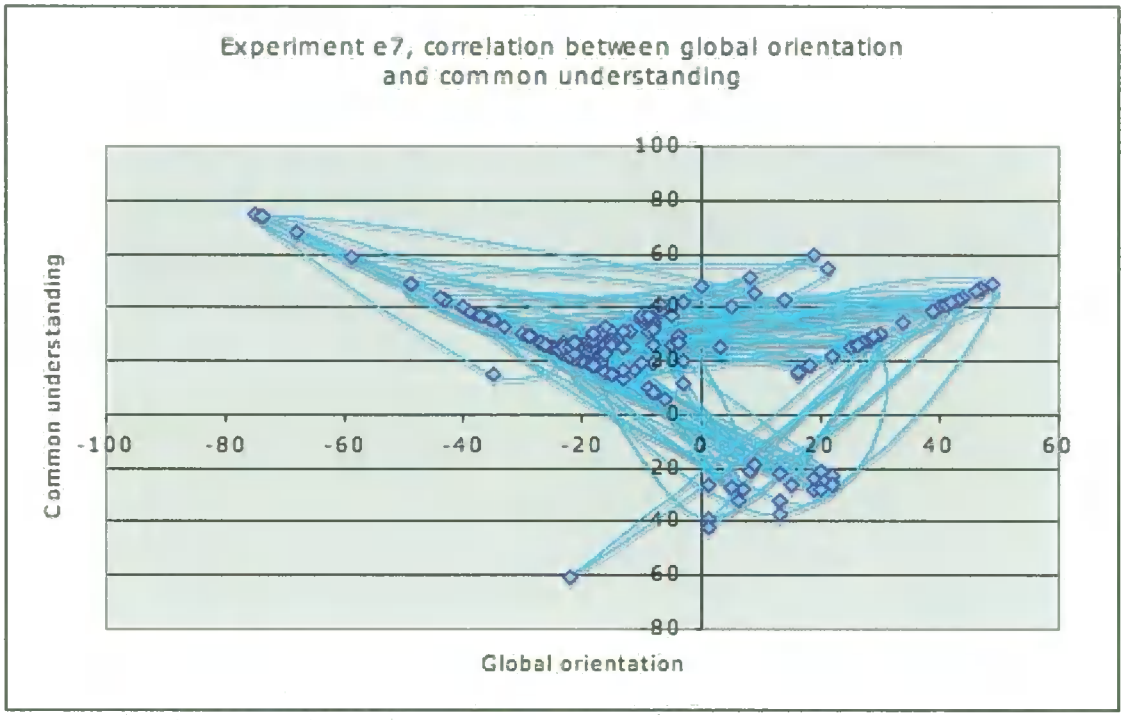


Figure 10.111: Correlation between the drives global orientation and system common understanding in experiment e7: social inclination, evolution and learning.

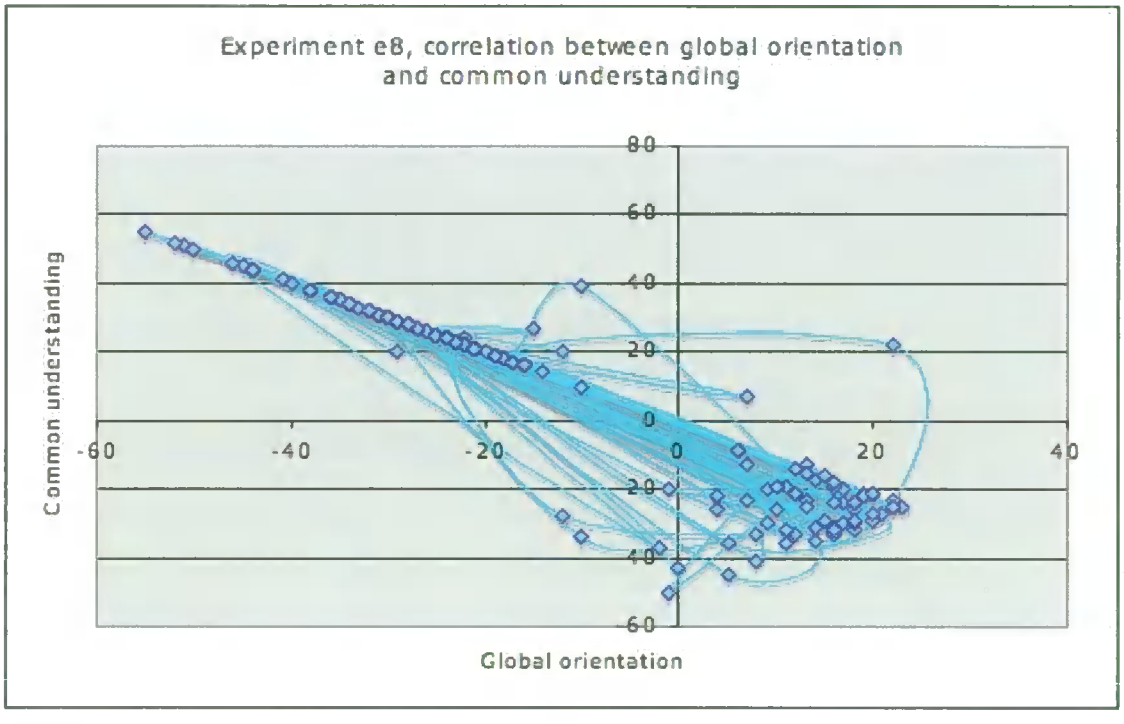


Figure 10.112: Correlation between the drives global orientation and system common understanding in experiment e8: selfish inclination, evolution and learning.

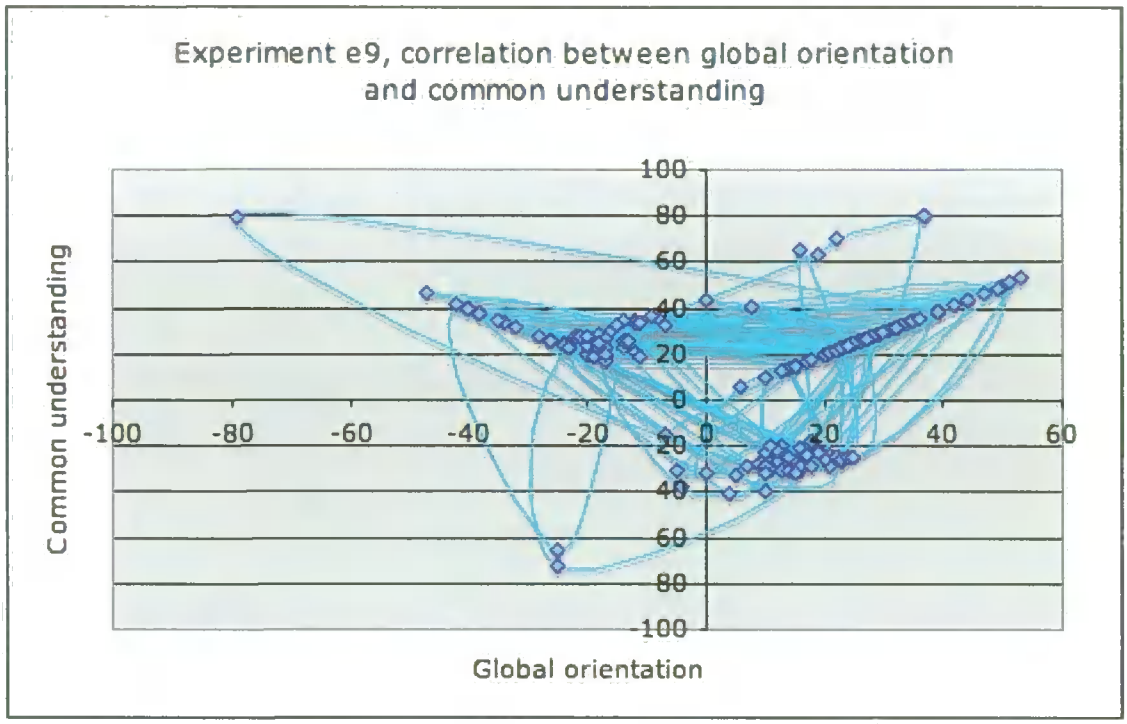


Figure 10.113: Correlation between the drives global orientation and system common understanding in experiment e9: social inclination, evolution only (no learning).

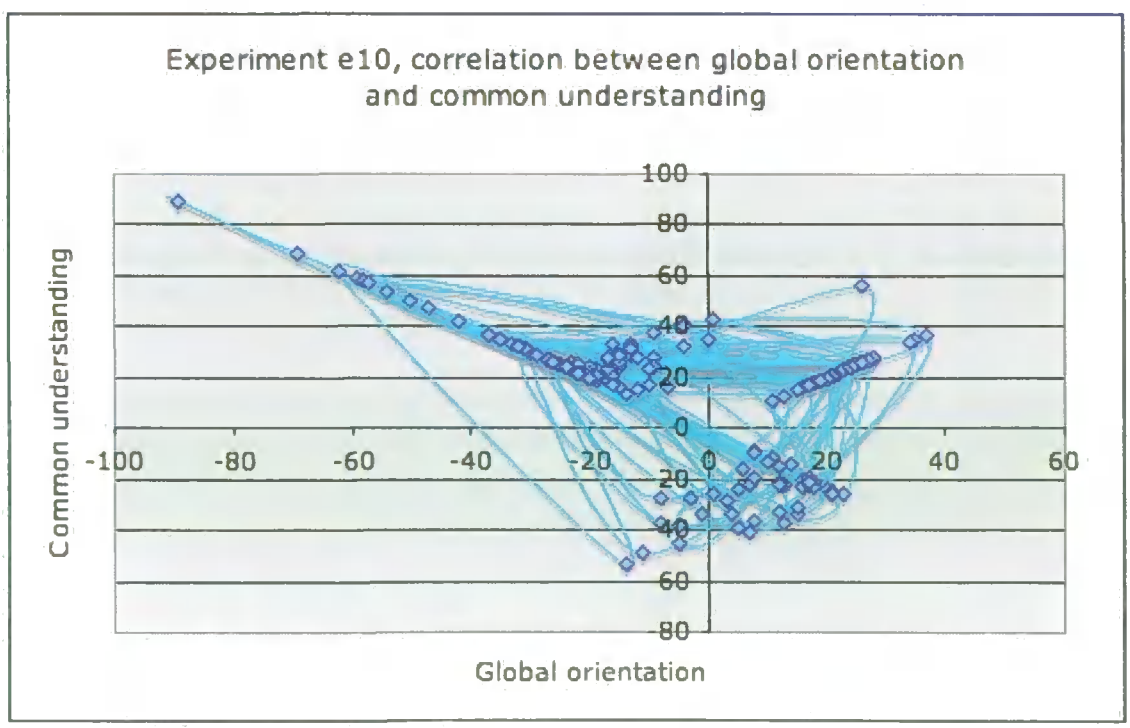


Figure 10.114: Correlation between the drives global orientation and system common understanding in experiment e10: selfish inclination, evolution only (no learning).

Figures 10.111 to 10.114 show a comparative analysis of the results of the four experiments in terms of *correlation* between the drives global orientation and the system common understanding. The respective signals (169 samples in every experiment) were collected in the evolution process, just before breeding the next populations. The drives output signal is between -100 and +100, equivalent to a drive aiming exclusively expression (no attention to the human suggested context) and, at the other end of the scale, total integration (full attention to current context).

System global orientation (see chapter 8, section 8.6) also oscillates between -100 and +100 – implying a scale from total conflict to complete agreement during man-machine interaction.

Following the language of complex dynamical systems (Thomson and Stewart 1986), the figures above may be interpreted as *phase portraits*. The output behaviour of the system is reflected in the topological structure of the phase space of dynamical trajectories.

The strength and the direction of the correlations between the two variables are visualised as scatter-plots. Perfect correlations show up as straight lines. Whatever the system settings for inclination and learning, all experiments reveal a strong correlation between negative global orientations; i.e., system prefers expression, and positive values for system common understanding; i.e., system manages to evolve an interaction regime characterised by agreement rather than conflict.

With the exception of figure 10.112, all figures exhibit a strong positive correlation between agreement and expression. More precisely, human-machine agreement suggests itself given oscillatory behaviour in the dimension of system global orientation. In addition, various data points cluster in irregular places. In

particular, there seems to be no correlation given a system orientation of integration and negative common understanding; i.e., a situation of conflict. This observation proves that Oscar has more trouble to develop an attitude of integration than an attitude of expression. Incidentally, figure 10.112 shows uncorrelated data given conflict and integration. This figure, documenting an experiment with selfish inclination and learning enabled, offers the most explicit phase portrait; just two behavioural regimes. Firstly, nearly perfect correlation given positive common understanding (agreement) and negative global orientation (system prefers expression). Secondly, we observe practically no correlation at all given negative common understanding (conflict) and positive global orientation (system prefers integration). In other words, the system oscillates between, on the one hand, behaviour based on relatively linear relationships and, on the other hand, quite chaotic behaviour.

However, the mutual divergence between these figures remains quite subtle. A social inclination entails the strongest oscillations between both system orientations. In addition the phase portraits in figures 10.111 and 10.114 are virtually identical. Also it proves impossible to develop a clear idea of the impact of learning from the interpretation of the figures above.

10.5 Experiment e7: Analysis of a Limited Interaction Context

A short excerpt from the two MIDI files documenting respectively the activity of man and machine in experiment e7 is analysed in detail. The agents activity was taken as a point of departure; the MIDI file (8 individual tracks, one track per agent) was dissected in terms of pitch, velocity, duration and rhythm – rhythm refers to the time span of the sounding events *and* the duration of the inter-event rests. In summary:

Total number of samples in pitch, velocity and duration: 6585.

Total number of samples in rhythm: 7082.

To start, a zone of 500 samples is located in a total of 6585 samples – starting at sample 1600 and reading to sample 2100; i.e., start-pointer is at 24.29% of the total dataset and the end-pointer is at 31.89% of the total dataset. As a result, a momentary interaction context of 7.6% of the total data is reflected on.

The global agents dataset as a whole is not visualised because, when shown as a linear list, the behavioural data merge and become meaningless. However, we may examine a few agents in isolation; agents nr. 1 and 3 are taken as examples.

10.5.1 Data Visualisation of Interaction Context

Four data sets are visualised as individual information strata in figures 10.119 to 10.122 and 10.123 to 10.126. Note that the rhythm data carries more samples than duration since the equivalent percentage time span includes active as well as passive (rests) durations.

The MIDI file holding the events produced by the human interactor is addressed next. Analysis of pitch, velocity and duration yields 1613 samples, rhythms yields 2602 samples. Extracting the same 7.6 percent as above, one gets respectively 166 and 242 samples. Information derived from human input is depicted in figures 10.115 to 10.118.

The next seven figures (10.127 to 10.133) address data gathered by the tracer-data collector. A region of 58 samples is taken from a total of 755 samples, again following a start-pointer at 24.29% of the total dataset to the end-pointer at 31.89% of the total dataset. The remaining figures (figure 10.134 to 10.138) display information gathered from the evolution-data collector; i.e., data available at the end of every

learning-cycle and just before genetic breeding takes place. This small dataset of just 14 samples reveals a remarkable amount of information.

The final pages show a 19-segment sequence of a 9-track score in common music notation; the top track documents human input, the next eight document machine-generated music, a single track is associated with a single agent. The excerpt shown covers 36 measures taken from a total of 469 measures, again according to the start and stop pointers as explained above.

The remaining sections are organised as follows: (1) comments are offered on the subject matter of the respective figures which document the source material for the analysis in question, and (2) the musical rendering is analysed (rather than the independent information strata) in an attempt to infer a deeper implication from the observation of the musical events in the 19-page score excerpt.

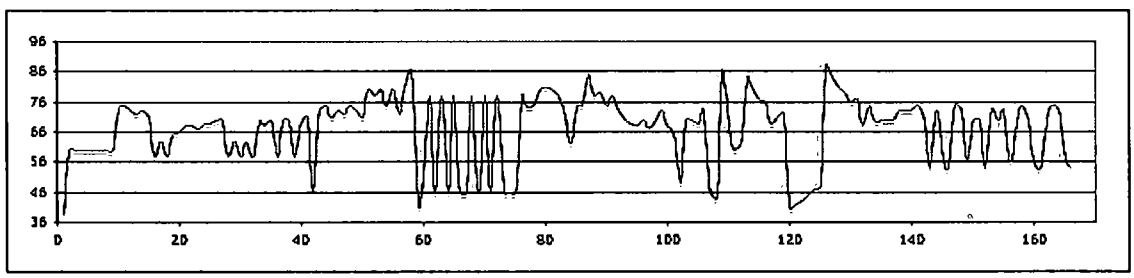


Figure 10.115: Human input, dimension of pitch; 166 samples.

Human input (figure 10.115) is mainly characterized by long sequences of consistently large, alternating positive and negative pitch intervals with occasional leaps. Such behaviour is bound to return a Boolean true for the angular-pitch sensor. It will also lower the quality-level of the data in working memory because of the relatively low diversity of pitch intervals. Other zones reveal arpeggios and recurring patterns with variations.

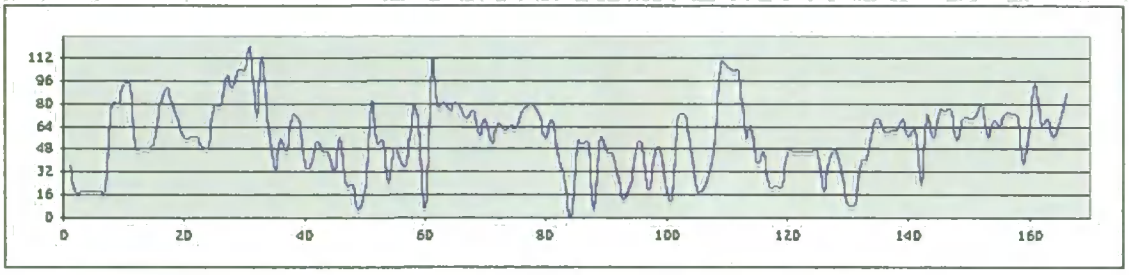


Figure 10.116: Human input, dimension of velocity; 166 samples.

The loudness of the events produced by the human interactor fluctuates in more erratic conduct (figure 10.116). Yet one may spot both unusual peaks as well as areas of more or less stationary input.

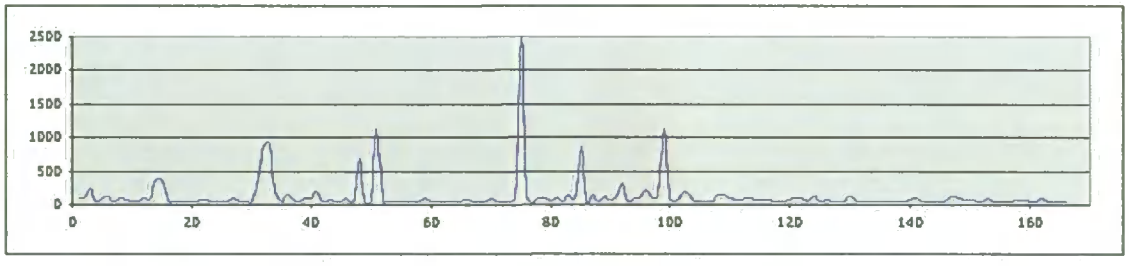


Figure 10.117: Human input, dimension of duration; 166 samples.

Figure 10.117: the human interactor mainly produces short events (duration < 500 milliseconds), with the exception of a few occasional very long events. The 2.5 seconds event (sample 75) happens to coincide with a repetition of pitch (MIDI pitch 74) signalling the end of a history of steadily oscillating pitches. This situation is sure to trigger the reflex sensor (see chapter 4, section 4.4.1.2).

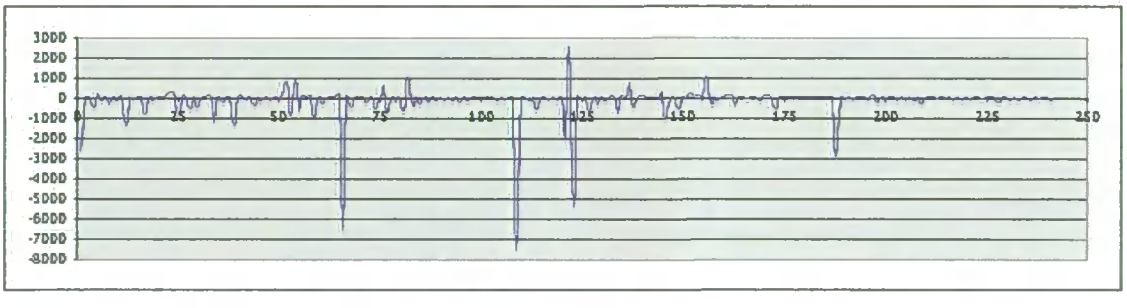


Figure 10.118: Human input, dimension of rhythm; 242 samples.

When considering the rhythm data in figure 10.118; positive values document active event durations, negative values reflect inter-event rests. This figure reveals mainly short durations and rests (a relatively fast input sequence) in addition to a few extended periods of rest. Incidentally, some exceptional data levels in the respective dimensions happen to coincide: consider the transition of a long event (sample 122, the highest peaking duration of 2500 milliseconds) to the occurrence of a very long silence (sample 123, a rest of 5360 milliseconds). In addition, this situation also matches the sudden occurrence of a zero pitch interval as shown in figure 10.115. Again, without doubt, such a dramatic change in context will trigger a considerable family of sensors.

As an example, agents 1 and 3 are visualised next.

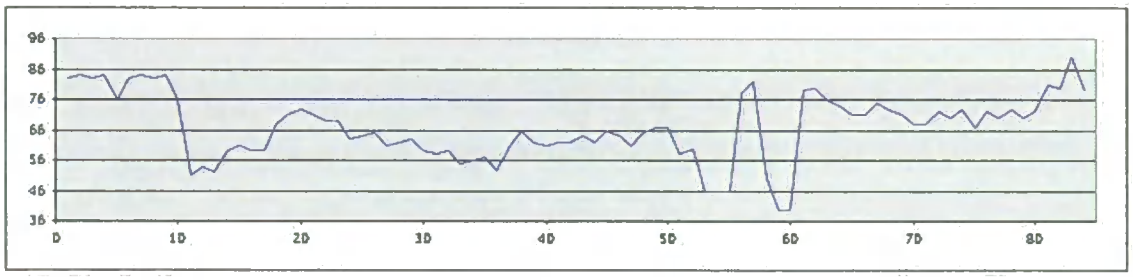


Figure 10.119: Agent 1, dimension of pitch; 84 samples.

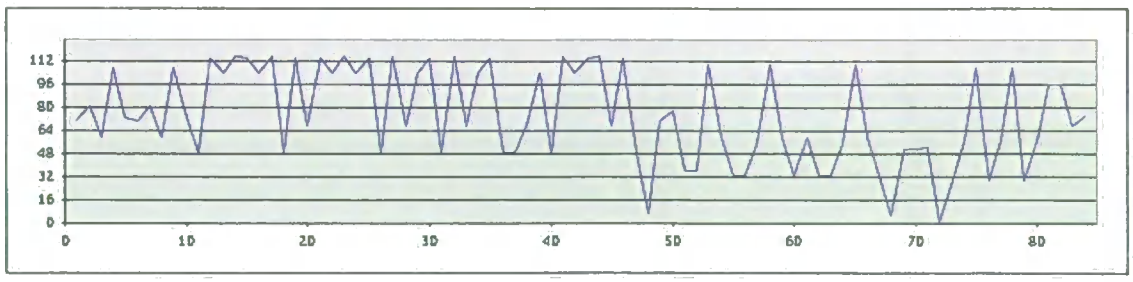


Figure 10.120: Agent 1, dimension of velocity; 84 samples.

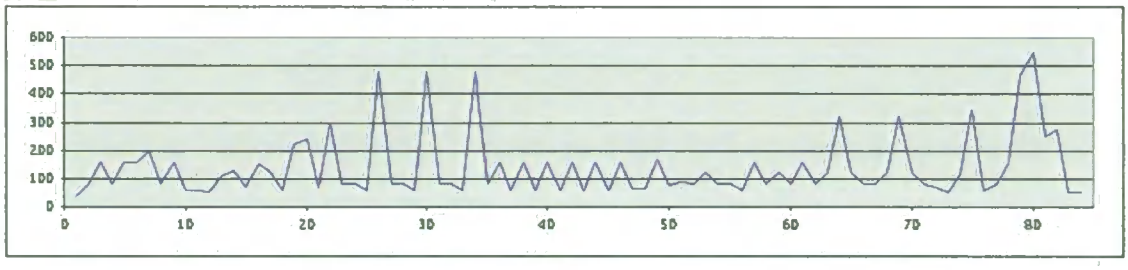


Figure 10.121: Agent 1, dimension of duration; 84 samples.

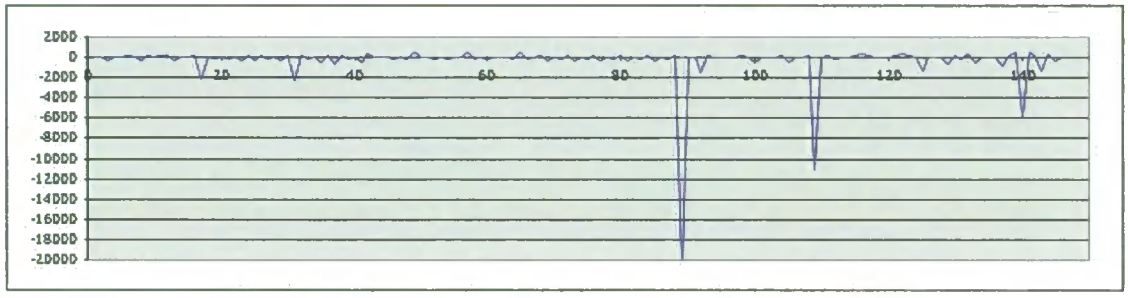


Figure 10.122: Agent 1, dimension of rhythm; 146 samples.

Figure 10.119 shows the dimension of pitch of agent 1. While this was *not* anticipated, a natural interpretation of the pitch contour is to attribute qualities of Brownian motion; occasional large, intermittingly positive and negative intervals alternate with zones of more coherent nearly periodic oscillations. In addition, small melodic figures that gradually move up or down are easily detected. As such, remarkable idiosyncratic pitch behaviour is revealed.

Figure 10.120 displays the coordinated activity of the agents-based orchestration algorithm in the dimension of velocity. Pivotal changes in the dimensions of pitch and velocity sometimes clearly coincide, at other times important changes are not synchronized. This reflects the variable pressures in coordinated action exercised by the *principle of multiple influences* (see chapter 6, section 6.4).

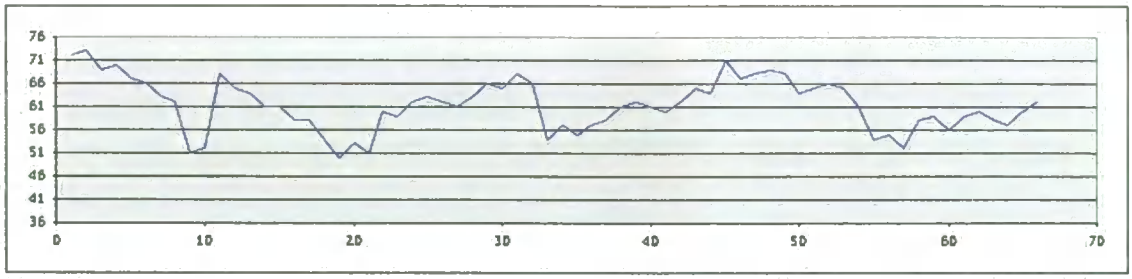


Figure 10.123: Agent 3, dimension of pitch; 66 samples

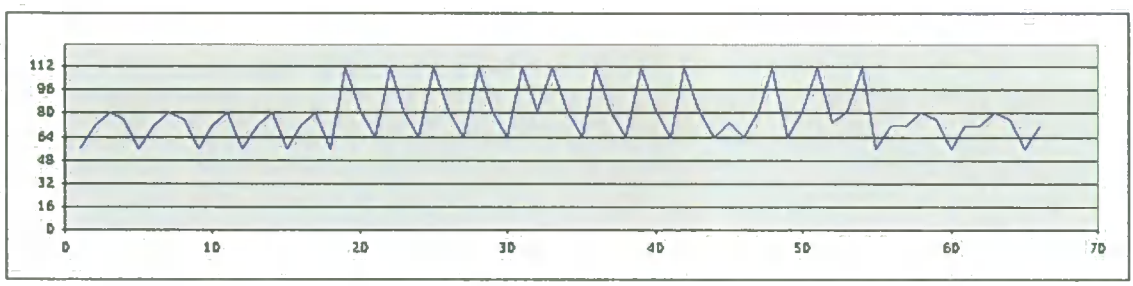


Figure 10.124: Agent 3, dimension of velocity; 66 samples

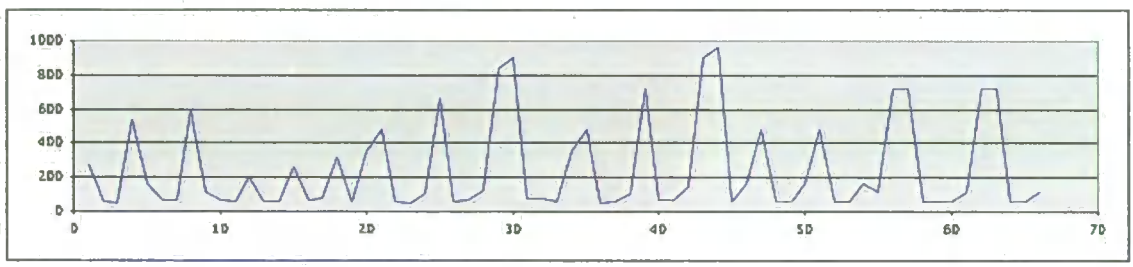


Figure 10.125: Agent 3, dimension of duration; 66 samples

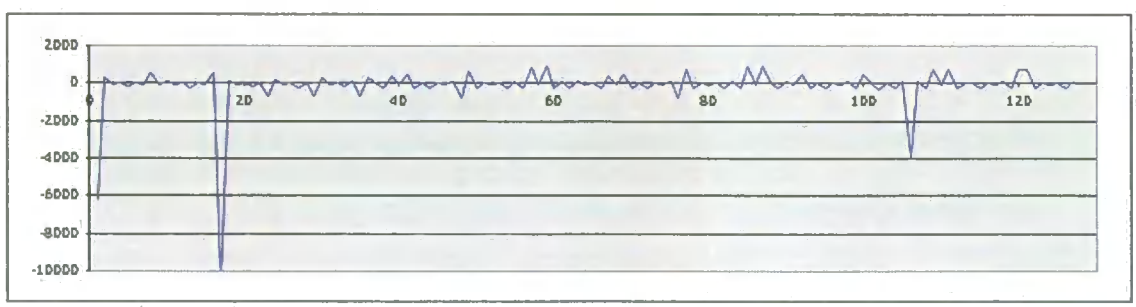


Figure 10.126: Agent 3, dimension of rhythm; 127 samples

Figure 10.121 shows how the duration of machine generated events cluster in specific value zones. The episodic nature of the data reflects the changes in compound-functions. Eventual continuity in the data may follow from the similarity between consecutively operational compound-functions, from the interactions of

agents equipped with similar duration data or from continuity introduced by an agent acting in isolation; an agent only relying on its own private data set.

Also figure 10.122 shows a great deal of regularity in the articulation of rests and active durations, values oscillate around zero with a relatively high degree of periodicity. Such resulting regularity is typically generated by the repeated application of short data lists; agents borrow small excerpts of data from working memory (the human performer or from neighbouring agents, if any) again according to the principle of multiple influences. Note how recognized behavioural regimes in some dimension (pitch, velocity and duration) are clearly delineated by significant contextual changes in another dimension – the peaking negative values in the dimension of rhythm.

Let us consider the output of agent nr. 3. Agent 3 produces less output than agent 1 but the data reveals a much higher degree of periodicity. This implies that the source data from which agent 3 generates its content remains more or less the same over the complete duration of the short interaction excerpt documented here. Also, the interference of the individual source patterns is clearly observed because the patterns for pitch, velocity and duration are of slightly different length.

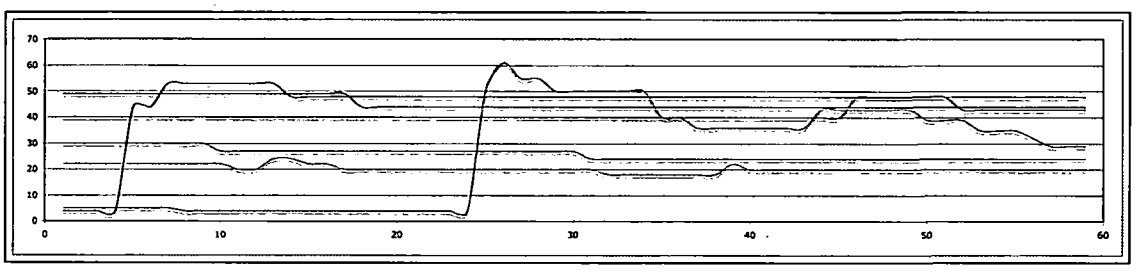


Figure 10.127:Agents input-pressure, single dimension (human), 58 samples. For clarity, only the data reflecting the six agents (out of 8) that actually engage in interaction are visualised.

Figure 10.127 shows how an agent swiftly adapts to human input. The input-deliberation algorithm manipulates the respective sensitivities according to external human pressures and according to changes in proximity of neighbouring agents (see chapter 6, section 6.5.5). Increased sensitivity in one agent (red curve) seems to coincide with a major peak in velocity – as noticed in figure 10.116.

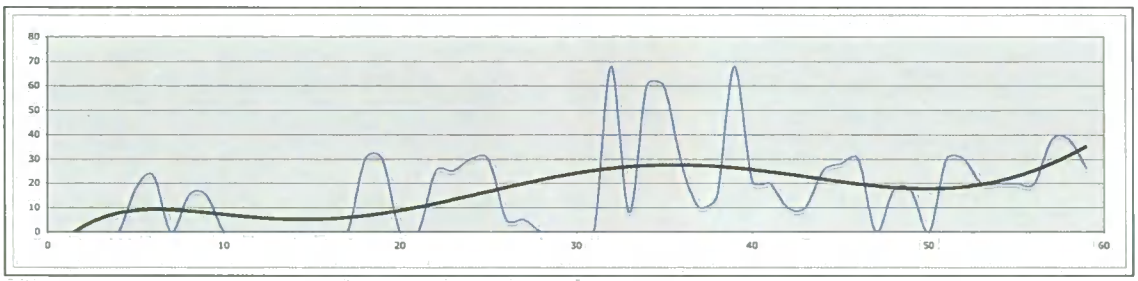


Figure 10.128: Human-machine total similarity), 58 samples.

A moderate increase in man-machine similarity characterises figure 10.128.

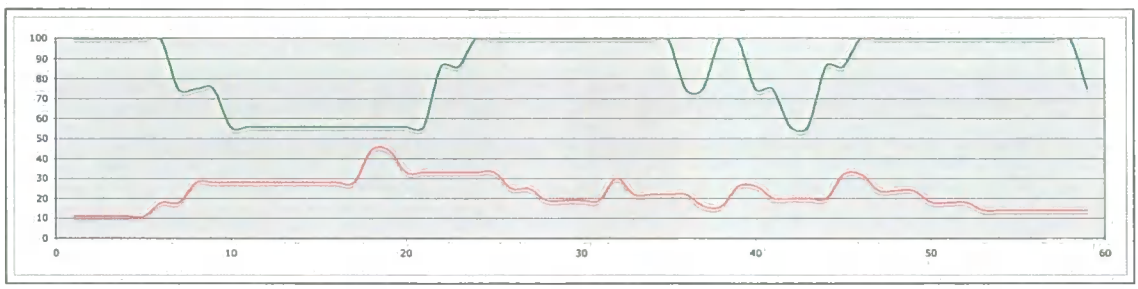


Figure 10.129: Human (green curve) vs. machine (red curve) agreement, 58 samples.

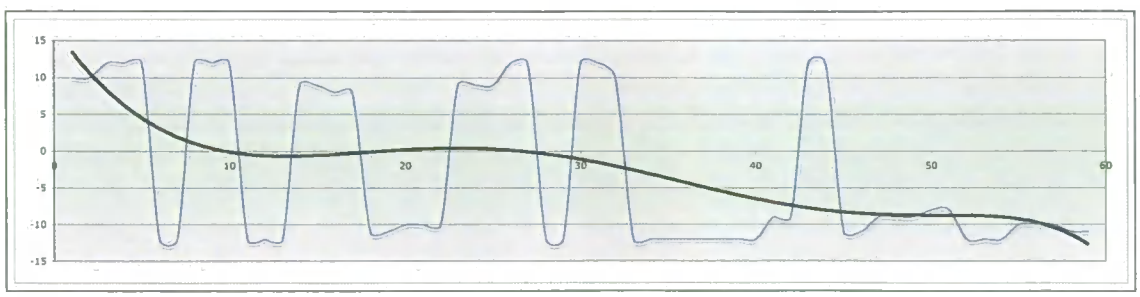


Figure 10.130: Drives global orientation, 58 samples.

The drives global orientation profile evolves from slightly positive values to foremost negative values, thus Oscar develops a preference for *expression* in this data block.

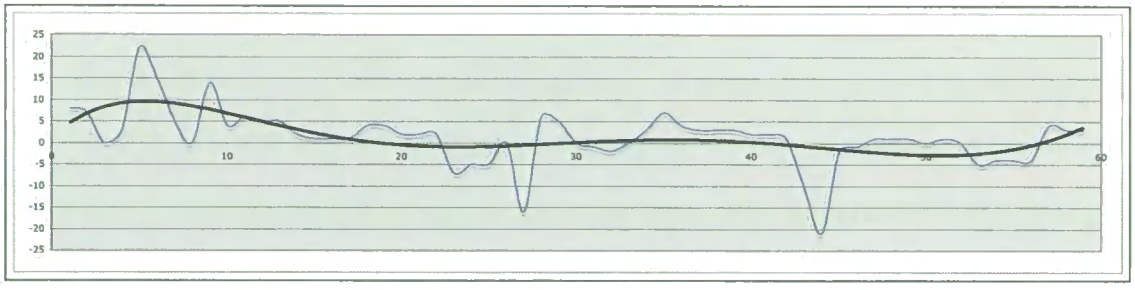


Figure 10.131: Patcher global orientation, 58 samples.

The patcher data in figure 10.131 first develops positive values (a wish to integrate) and then settles on an oscillating regime with occasional negative (expressive) peaks.

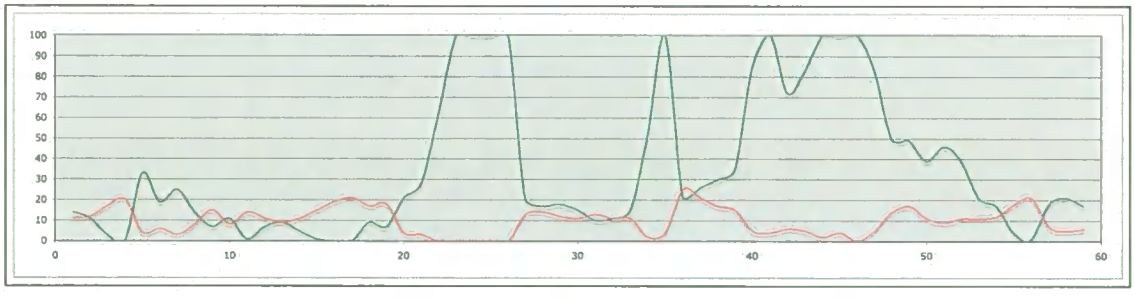


Figure 10.132: Exploitation (green curve) vs. exploration (red curve), 58 samples.

Figure 10.132 documents the competing levels of exploration and exploitation as they adapt according to changes in human-responsiveness (see chapter 4, section 4.5.8). The momentary values are brought into play when *real-time* learning is active.

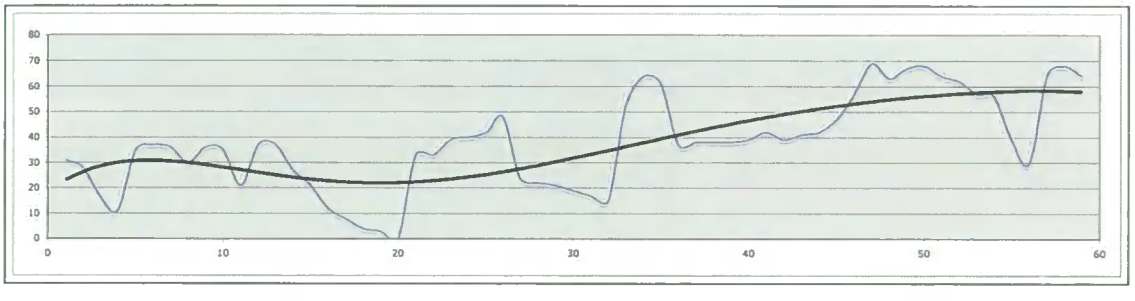


Figure 10.133: Human-responsiveness, 58 samples.

When comparing the incremental nature of human-responsiveness in figure 10.133 to the data in figure 10.132 one observes the following: while exploitation level has developed a first peak around sample 25, human-responsiveness must gain

considerable momentum before it succeeds in pushing the exploitation level down and the exploration level slightly up. A correlation is certainly observed. However the influence on exploration/exploitation levels is delayed and obscured by the non-linear couplings inside the system's networked architecture.

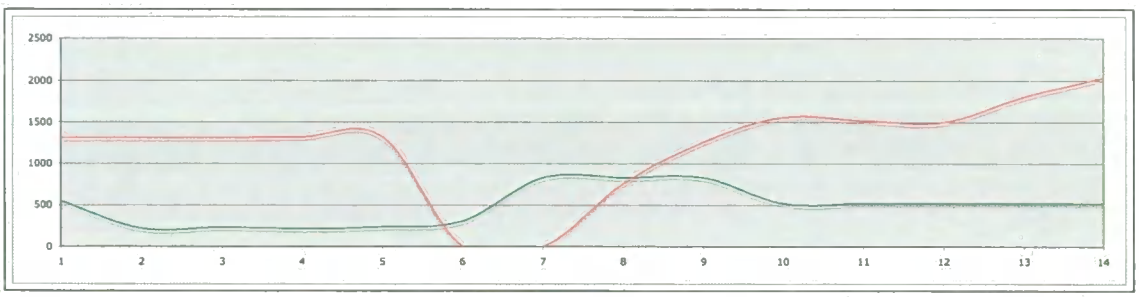


Figure 10.134: Compound-function-pool levels: integration (red), expression (green), 14 samples.

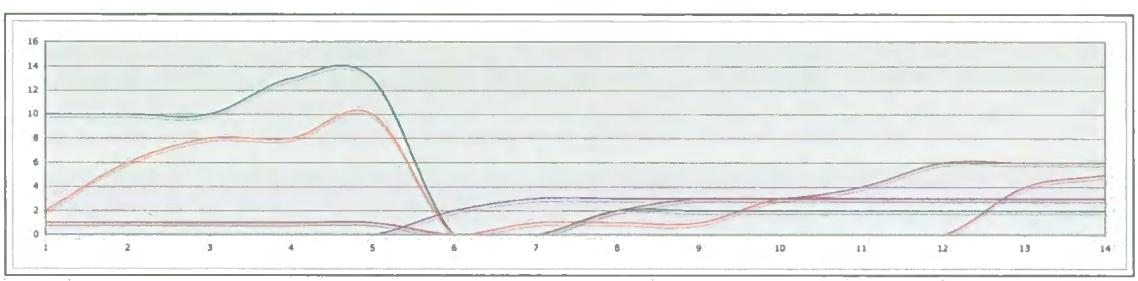


Figure 10.135: Compound-function-pool, function application frequency, 14 samples

Figure 10.134 and 10.135 documents the effect of evolution in the compound-function pool in a limited span of just 14 learning sections, the onset of a fresh population by genetic breeding is observed at the start of phase number 6.

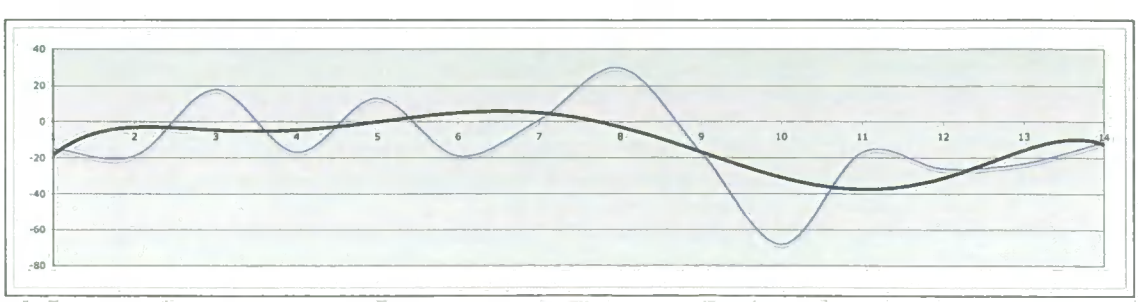


Figure 10.136: System-global-orientation, 14 samples.

The system-global-orientation level in figure 10.136 initially oscillates with a cycle of approximately one sample and progressively slows down its frequency to a cycle of about three samples. The net effect is a transition from near zero values to mildly peaking negative samples; i.e., the inception of a system oriented towards expressive behaviour.

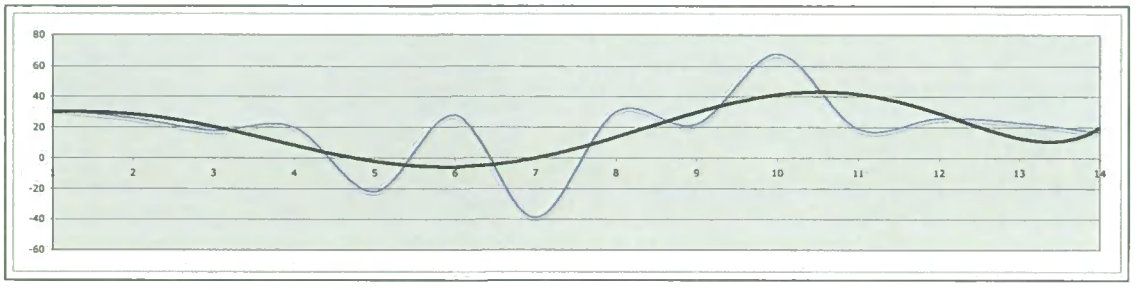


Figure 10.137: System common understanding, 14 samples.

Figure 10.137: System common understanding advances from agreement (30%) at the beginning to minor conflict at phase 6 (5%) to peaking agreement (40%) at phase 11. The peaking is explained by the global trend towards negative values in patch *and* drive (see respectively figures 10.130 and 10.131).

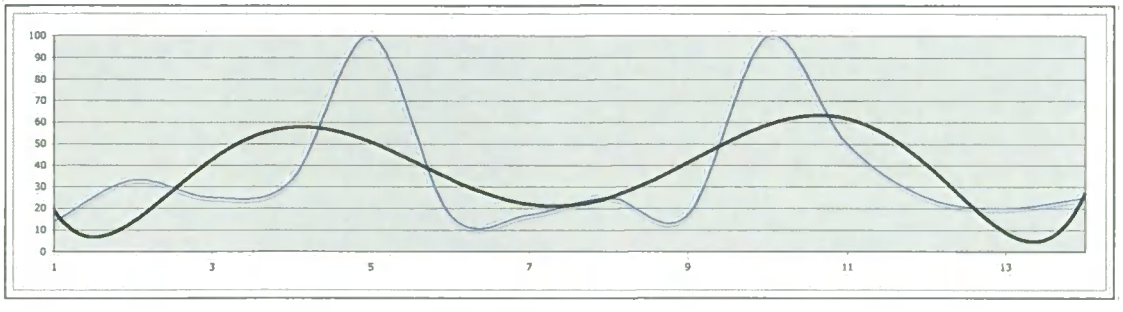
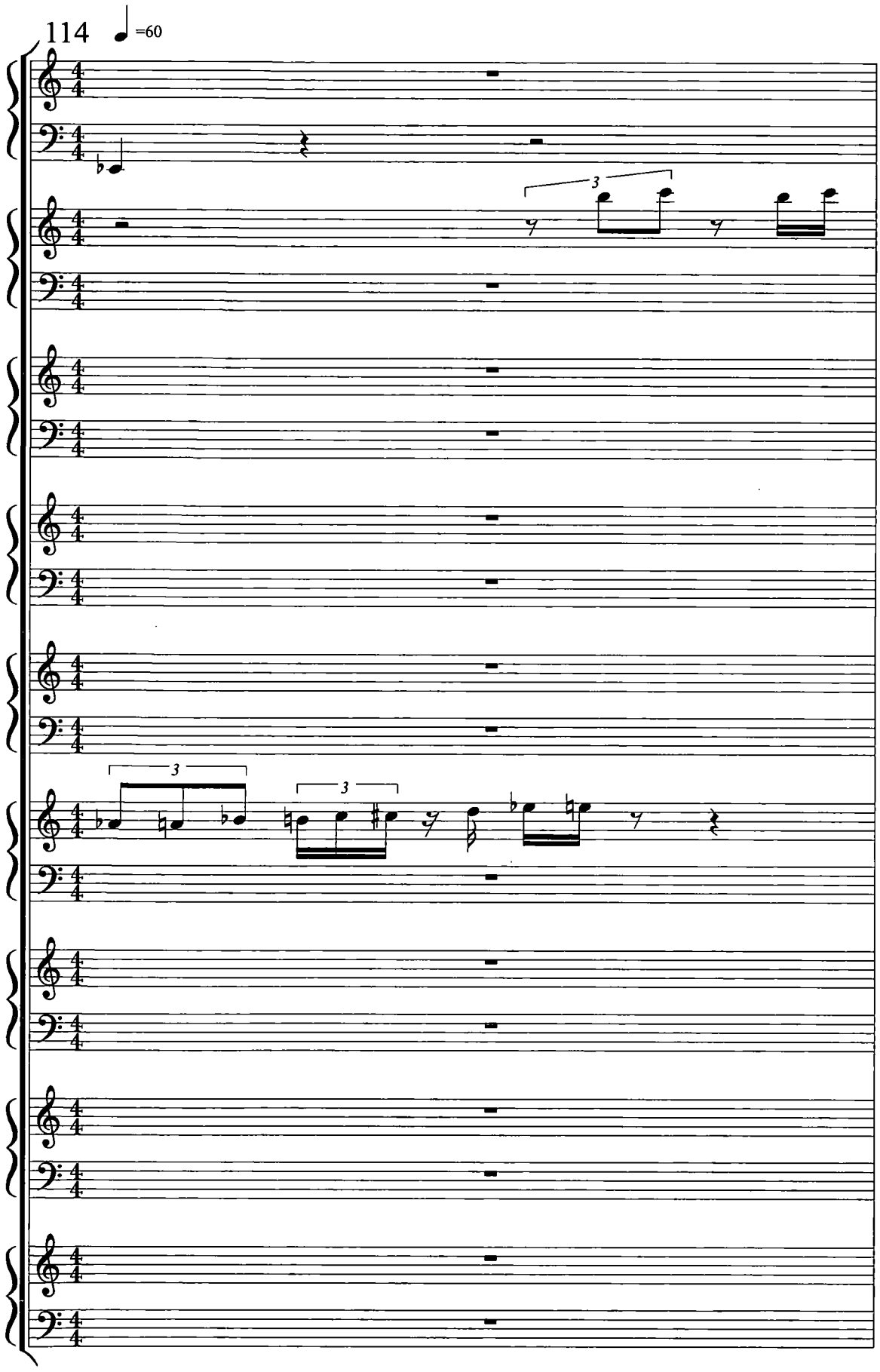
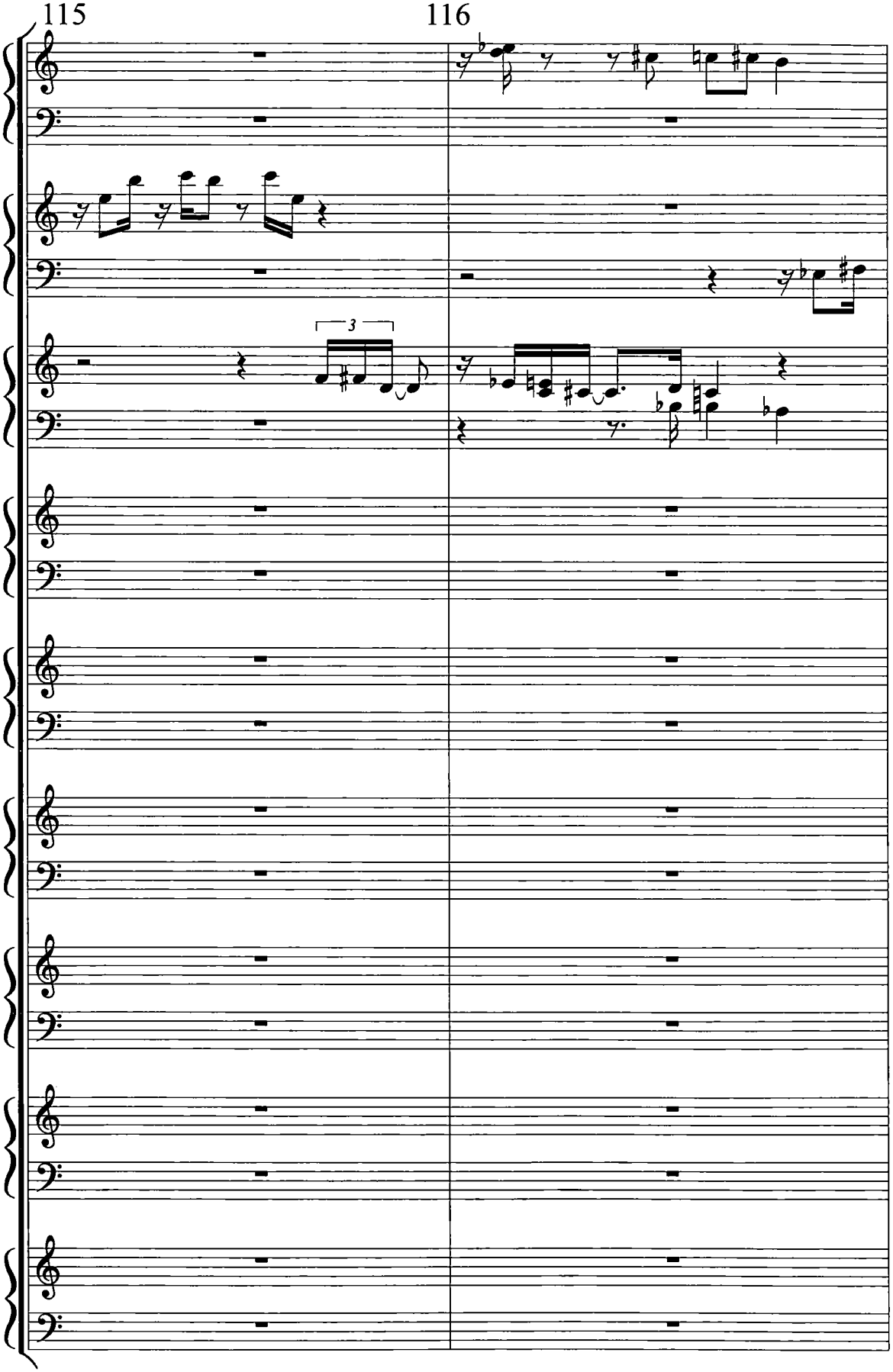


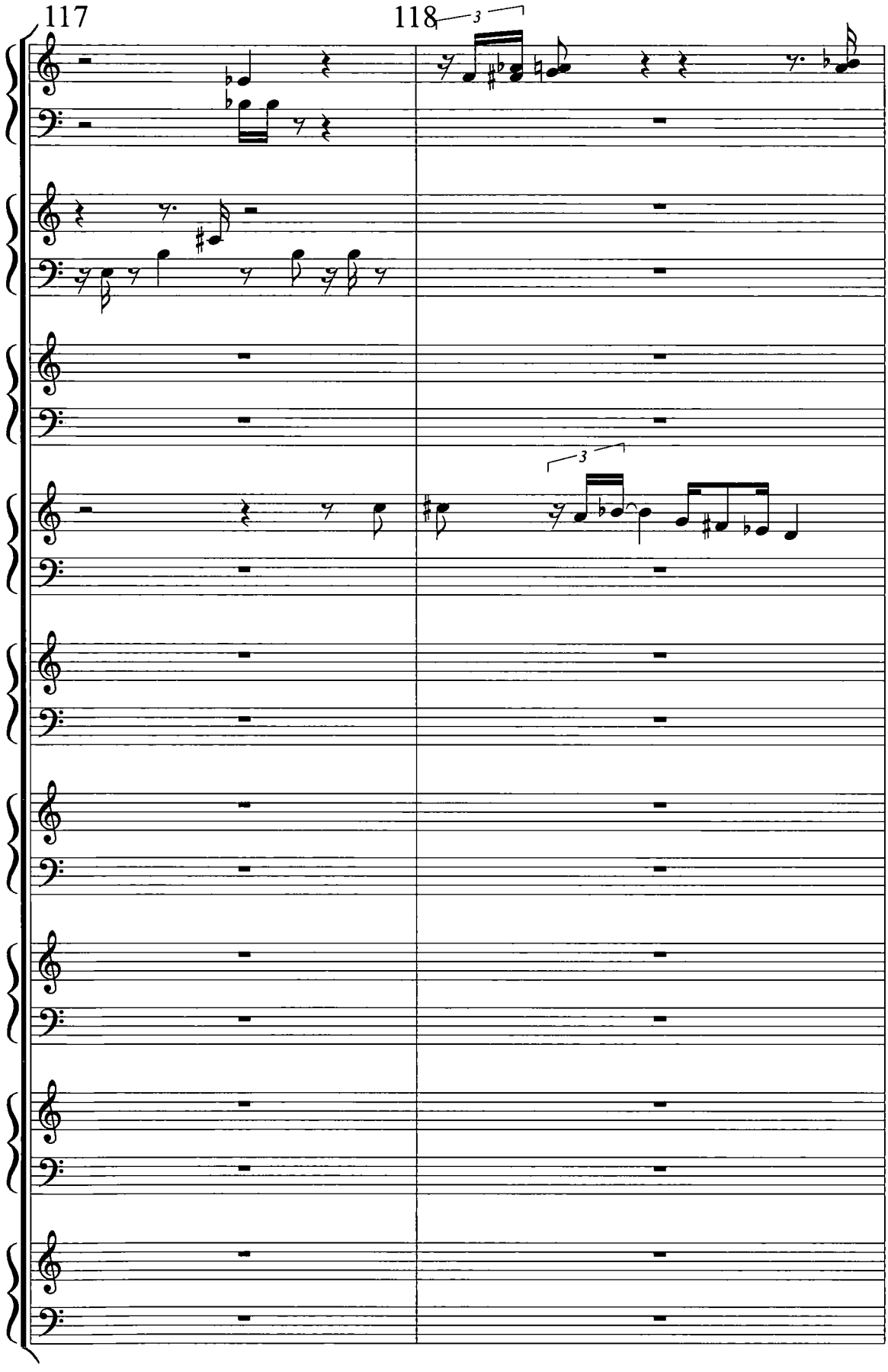
Figure 10.138: Percentage unique agents' ID's, 14 samples.

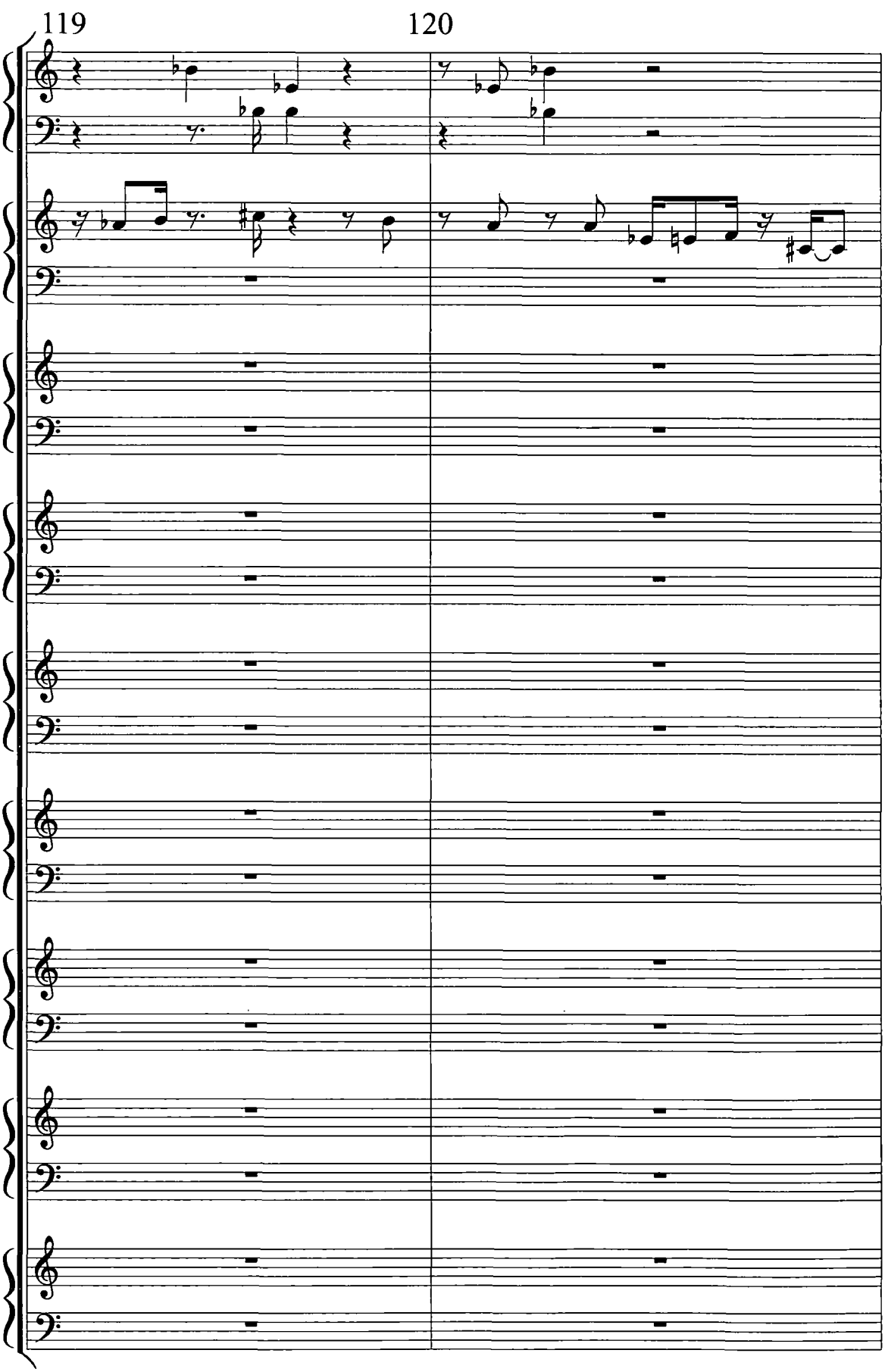
The percentage of unique agents' ID's is proportional to the specificity of the compound-function pool selection procedure, a procedure that picks certain functions from the current population and attributes a single function to every agent in the players agency. The orientation of the current drive (integration or expression)

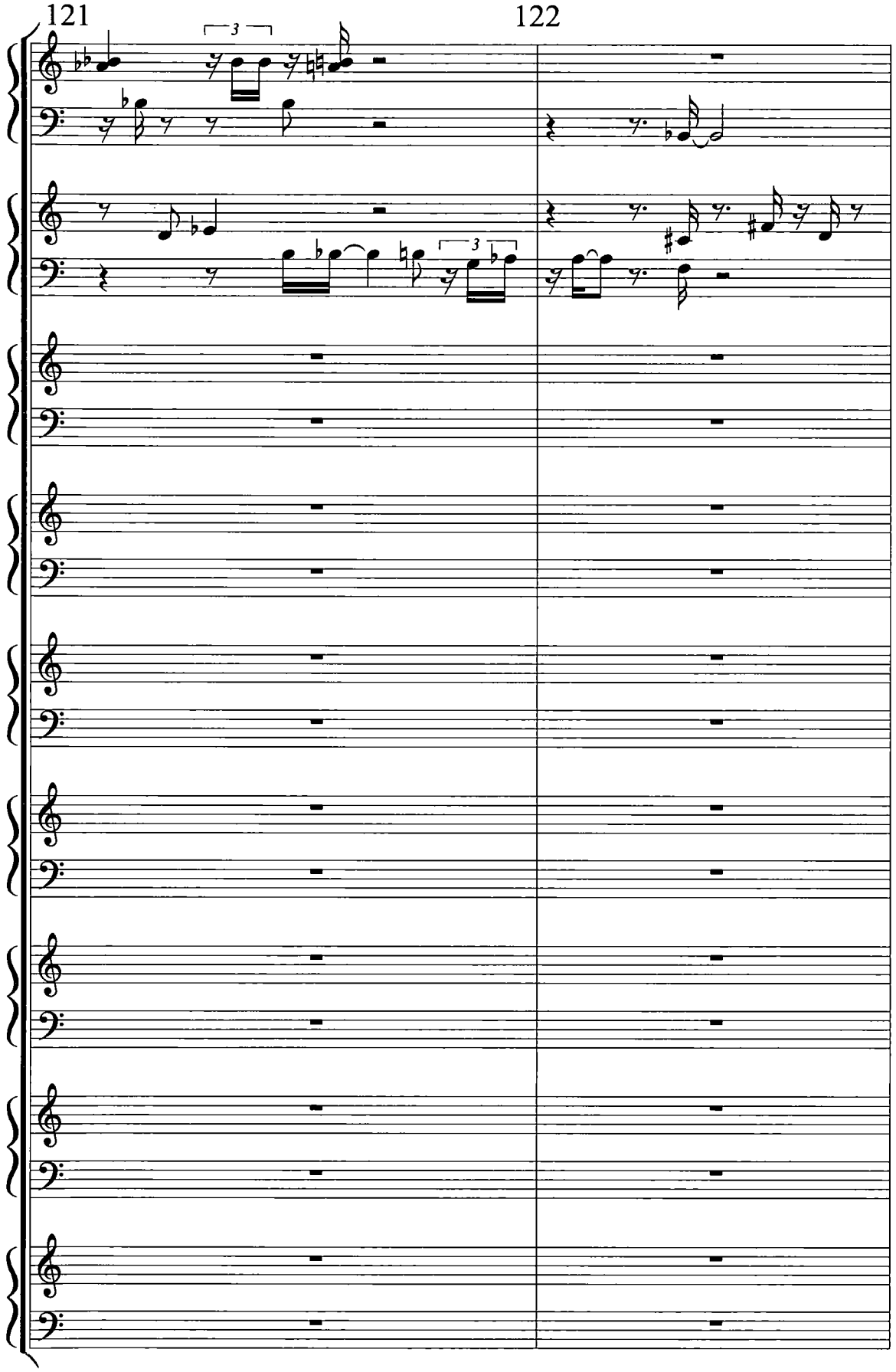
determines what kind of CF will be considered for selection; i.e., only the functions that have developed positive fitness of that drives' current orientation. Thus, a list of candidate CF is generated and sorted by fitness given the current purpose; i.e., help to integrate or offer expression. In case few candidates exist, specificity is high, when only one CF provides functionality for the present objective, specificity equals 100% (see chapter 9). Specificity tops 100% at phase 5 and phase 10.

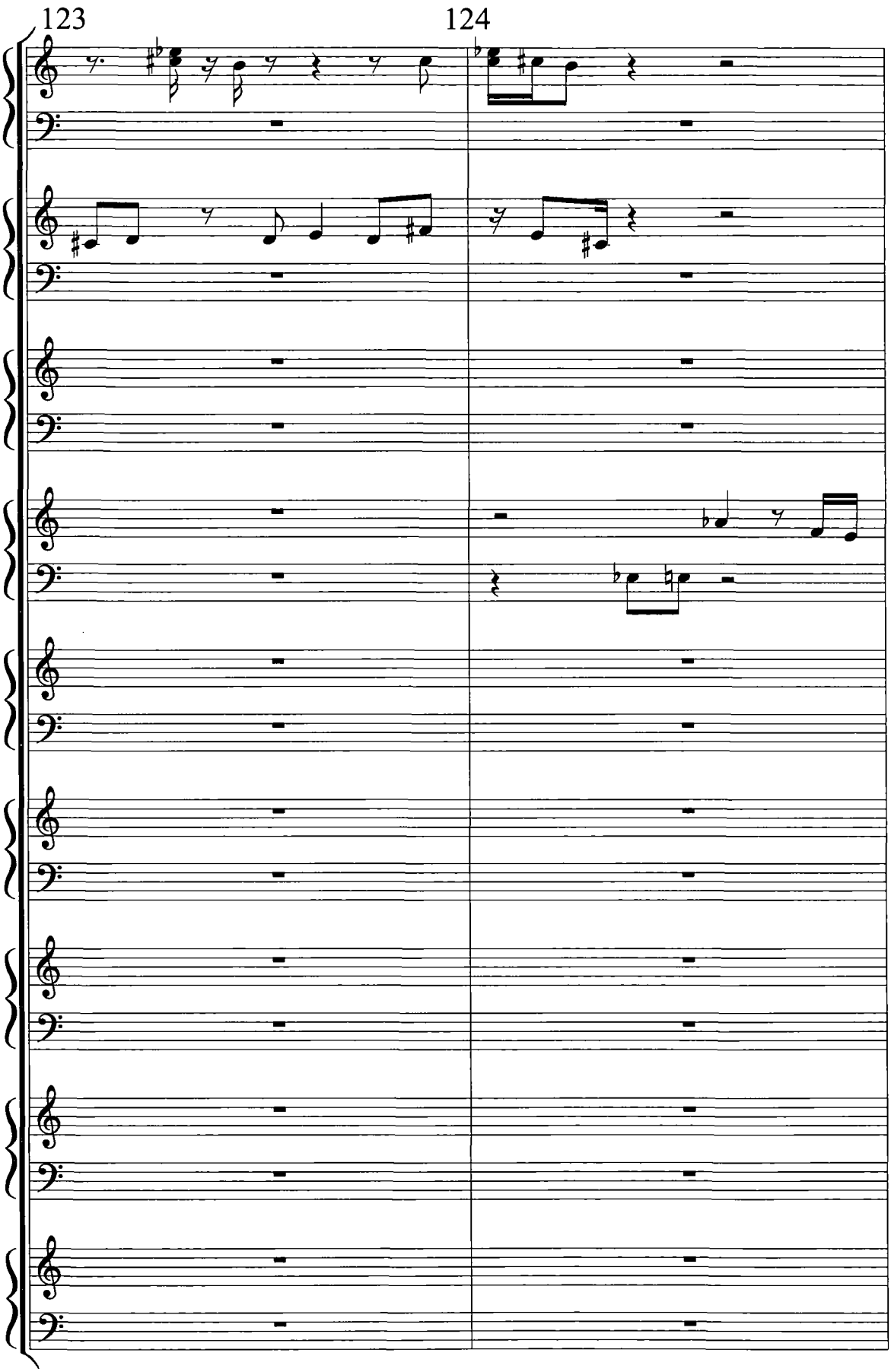

114
=60

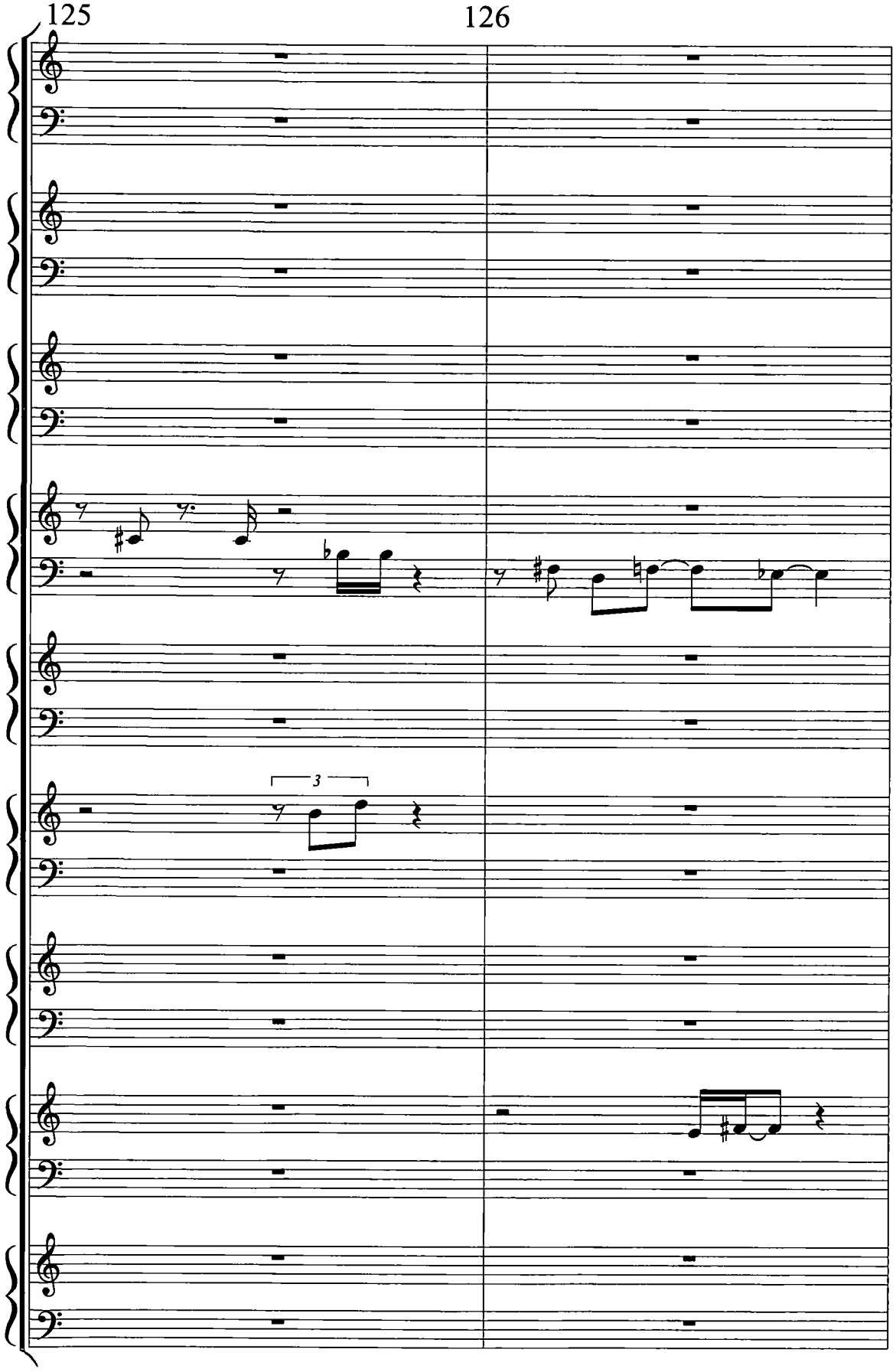

115
116
3

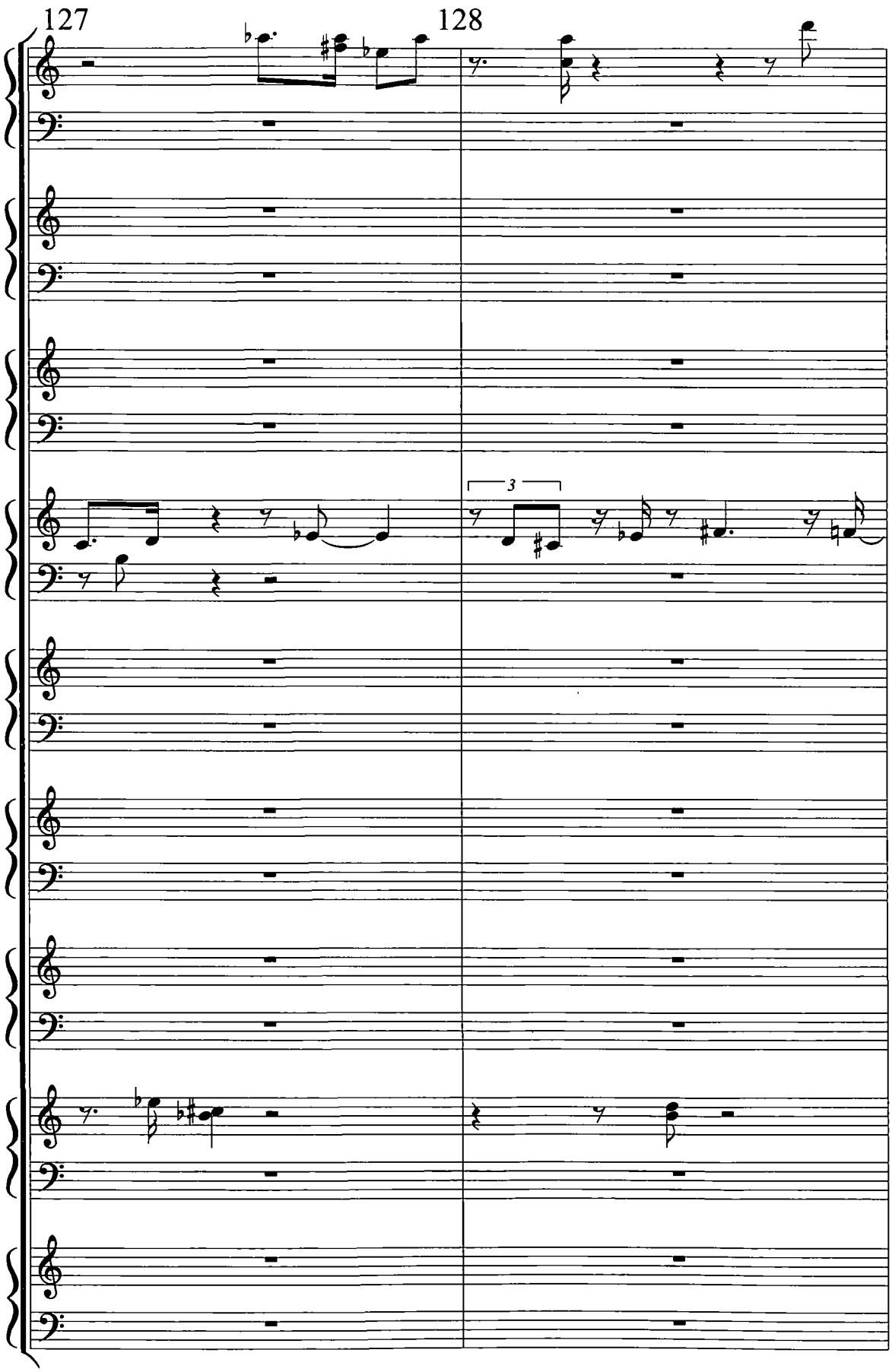

117
118
3
3

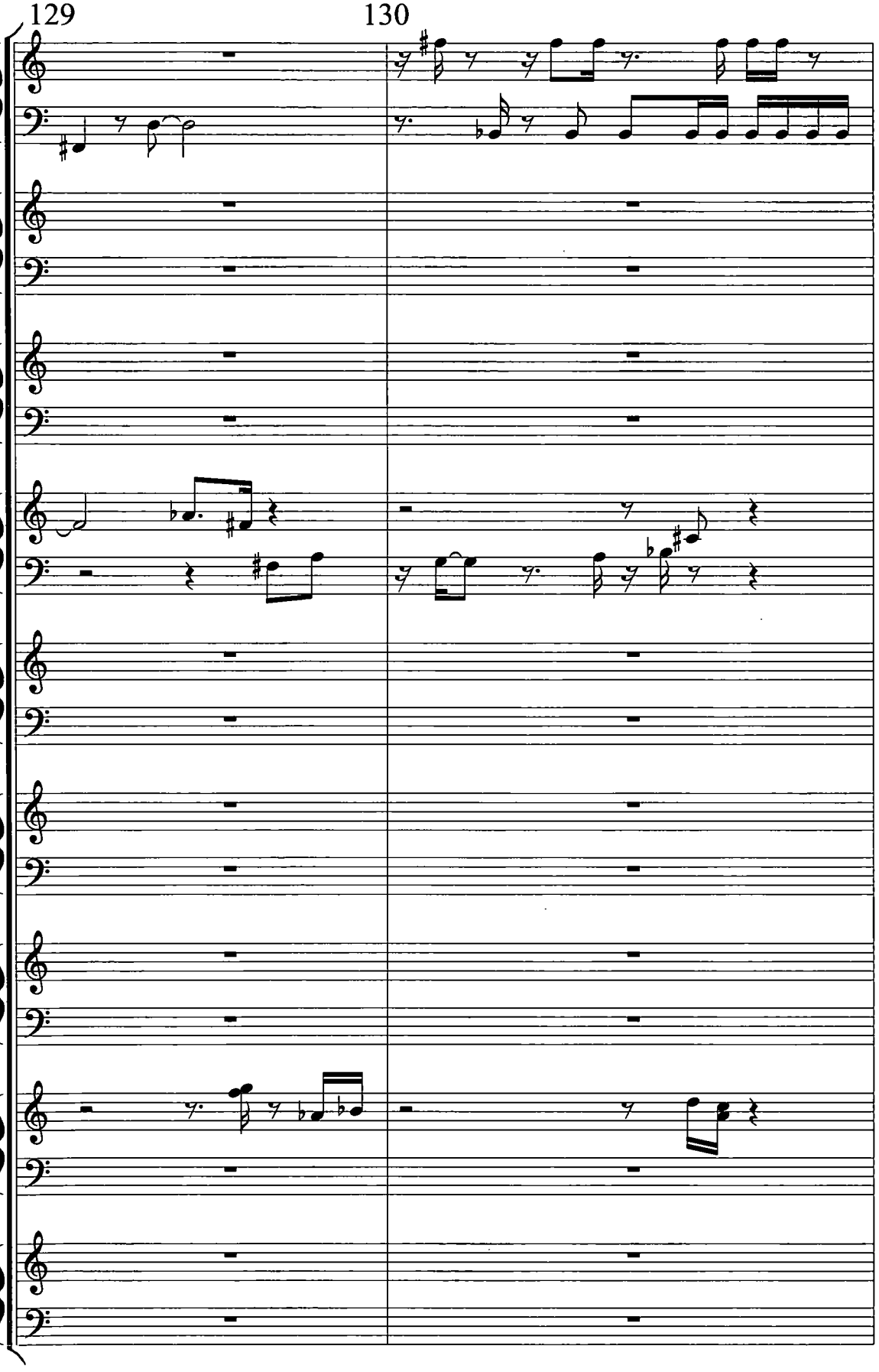

119
120

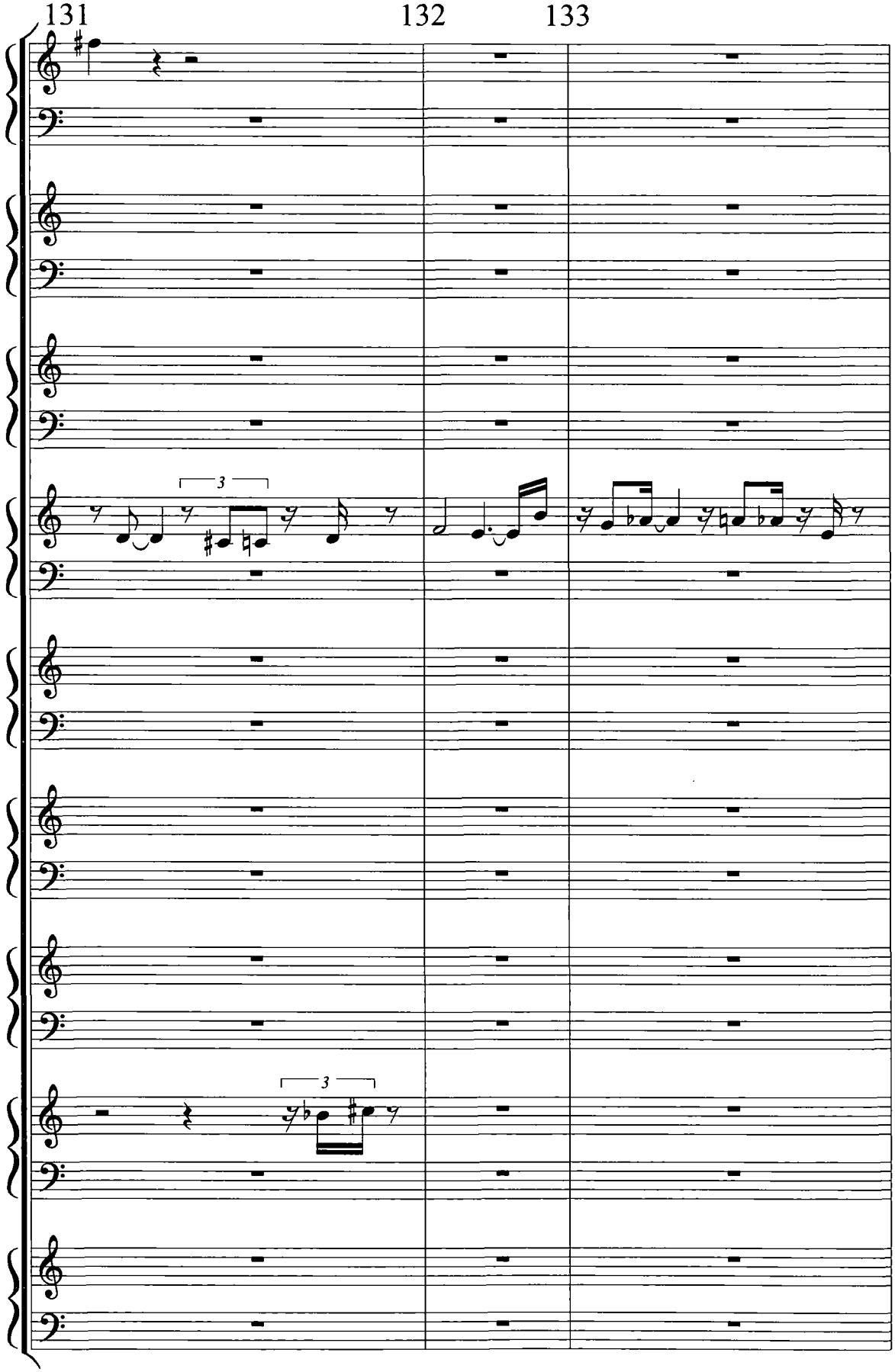

121
122

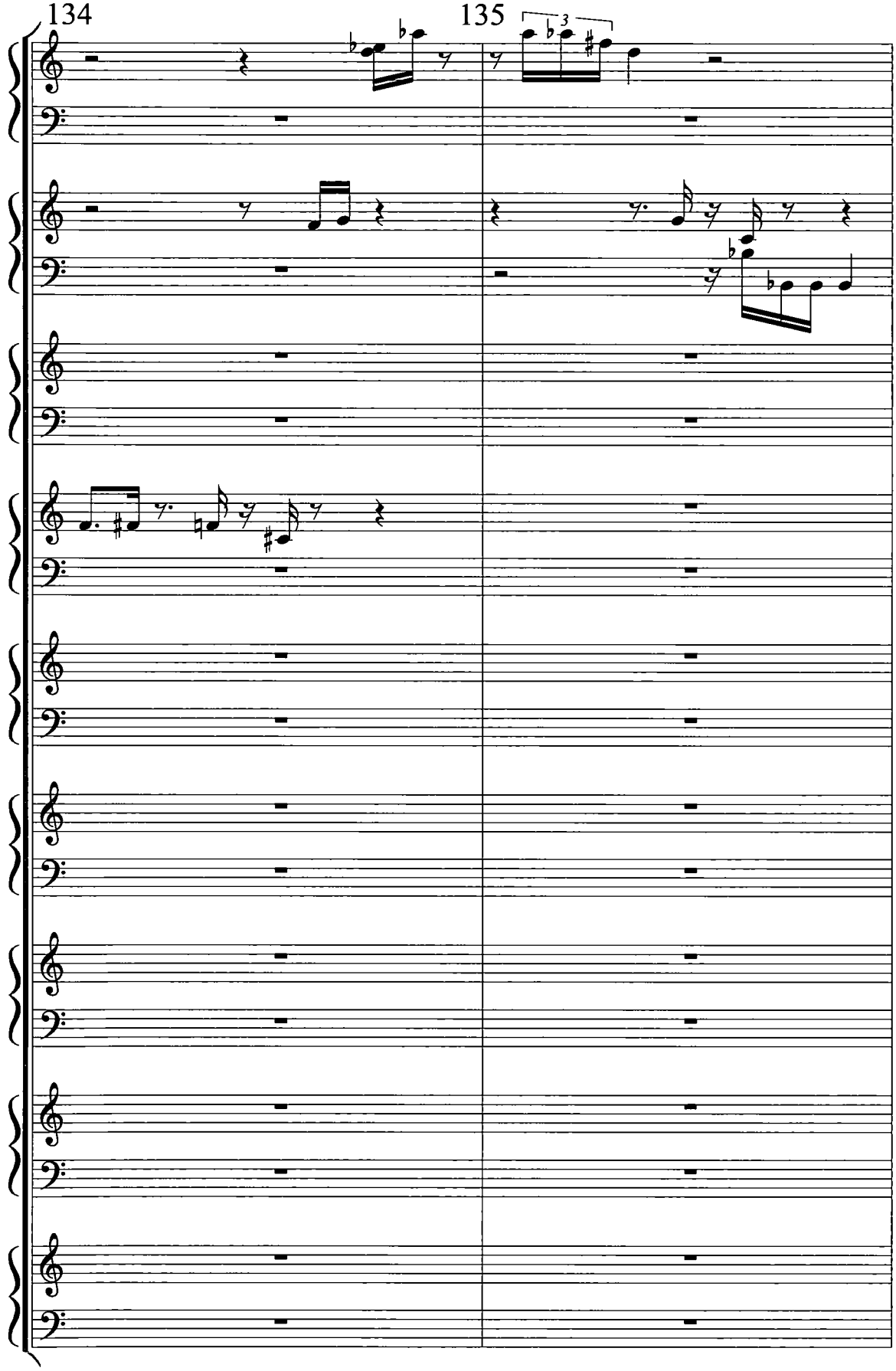

123
124

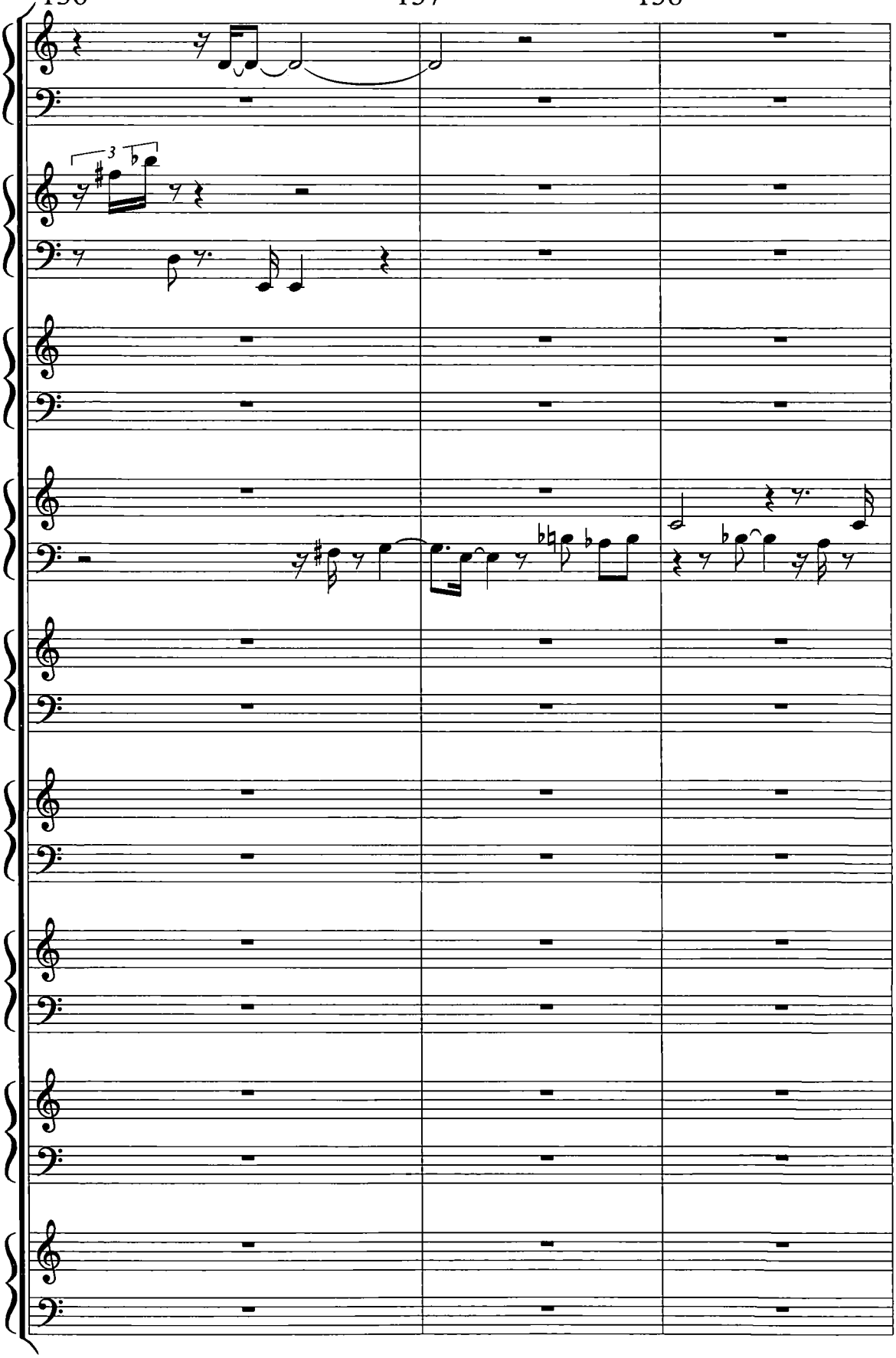

125
126
3

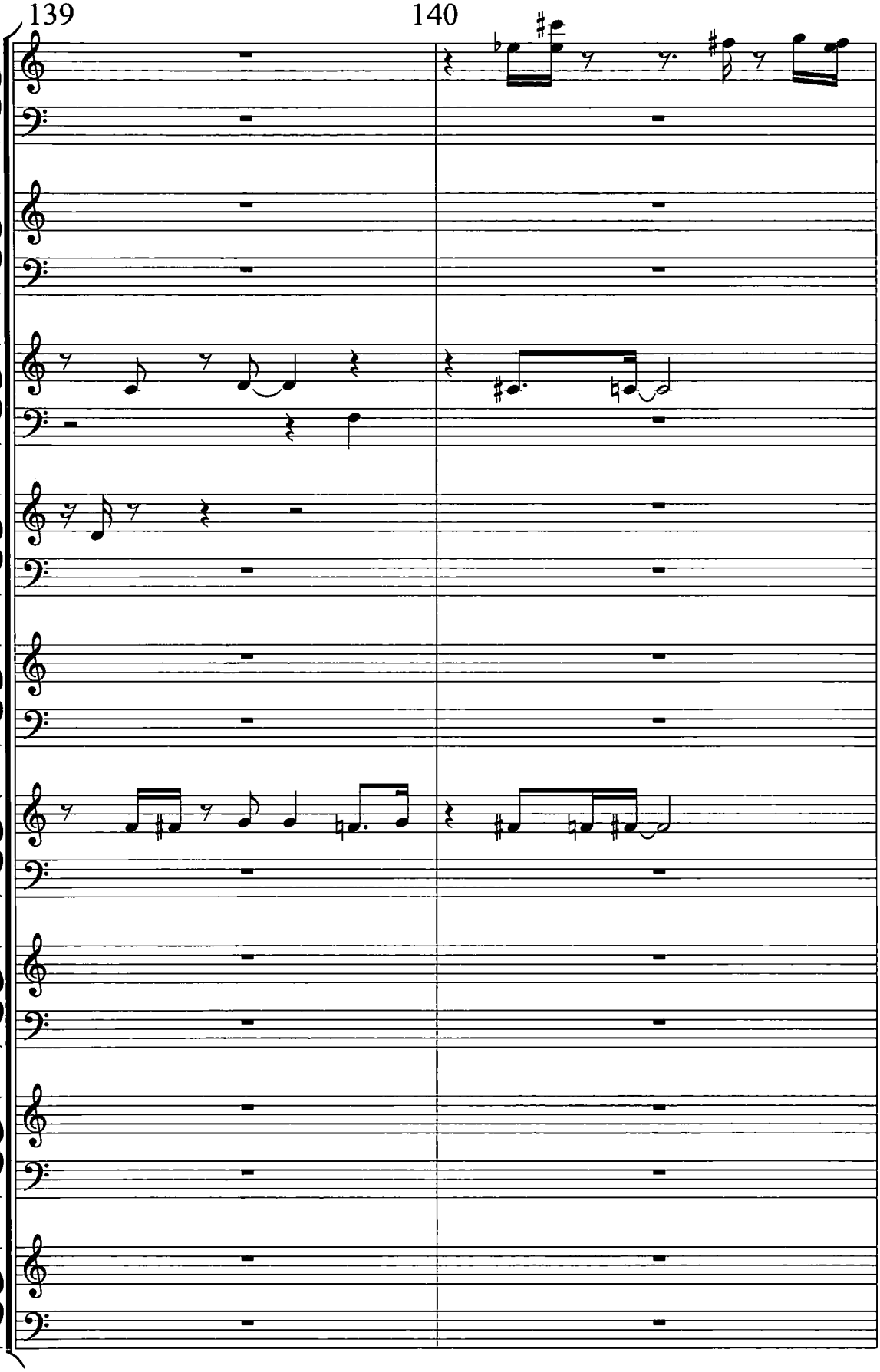

127
128
3

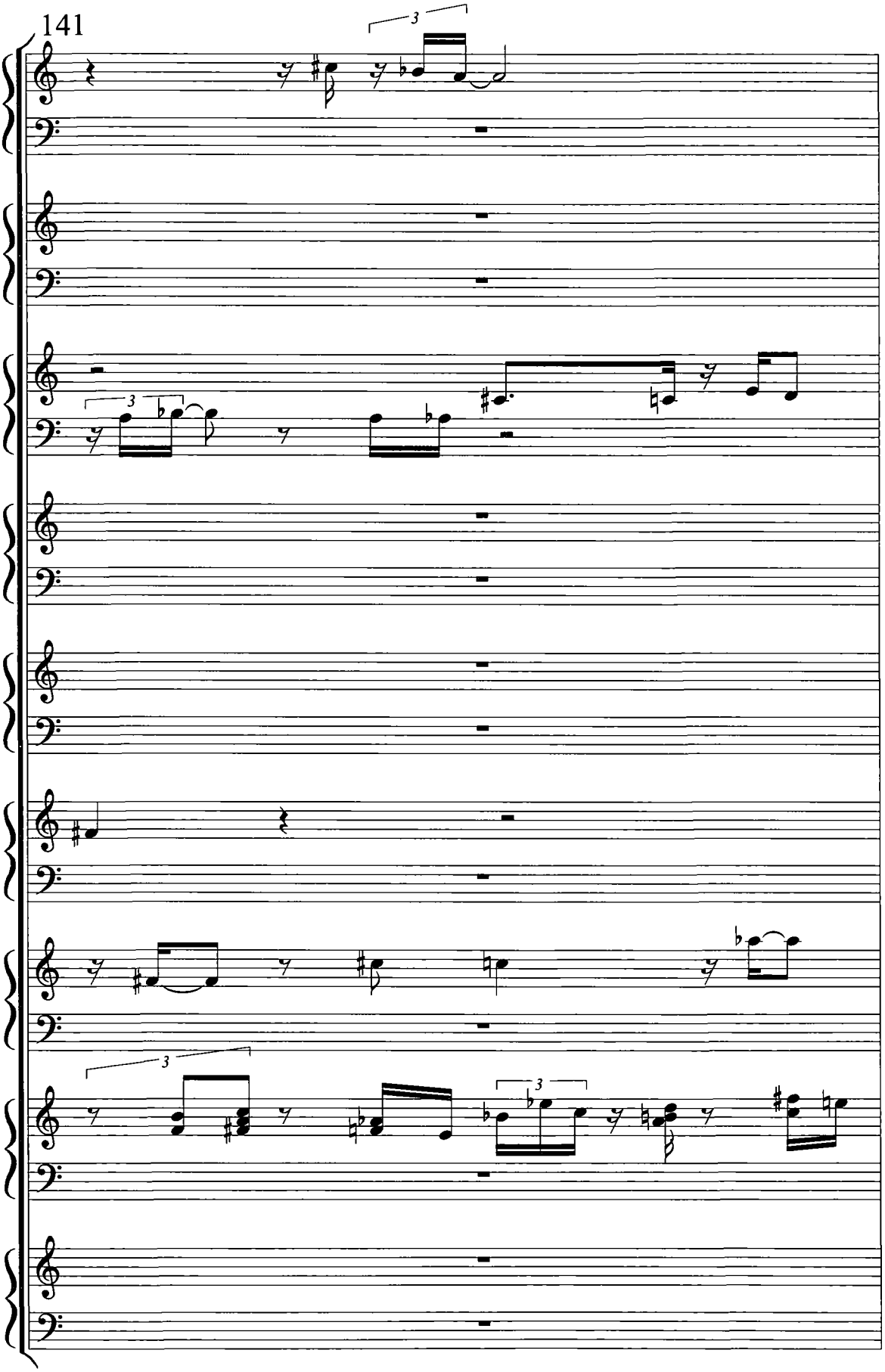

129
130

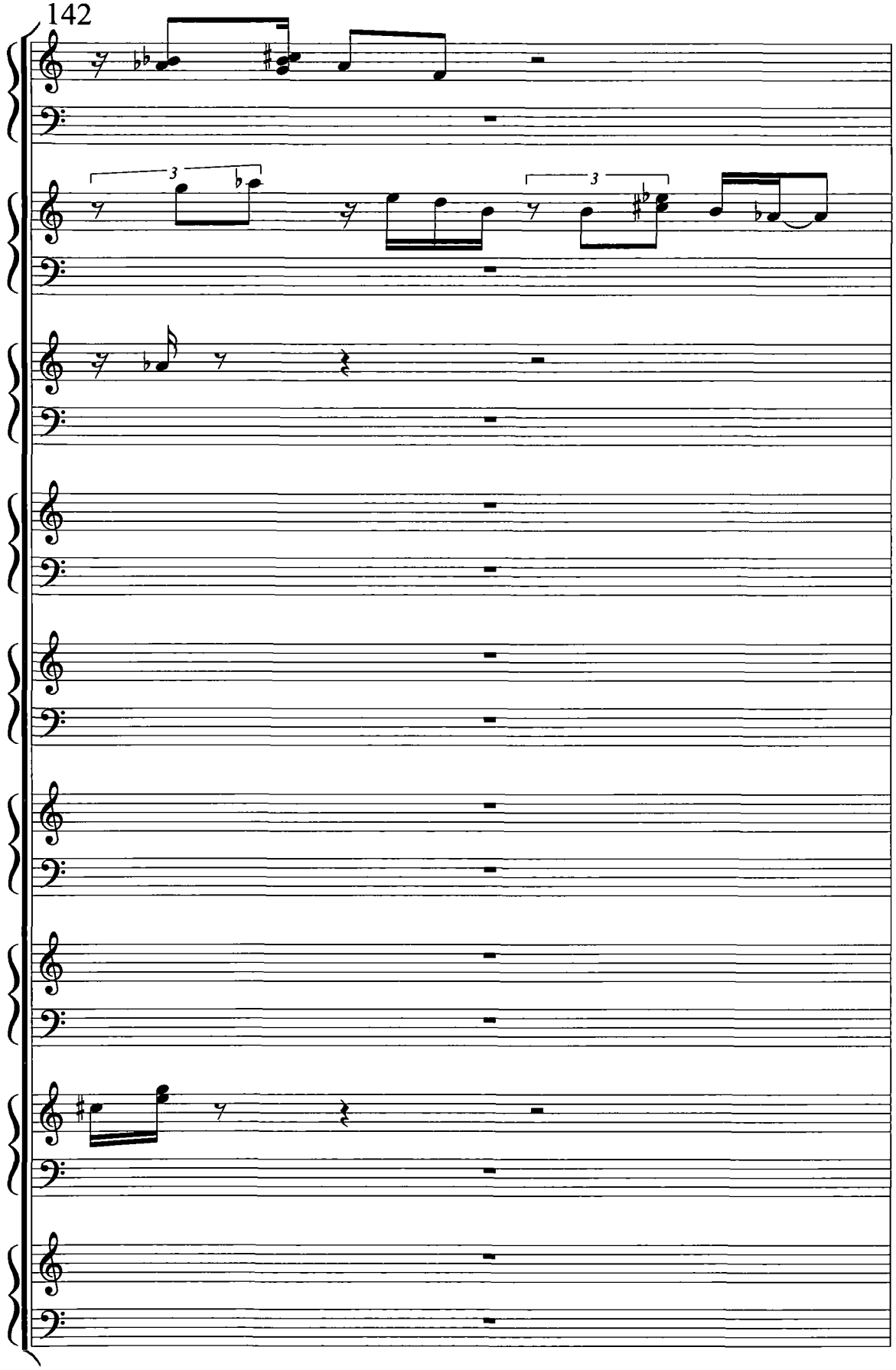

131
132
133
3
3

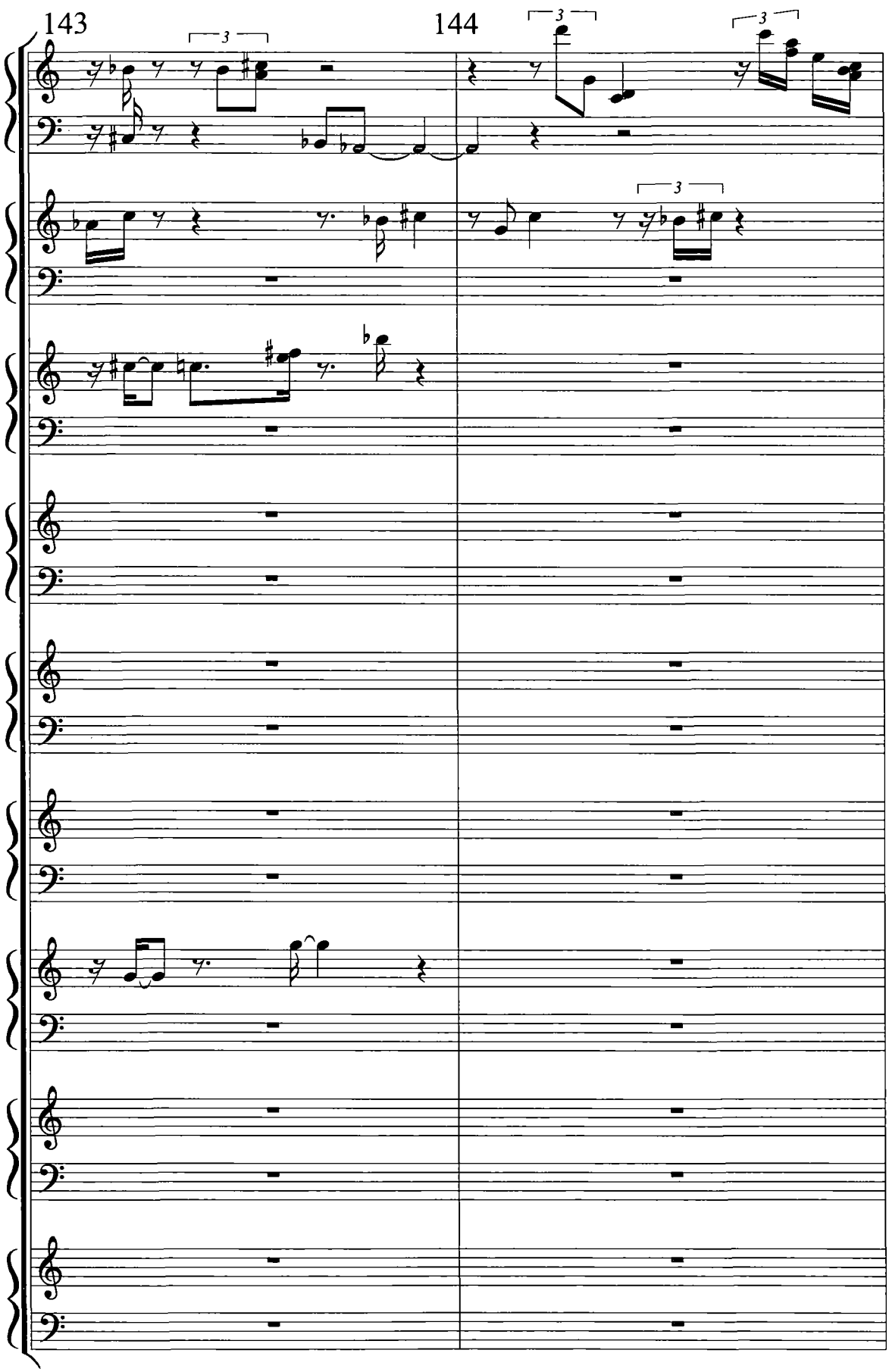

134
135
3

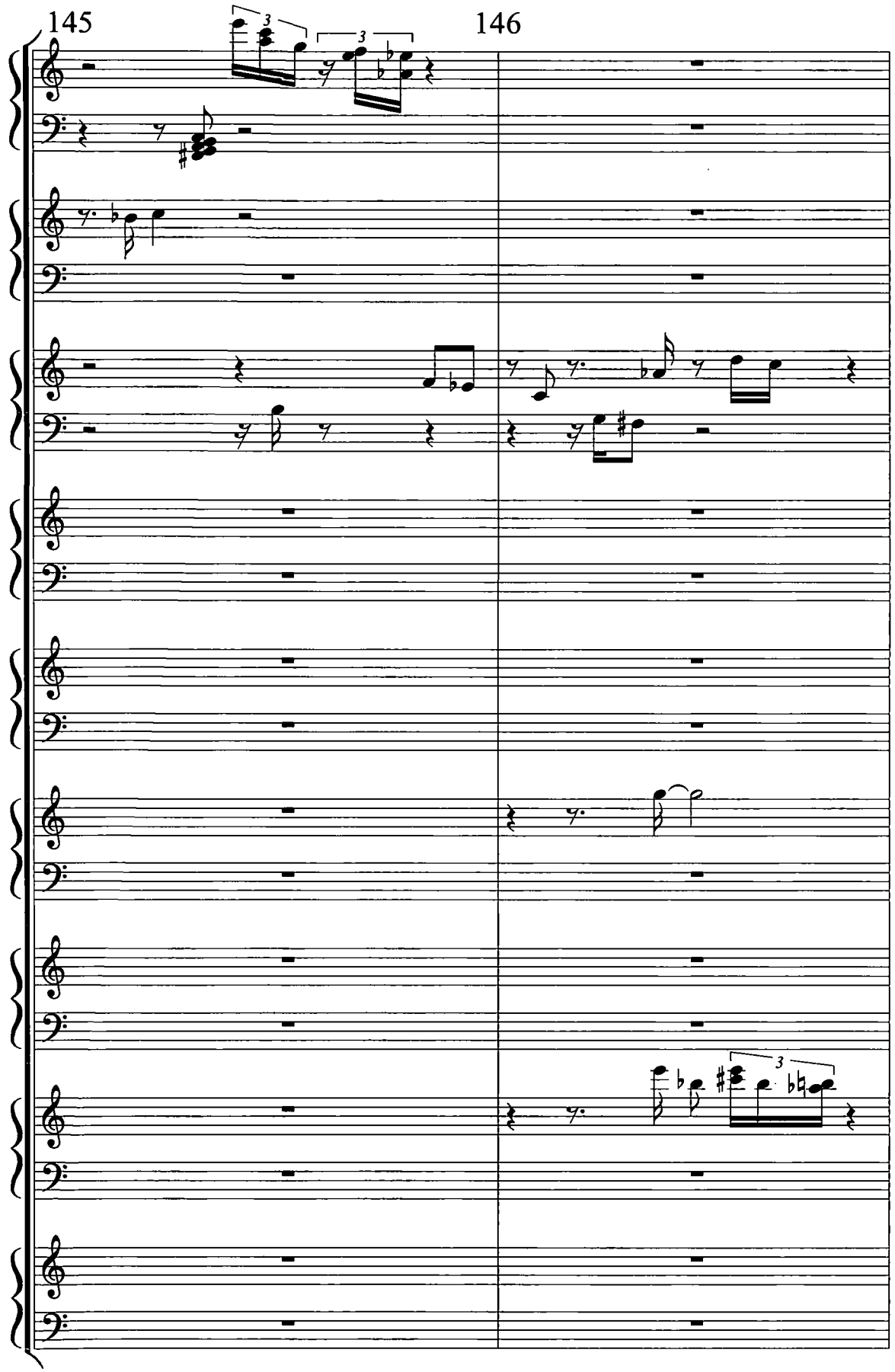

136
137
138

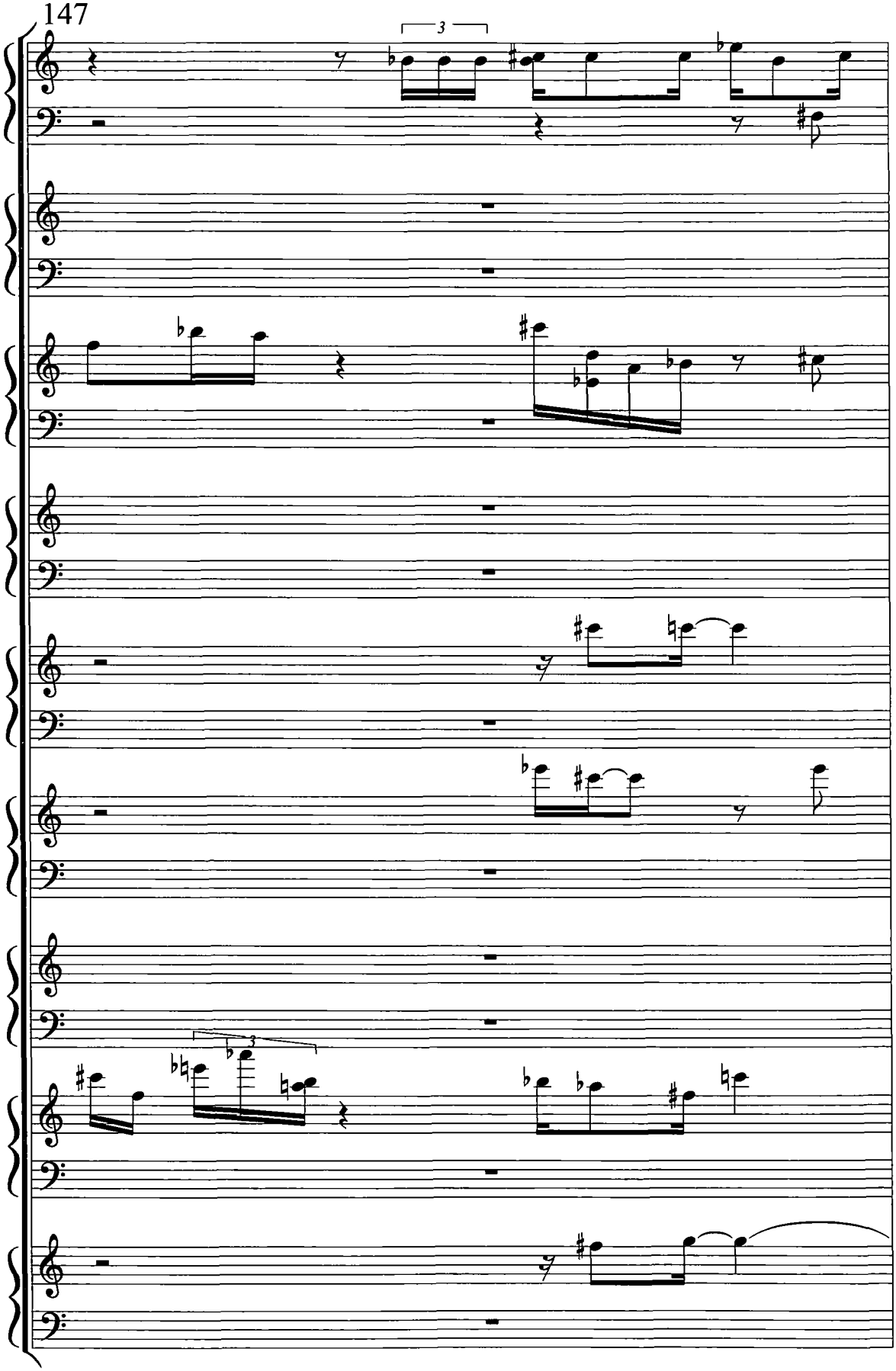

139
140

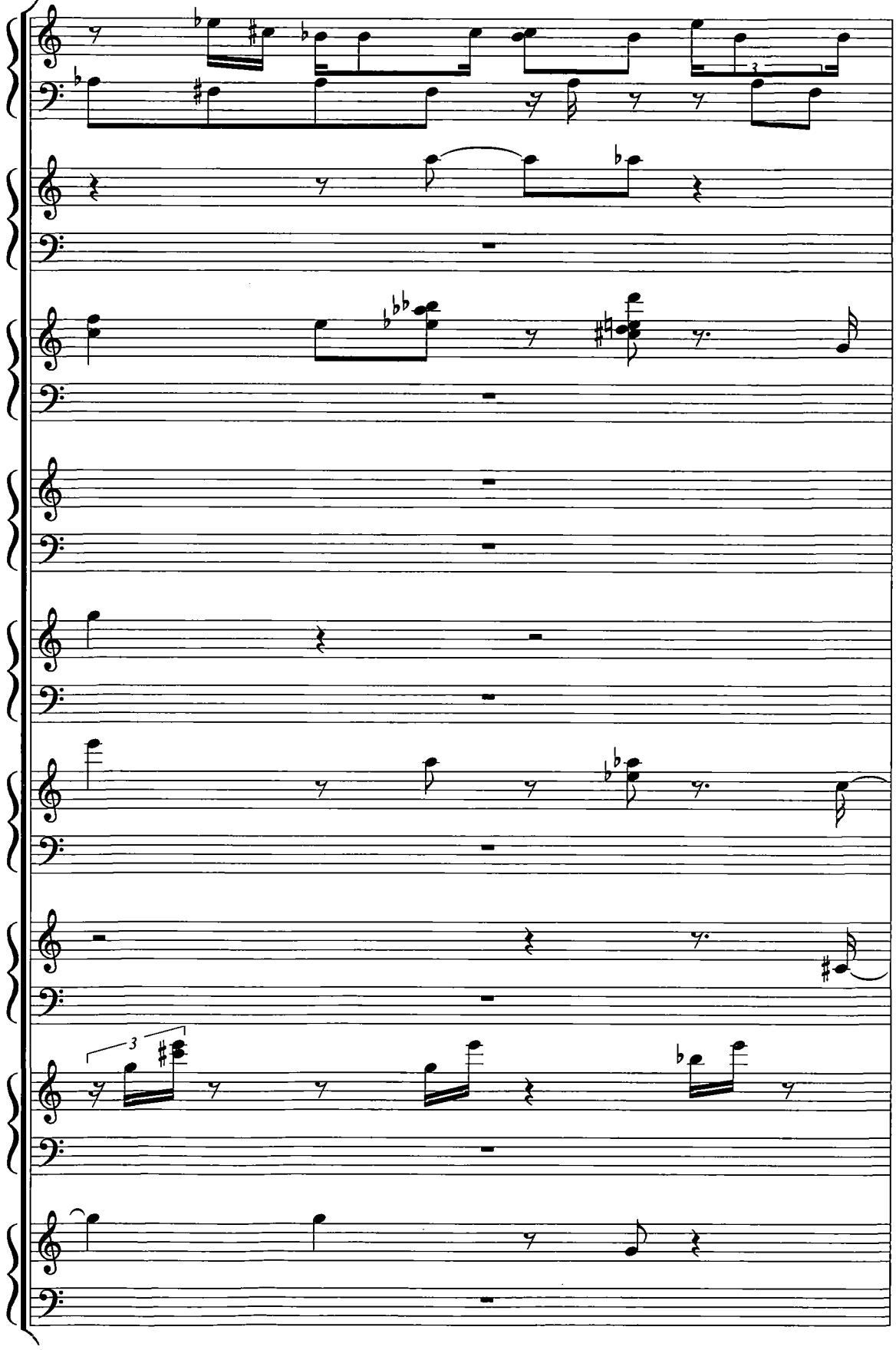


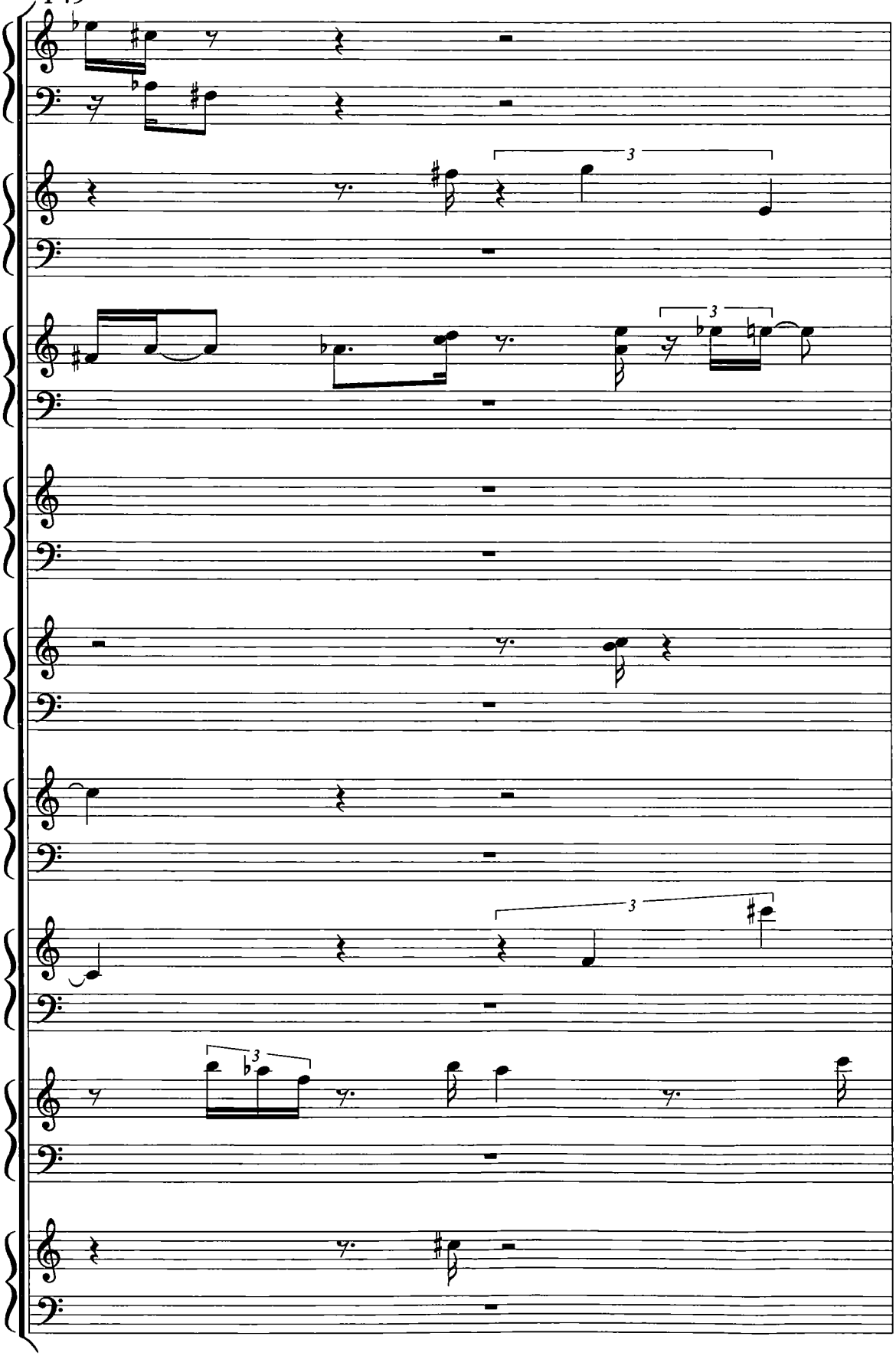


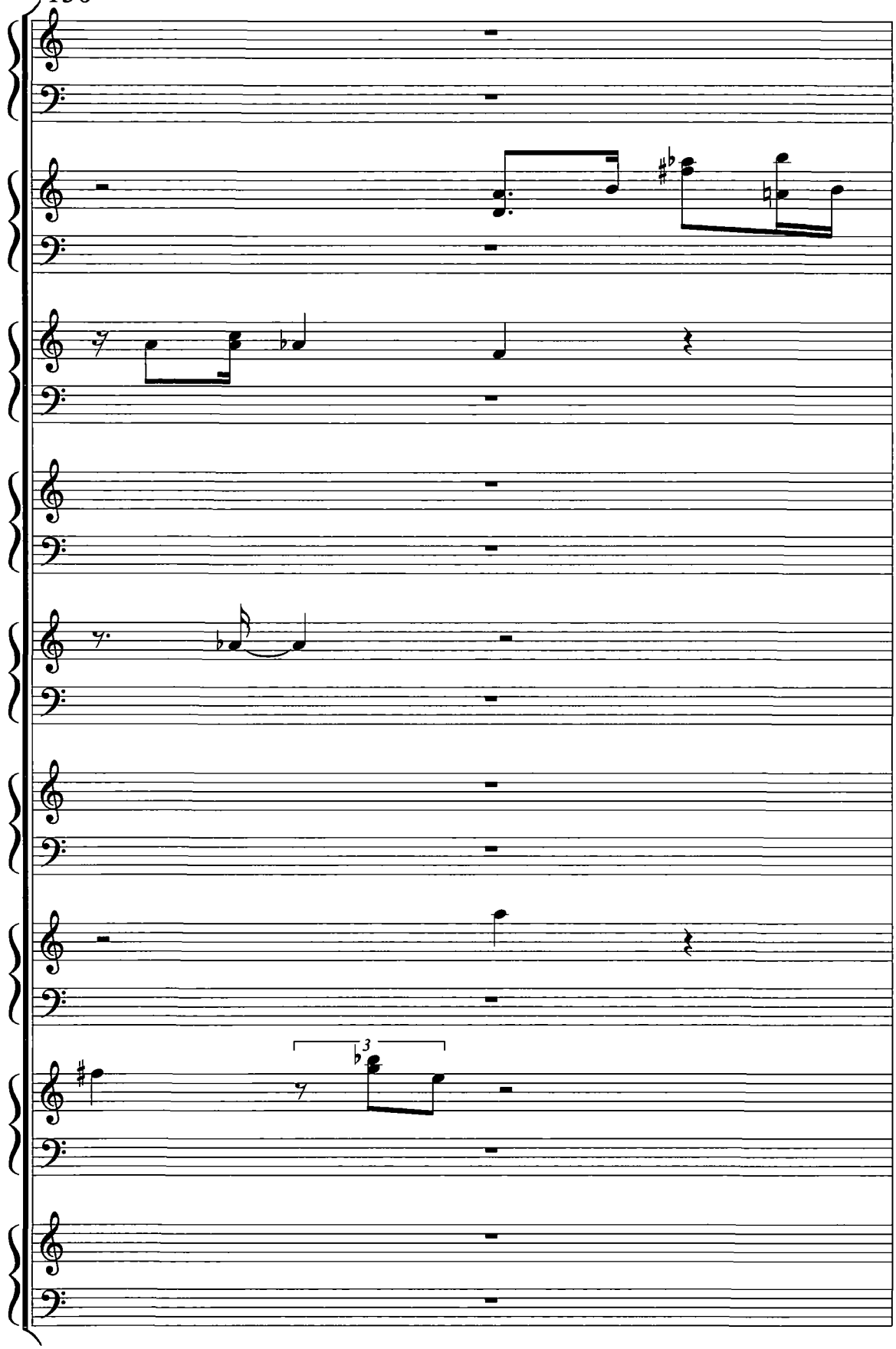

143
144
3
3
3
3


145
146









10.5.2 Analysis of the Musical Score

The previous pages show an excerpt of an interaction session as the combination of two MIDI files; (1) the output of the human interactor (top stave) and (2) the output of the eight player agents performing on their respective MIDI channels (1 to 8).

The top stave shows human input, the eight subsequent staves document player activity in the players' agency, one agent per stave. The excerpt encompasses measure 114 to measure 149.

It is tempting to focus analysis on the detection of highly similar patterns in the output of man and machine. However, as Oscar avoids the question-answer interaction paradigm such an analysis approach is not very informative. In contrast, Oscar evolves and behaves musically from the interpretation of changes in a complex networked architecture. As a consequence, the system can be influenced but not controlled, thus interaction happens on a higher level of abstraction, much higher than in an event-based interaction setting. For instance, the input-deliberation algorithm (chapter 6, section 6.5.4), the principle of multiple-influences (chapter 6, section 6.4) – instrumental in the agent's melody construction process – and the effect of nonlinear couplings spread out in the constituent networks, all this arranges for the expression of *continuous* rather than discrete levels of internal parametric specification. Nonetheless, one may trace correlations between, on the one hand, input data affecting agents output in non-trivial ways, and on the other hand, musical structures emerging from inter-agent interactions.

The score clearly echoes the clustering activity in the players' agency; the reference agent in the current cluster is identified by its MIDI-channel, which in turn specifies the stave number: 2 to 9 in the score. The melodic activity switches between

three agents in the first two measures (starting measure 114); from agent 5 to 1 to 2. The relative coherence of interval content of the three short sequences reveals a tight coupling between the three agents, the data in the agents input-deliberation vector most likely favours selection from the same source (neighbour) agent. However, agent 1 (stave 2) clearly exploits a private duration pattern.

Human input in measure 116: the last three events form the following interval sequence: -1, 1, -2 semitones. An especially rare instance of machine imitation occurs in agent 1, measure 121 where the same pattern appears starting on pitch B.

A clear instance of imitation by the human performer is spotted in measure 119, the negative interval of a fifth is an immediate response to the interval content of the short sequence produced by agent 3 in the preceding measure.

More activity develops starting measure 124; four agents (1, 3, 5 and 7) interact, reference agent is agent 3. In measure 126, the pattern F, E-flat produced by agent 3 is expanded into the following two-event pattern: E, F-sharp – the trailing source quarter note is echoed as a quarter rest in agent 7.

The duration dimension of the half-note event in measure 137 is picked up immediately by agent 3 in measure 138 and in agent 2 in measure 140. Figure 10.132 confirms a peaking exploitation level around 63%[5] way through the illustration – this means that the level human-responsiveness receives a significant boost – in other words, the human interactor is asking for attention by causing a contextual break by producing a loud event of prolonged duration.

---

[5] Percentage wise position in a figure is computed as: number of measures from the start divided by the total number of measures (114 to 149 = 35 measures) multiplied by 100. The resulting value makes it possible to relate the position of a given measure in the score.

Measure 139 documents another unambiguous example of melodic expansion in agent 5, the source events being produced by reference agent nr. 2. The first event, pitch middle-C, eight note, expands into two sixteenth-note events given a pitch offset of a fourth and a subsequent intervals-list starting with a positive interval of a semitone – thus F, F-sharp results. The connected events 2 and 3 in agent 2 appear as disconnected in agent 5 transposed by four semitones. Next, the quarter note F in agent 2 is split as a dotted eight plus a sixteenth note in agent 5 following an interval offset of respectively an octave (F) and 14 semitones (G). A modified version of the contents of measure 140, agent 2 appears in agent5; a dotted C-sharp is expanded into F-sharp and F, the remaining C appears as F-sharp. The human performer is quick to react; the final small pattern in measure 140 equally focuses on F-sharp. The increasing value of human-machine similarity (figure 10.128) at 74% confirms this observation.

Further examples of cluster-based orchestration occur in measure 141; considering reference agent nr. 2, contractions of interval patterns show up in agent 6 while complementary expansions of the same source patterns appear in agent 7. Also, the trailing pattern (duration eight note plus half note, pitch A) input by the human performer appears in measure 146, agent 5. Synchronized, expanded versions of that same pattern emerge in agents 2 and 7 in measure 146. As from measure 144, the human performer provides descending arpeggios spanning a pitch range of up to almost two octaves – figure 10.133 documenting human-responsiveness and the peaking levels of velocity (figure 10.116, human input, dimension of velocity) all confirm a change in context.

The density of agents' activity is exceptional in measure 147 since 5 agents exchange information as they agglomerate into the current cluster. Reference agent is

agent nr. 2. There are plenty of examples of orchestration by contraction and expansion. For example, the opening eight-note F in agent 2 is expanded into two sixteenth notes C-sharp, F. The following two events – interval of a minor second – are expanded into a triplet figure, intervals of four semitones up and 11 semitones down. Modified versions of the second figure starting on C-sharp appear in agents 4,5,7 and 8. The human performer has managed to exert increased influence on the current agents activation; this is confirmed by an incremental trend in human responsiveness shown in figure 10.133.

Large intervals and a regular rhythm typify human input, starting measure 148. The intervals are echoed without much delay in the following measures; interval of -15 semitones in agent 1 and +20 semitones in agent 6.

In conclusion, the score excerpt demonstrates musical behaviour of a distributed system based on loosely coupled components; influence from an external human performer merges seamlessly with an internal musical climate issuing from social affinities between individual agents. The performer has no authority to control but rather interferes in the expression of autonomous social affinities while the listening and motivation networks also interact in complex ways. The global result is coherent but non-trivial emergent functionality.

10.6 Experiment e8: Analysis of System Behaviour in Relation to Musical Output

This experiment aims to offer detailed insight on how specific momentary situations in systems behaviour give rise to the synthesis of particular musical output. In other words, how the dynamics of various system components are related to the nature of the actual musical output. The following systems quantities have immediate impact on the music producing algorithms:

1) The input-deliberation-vector
2) The activation levels of the agents
3) The energy levels of the agents
4) The private patterns held by every agent
5) The compound-function held by every agent
6) The current clusters of the player agency
7) The current level of human-responsiveness
8) The parameter settings of the player agency: primarily the value of the critical-distance parameter and contents of the social affinities matrix.
9) The contents of working memory
10) The orientation of the current drive

The settings for this experiment are as follows:

N	Steps learn	Steps breed	Steps perform	System inclination	Learn	Input	Learning-mode
8	1	16	160	Selfish	Yes	Human	Real-time

Table 10.10: Global parameter settings in experiment e8.

The critical-distance in the agency remains fixed at 74 throughout the experiment. The affinities matrix is filled with random numbers between 10 and 400, the orientation-bit is switched off and energy is set to 100.

A number of typical score excerpts are presented reflecting the complexity of the interpretation algorithms, in particular, the creation of polyphonic sequences from a melody held by the reference agent as conditioned by the activation levels of the agents in a given cluster. In every performance step (1 to 160), machine output is computed and two MIDI files are stored to disk for later analysis: (1) a channel-zero file and (2) the reference-agent file. As explained in chapter 6, the contents of the channel-zero file are used to compute the current musical distance between human and machine. The reference-agent's MIDI file holds a manipulated version of the channel-zero data; the data processed by the current compound-function in addition to a number of potential parallel voices. At every point in time, only the reference-agent's melody is performed.

A note on the score excerpts: the optimal score visualisation is a compromise between readability and accuracy. Our score notation program (Apple Logic Express) includes an "interpretation" function that helps to create a cleaner, more readable score image, however partially distorting the actual duration data in specific ways. Switching "interpretation" off adds excessive visual notation complexity but provides a score image more accurately reflecting the rhythms in the MIDI file. It was decided to keep "interpretation" on because the timing relationships between the various voices remain very well synchronised since the programs optimisations are consistent – this results in a clean image also clearly reflecting the timing relationships between the various parallel voices.

10.6.1 Experiment e8, Score Excerpt nr. 1: Performance–step nr. 9

10.6.1.1 Description of Score Excerpt nr. 1

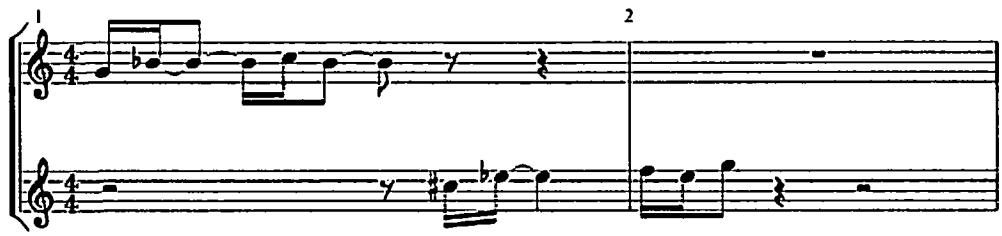


Figure 10.139: player-agency score excerpt at performance step nr. 9, voices 1 and 2.

Listing nr. 1: listing of system output in the Lisp listener, documenting performance-step nr. 9.

```
Clusters: ((3 5 8) (0 1 2 7 9))

Strongest cluster: (3 5 8).

Reference-agent: #<AGENT #x25214CE>, ID: 3

Cluster agents:

agt:3  stat: ASLEEP cdis: 74  egy: 27  act: 77  ne: 9  neC0: 9  cf: 1 ip:#(44 45 45)
ipa:#(40 64 20)  nb:(5 8)  pnb:NIL

agt:5  stat: ASLEEP cdis: 74  egy: 27  act:  7  ne: 7  neC0: 5  cf: 0 ip:#(50 48 53)
ipa:#(90 13 24)  nb:(3 8)  pnb:(8)

agt:8  stat: ASLEEP cdis: 74  egy: 27  act:-45  ne: 6  neC0: 8  cf: 4 ip:#(41 54 46)
ipa:#(40 63 12)  nb:(3 5)  pnb:(5)

Human-responsiveness: 41.
```

```
Agency autonomous.
Nr-events: 3, reference-channel: 3, intervals: (-1 1 -2), density: 27.

Ego/neighbour selection chance: 52 %, bits: (1 0 0 1), neighbour: 5.

Make-reference-melody:
     nr-events: 5
     density: 50
     pitch-0: 46
     intervals: (-1 1 -2)
     durations: (-0.4 0.4 0.1 0.1 0.1 1.2 0.2) (seconds)
     velocities: (117 117 81 76 60 67 117)
     iot: (0.2 0 0.05 0 0 0)
     channel: 3

Melody: CP_1 [Nr-events: 4] Duration: 0 min 2 sec 550 msec.

 S:   600 1150 1250 1350
 P:    74   71   72   70
 V:   117   76   60   67
 D:   400  100  100 1200
 C:     3    3    3    3

Feedback algorithm using compound-function:

COMPOUND-FUNCTION CP_1:
       source    : #<ppm-melody "events-pane-mel">.
       fitness-ie: 0  0.
       fitn-critc: 0.
       nr-events : 4, nr-events channel-zero: 4.
       nrt-used  : 0.
```

```
read-pntrs: (20 0 30 29).
read-range: (5 5 5 6).

1:   9 PRESENT EXTEND-MELODY            (5 (-3 2 -2))
2:   1 PRESENT TRANSPOSE-PITCHES        5
3:  10 PRESENT RETROGRADE               NIL

Melody: CP_1 [Nr-events: 9] Duration: 0 min 4 sec 650 msec.

S:     0   99 1299 1399 2599 2699 3899 4000 4250
P:    67   70   72   70   73   75   77   76   79
V:    60   67   60   67   60   67   60   76  117
D:   100 1200  100 1200  100 1200  100  100  400
C:     3    3    3    3    0    0    0    0    0
```

10.6.1.2 Analysis of Score Excerpt nr. 1

There are two cluster configurations of respectively three and five agents. All agents in the player agency are in sleep mode and feature the same energy level (27). Therefore, there is no "strongest" cluster and the first one with agents ID's (3 5 8) is selected by default. The first agent (3) of that cluster is elected as the reference-agent and the compound-function attributed to that agent (CP_1) will be used for musical processing, as detailed in a moment. Also notice that the agent's status (asleep or awake) has only impact on the creation of parallel events to the reference-melody (detailed in a moment), not to the creation of the reference-melody itself.

Current human-responsiveness is rather low (41) and the probabilistic selection algorithm selects autonomous rather than responsive behaviour (please refer to chapter 7, section 7.3). Next, the relationship between two values in the input-pressures-adapted will condition what data will be used to construct the reference-

melody. In autonomous mode, the vector slots two and three (counting from one) hold respectively the pressures for Ego and Neighbour – the first vector slot, Human, is ignored since we perform in autonomous mode. Ego means specific data is taken from the reference agent itself while Neighbour means that specific data is borrowed from a the chosen neighbouring agent. The Ego/Neighbour selection chance happens to be 52%, meaning that both pressures are approximately equal in strength. The chance level is remapped as to act as a selection device in order to select a bit-pattern from the *sorted-patterns* collection (chapter 6, figure 6.1); the bit pattern (1 0 0 1) is chosen. This implies that the input material serving the construction of the reference melody is as follows; the first bit (value 1) meaning pitch intervals are taken from Neighbour, the second bit and the third bit are zero, meaning velocity and duration data is taken from Ego and the last bit is one, meaning inter-onset-times are taken from Neighbour. This example illustrates the principle of mixing multiple influences (chapter 6, section 6.4) to construct machine statements whether in autonomous or responsive mode.

The function *make-reference-melody* is called next, its arguments are as follows:

1) Nr-events (value: 5) is proportional to the energy level of the reference-agent; nr-events is computed as (round (max 1 (/ (energy reference-agent) 5))), therefore the number of events varies between 1 and 21.

2) Density (value: 50) offers probabilistic conditioning of whether a sounding event is generated rather than a rest, density is computed as (max 50 (energy reference-agent)).

3) Pitch-0 (value 46) is the starting pitch of the melody to be generated. It is computed as (+ 48 (choose (intervals reference-agent)) e.g., the melody

starts with a pitch offset randomly chosen from the agent's current intervals.

4) Intervals (value: (-1 2 -2)). In this particular example, this interval pattern is provided by the reference-agent.

5) Durations (value: (-0.4 0.4 0.1 0.1 0.1 1.2 0.2) in seconds) In this particular example, the durations pattern is taken from neighbour agent nr. 5.

6) Velocities (value: (117 117 81 76 60 67 117)), this pattern is also taken from neighbour agent nr. 5.

7) Inter-onset-times (value: (0.2 0 0 0.5 0 0 0) in seconds). This pattern is provided by the reference-agent.

8) Channel (value: 3, meaning MIDI channel 4) is the destination MIDI channel for the events to be generated, it is equal to the MIDI channel of the reference-agent.

The algorithm cycles through all the argument lists (modulo the length of each list) which yield a reference-melody of four events, not five as expected from the nr-events argument. The first prospective event is not collected because its duration argument is negative (-0.4).

```
Melody: CP_1 [Nr-events: 4] Duration: 2.55 sec.

S:   600 1150 1250 1350
P:    74   71   72   70
V:   117   76   60   67
D:   400  100  100 1200
C:     3    3    3    3
```

The reference-melody listing above shows five data items per event; starting-time (S), pitch (P), velocity (V), duration (D) and MIDI channel (C). Times and durations are in milliseconds. The contents of the reference-melody are copied to the channel-zero instance variable of the current CF, its contents will be used with the comparator object to compute musical distance between human and machine (chapter 8, section 8.3).

A new melody is now computed by applying the feedback-algorithm (chapter 6, section 6.5.3) to the contents of the reference-melody, the result is collected in the compound-function (a subclass of the Melody object) itself.

As seen above in listing nr. 1, the current compound (CP_1, i.e. the first function in the compound-function-pool) was never applied before so its integration and expression fitness levels are zero. The data in the read-pointers and read-ranges specify how the source information for the respective dimensions (pitch, velocity, duration, iot) will be read; read-pointers provide a start-position and read-ranges specify how many values are read – all pointers are automatically kept within appropriate ranges because they are applied modulo the length of the source lists in question. CP_1 contains three processing functions with their respective arguments and are applied in sequential order: extend-melody, transpose-pitches and retrograde. Five events (channel equals 0, meaning MIDI channel 1) are added to the events already in the reference-melody, the accumulative result is seen in figure 10.139. Finally, since all cluster agents are asleep, none has the capacity to generate additional parallel events to the present contents of CP_1.

10.6.2 Experiment e8, Score Excerpt nr. 2: Performance–step nr. 63

10.6.2.1 Description of Score Excerpt nr. 2

Listing nr. 2: listing of system output in the Lisp listener, documenting performance-step nr. 63.

```
Clusters: ((6 8) (2 4) (0 1 5 7 9)).

Strongest cluster: (0 1 5 7 9).

Reference-agent: #<AGENT #x250A2EE>, ID: 0

Cluster agents:

agt:0  stat: ACTIVE cdis: 89  egy: 69  act:  7 ne:50  neC0:10  cf: 3
ip:#(59 42 54)  ipa:#(100 18 5)  nb:(1 7)  pnb:(1 7)

agt:1  stat: ACTIVE cdis: 89  egy: 69  act:-94 ne:11  neC0: 8  cf: 7
ip:#(46 49 45)  ipa:#(66 2 2)  nb:(0)  pnb:(0 7)

agt:5  stat: ACTIVE cdis: 89  egy: 69  act:-26 ne:11  neC0: 8  cf: 7
ip:#(50 48 53)  ipa:#(100 28 32)  nb:(7 9)  pnb:NIL

agt:7  stat: ACTIVE cdis: 89  egy: 69  act: 94 ne: 5  neC0: 5  cf: 1
ip:#(47 58 57)  ipa:#(31 67 66)  nb:(0 5)  pnb:(0 1 2 6 8)

agt:9  stat: ACTIVE cdis: 89  egy: 83  act:  0  ne: 7  neC0: 5  cf: 4
ip:#(56 58 44)  ipa:#(73 2 2)  nb:(5)  pnb:(4)

Agency-autonomous.
Nr-events: 7, reference-channel: 0, intervals: (-2 1 -2), density: 69.
```

```
Ego/neighbour selection chance: 57 %, bits: (1 1 0 0), neighbour: 5.


Make-reference-melody:
     nr-events: 14
     density: 69
     pitch-0: 46
     intervals: (-3 5 -5 2 -1 3)
     durations: (0.1 1.2 0.2 1.2 1.2 1.2)
     velocities: (71 73)
     iot: (0.05 0 0 0.4 0)
     channel: 0


Melody: CP_3 [Nr-events: 7] Duration: 11 sec 25 msec.


 S:   150 1550 3150 4350 5600 750011150
 P:    72   69   68   71   68   70   69
 V:    73   73   71   73   71   73   71
 D:  1200 1200 1200 1200  100 1200  100
 C:     0    0    0    0    0    0    0


Feedback algorithm using compound-function:


       name      : CP_3.
       source    : #<ppm-melody "events-pane-mel">.
       fitness-ie: 0  4.9.
       fitn-critc: 88.
       nr-events : 7 nr-events channel-zero: 7.
       nrt-used  : 3.


       read-pntrs: (24 9 28 7).
       read-range: (9 6 7 4).


      1:   3 PRESENT INVERT-INTERVALS          NIL
```

```
2:   9 PRESENT EXTEND-MELODY          (3 (0 0 1))

Melody: CP_3 [Nr-events: 10]

Apply de-grouping algorithm, activation: -94.
Arguments: 0 (-5 1 -2 1 -2 2 -1) 7 0.3 (2 4 4 4) (T T T T) 1 (0).

Apply de-grouping algorithm, activation: -26.
Arguments: 0 (-3 5 -5 2 -1 3) 12 0.6 (2 2 2 3) (T T NIL T) 5 (2 2 0).

Apply grouping algorithm, activation: 94.
Arguments: (4 2 4 3) 0 (-2 1 -2) (T T T T) -12 7.

Melody: CP_3 [Nr-events: 50], Channels: (0 1 5 7).
Melody duration: 0 min 13 sec 749 msec.
```

10.6.2.2 Analysis of Score Excerpt nr.2

We address system activity at performance-step nr. 63. From the data above we can see that this example is similar to the previous one with the exception that three parallel voices are created. Given the agents cluster (0 1 5 7 9), only agents 1, 5 and 9 contribute a parallel voice since the activation level of agent 9 is zero. In figure 10.140, the top stave shows the reference-melody computed according to the bit pattern (1 1 0 0).

The second agent in the cluster (agent ID nr. 1) features an activation level of -94 meaning it may contribute additional events using the de-grouping algorithm. This algorithm creates many additional events from the inspection of a single source event, therefore it is also referred to as an "expansion" algorithm. The arguments are:

0 (-5 1 -2 1 -2 2 -1) 7 0.3 (2 4 4 4) (T T T T) 1 (0)

meaning respectively, the source-channel, pitch-intervals, transposition, minimum-duration, duration-dividers, groups-flags-list, destination-channel and delays-list. Any source event must at least be 0.3 seconds in duration in order to be considered for expansion. The list (2 4 4 4) specifies the number of events to ge generated from a single source event, starting at a pitch offset of 7 semitones, generating pitches from the intervals-list and collecting the nascent event or rejecting it as specified in the groups-flags-list – all elements are True so all potential events are accepted.

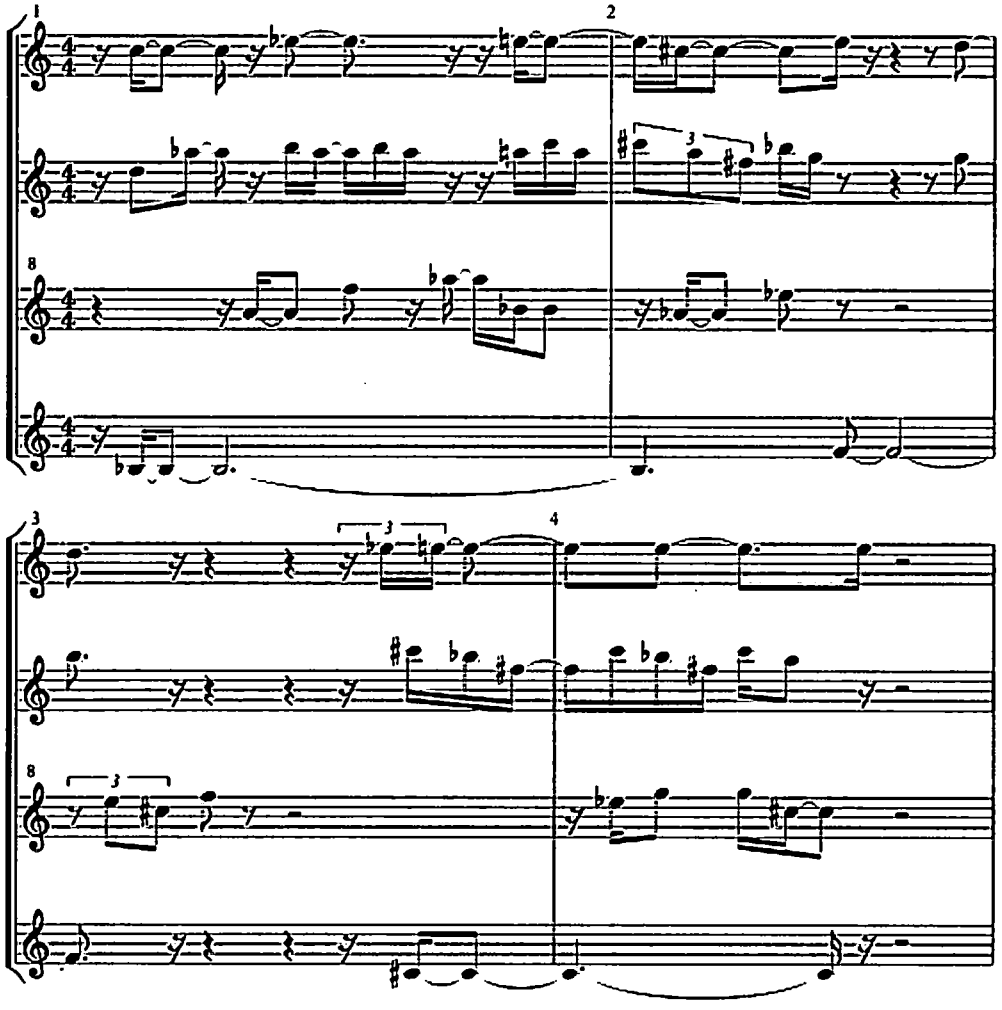


Figure 10.140: player-agency score excerpt at performance step nr. 63, voices 1 to 4, top to bottom.

The intervals argument corresponds to the contents of the intervals data held by the providing source agent, i.e., the reference agent identified by source-channel 0. The flags-list is computed as follows:

```
(loop repeat 4 collect (< (random 75) (activation a)))
```

meaning that the number of True elements is stochastically proportional to the activation level of the agent in question. The transposition argument is chosen at random from the list (-12 -3 -4 -7 3 4 7), pitch intervals in semitones.

For example, the first event in voice 1 (pitch C5, duration is one quarter note) is split in two events appearing in voice 2 (pitches D4 and G#4, durations are two eight notes). The starting pitch of voice 2 is computed as the sum of the starting-pitch of the source channel, the transposition value and an interval taken from the intervals list, in this case (in MIDI key numbers): 72 + 7 + (-5) = 74 e.g., pitch D4. The seconds pitch in voice 2 is 72 + 7 + 1 = 80 e.g., pitch G#4. When an argument is a list of values (rather than a single number), its values are addressed sequentially, again using a pointer computed as modulo the length of the list.

The second event in voice 1 is split in four events as is the third and the fourth using the same de-grouping algorithm. Given a delays-list argument of (0), meaning no delays, the events in voice 1 and 2 are synchronised; the first four events in voice 1 coincide with the first 14 events in voice 2, event 6 in voice 1 with events 15 and 16 in voice 2 and finally, events 18 and 19 coincide with events 17 to 24 in voice 2.

The contents of voice 3 is also conditioned by a de-grouping algorithm with the following arguments:

```
0 (-3 5 -5 2 -1 3) 12 0.6 (2 2 2 3) (T T T NIL) 5 (2 2 0)
```

The minimum-duration is 0.6 seconds, the groups-flags-list contains one NIL meaning there will be less activity in voice 3 and the delays-list is (2 2 0), therefore the source-voice (voice 1) and the destination-voices (voice 3) will not synchronise. The first pitch of voice 3 is computed as: 72 + 12 + (-3) = 81 e.g., pitch A5. Its duration is half the duration of the first event in voice 1. The event takes a delay of 2 steps in consideration (first element of delays-list); this means that it starts playing with an entry-delay equal two times its duration plus the entry-delay of the source event. The entry-delay of the first event in voice 3 is computed as (2 * 576 msec) + 144 msec (entry-delay of first event in voice 1) = 1296 msec. Note that the timing information received a slightly distorted representation in the score of figure 10.140 due to the interpretation function of the score notation program, as explained above. Notice that the third event in voice 1 (pitch C#5) is not expanded because it coincides with the NIL value in the groups-flags-list.

Voice 4 is the result of the activation in agent nr. 7, more precisely positive activation (level 94), therefore the grouping-algorithm will be put to work, its arguments are:

```
(4 2 4 3) 0 (-2 1 -2) (T T T T) -12 7
```

The arguments represent respectively, groupings, source-channel, intervals, groups-flags-list, transposition and destination-channel. The grouping-algorithm creates additional events by combining the durations of groups of source-events. The groupings are computed as follows:

```
(loop repeat 4 collect (choose '(2 2 3 4)))
```

The intervals data matches the pitch-intervals currently held by agent nr. 7. All potential events will be collected since all elements in the groups-flags-list equal True. The first event of voice 3 (pitch Bb3) coincides with the first four events in voice 1 and the second event (pitch F4) matches the next group of two source events pitches E5 and C#5.

This example illustrates the combined effect of the grouping and de-grouping algorithms in addition to the control of rhythmic organization by way of the groups-flags-list argument.

10.6.3 Experiment e8, Score Excerpt 3: Performance–step nr. 68

10.6.3.1 Description of Score Excerpt nr. 3

Listing nr. 3: listing of system output in the Lisp listener, documenting performance-step nr. 68.

```
Clusters: ((4 9) (0 1 2 6 7 8)).

Strongest cluster: (0 1 2 6 7 8).

Reference-agent: #<AGENT #x250A2EE>, ID: 0

Cluster agents:

agt:0  stat: ACTIVE cdis: 89  egy: 83  act: 94 ne:48  neC0:12  cf: 3
ip:#(59 42 54)  ipa:#(100 5 2)  nb:(1 7)  pnb:(1 7)

agt:1  stat: ACTIVE cdis: 89  egy: 83  act: 94 ne:48  neC0:12  cf: 3
ip:#(46 49 45)  ipa:#(100 2 2)  nb:(0 7)  pnb:(0)
```

```
agt:2  stat: ACTIVE cdis: 89  egy: 83  act: 84 ne:48  neC0:12  cf: 3
ip:#(58 46 50)  ipa:#(100 13 18)  nb:(6 7 8)  pnb:(4)

agt:6  stat: ACTIVE cdis: 89  egy: 83  act:-94 ne:14  neC0:22  cf: 2
ip:#(43 53 56)  ipa:#(100 2 2)  nb:(2 7 8)  pnb:(8)

agt:7  stat: ACTIVE cdis: 89  egy: 83  act: 73 ne:14  neC0:22  cf: 2
ip:#(47 58 57)  ipa:#(70 46 46)  nb:(0 1 2 6 8)  pnb:(0 5)

agt:8  stat: ACTIVE cdis: 89  egy: 83  act:  4 ne:14  neC0:22  cf: 2
ip:#(41 54 46)  ipa:#(100 2 2)  nb:(2 6 7)  pnb:(6)

Agency responsive.
Human/ego selection chance: 95 %, bits: (0 0 0 0).

Melody: working-memory [Nr-events: 32] Channels: (0).
Duration: 0 min 21 sec 966 msec.

intervals = (-5 -5 -2 -1 -2 -2 -3 8 9 14 1 5 2 5 2 3 4 -2 -20 0 0 -7 7 -7 7 -8 8 -7 0
0 7).

durations = (0.093 0.11 0.086 0.092 0.091 0.109 0.094 0.208 0.214 0.116 0.086 0.05
0.14 0.054 0.106 0.088 0.093 0.092 0.099 1.437 1.591 1.344 0.242 0.196 0.235 0.117
0.145 0.192 1.922 0.281 0.201 0.197)

Feedback algorithm using compound-function:

       name      : CP_3.
       source    : #<ppm-melody "events-pane-mel">.
       novelty   : 50.
       fitness-ie: 0.0  9.857.
       fitn-critc: 76.
       nr-events : 0 channel-zero: 0.
```

```
        nrt-used   : 8.

        read-pntrs: (24 9 28 7).
        read-range: (9 6 7 4).

      1:    3 PRESENT INVERT-INTERVALS          NIL
      2:    9 PRESENT EXTEND-MELODY             (3 (0 0 1))

Melody: CP_3 [Nr-events: 9]

Apply grouping algorithm, activation: 94.
Arguments: (4 4 3 2) 0 (-5 1 -2 1 -2 2 -1) (T T T T) -3 1.

Apply grouping algorithm, activation: 84.
Arguments: (2 2 2 2) 0 (-3 1) (T T T T) -4 2.

Apply de-grouping algorithm, activation: -94.
Arguments: 0 (-1 5) 12 0.3 (4 4 2 4) (T T T T) 6 (0).

Apply grouping algorithm, activation: 73.
Arguments: (3 3 4 2) 0 (-2 1 -2) (T T T T) 4 7.

Apply de-grouping algorithm, activation: 4.
Arguments: (2 2 2 2) 0 (-2 2 -4 2) (NIL NIL NIL NIL) 7 0.

Melody: CP_3 [Nr-events: 48] Channels: (0 1 2 6 7).
```

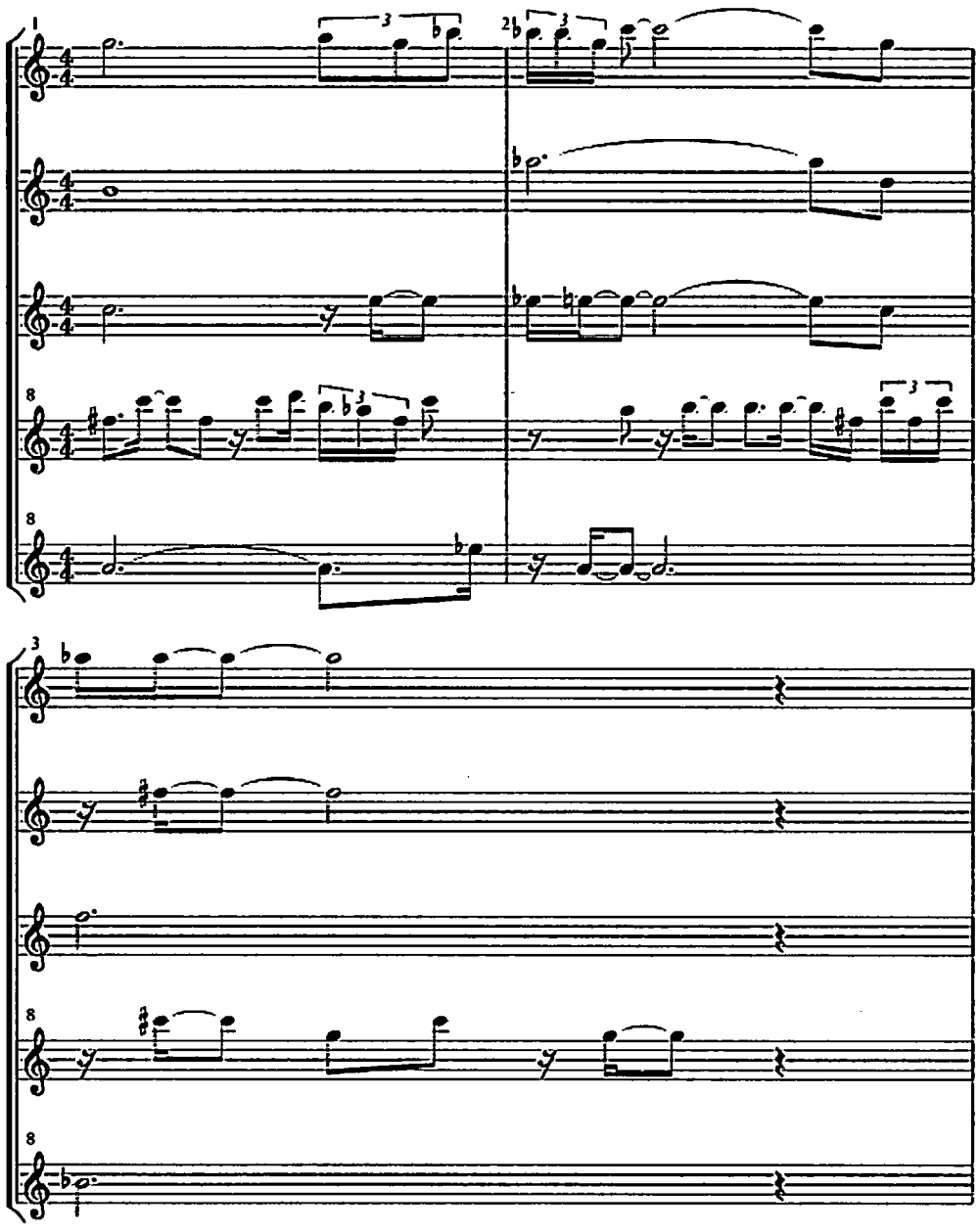


Figure 10.141: player-agency score excerpt at performance step nr. 68, voices 1 to 5, top to bottom.

10.6.3.2 Analysis of Score Excerpt nr. 3

This section contains the analysis of system activity at performance-step nr. 68. The input-pressures-adapted vector for all agents in the current cluster equals 100 implying that the level of human-responsiveness is very high. The probability for

selecting Human vs. Ego is 95 %. This results in a bit-pattern (0 0 0 0) meaning that working-memory will serve as a source to provide data from its four dimensions: pitch, velocity, duration and inter-onset-times. The current content of working-memory is depicted in figure 10.142.

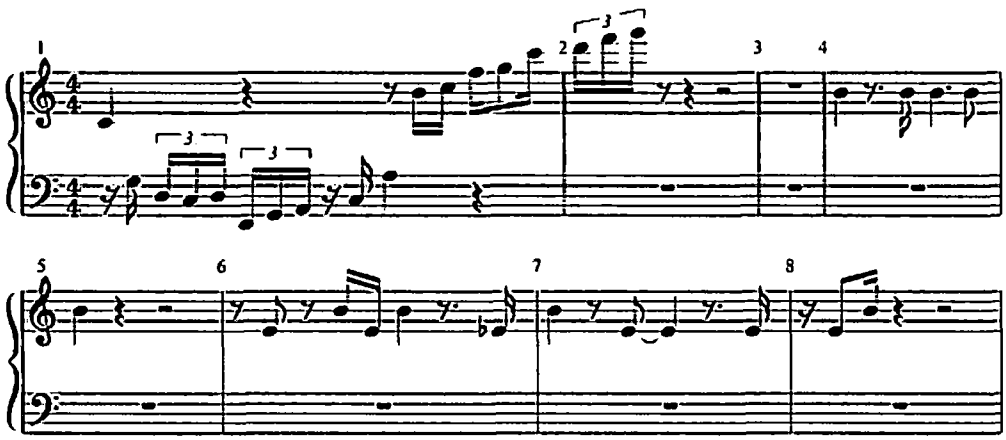


Figure 10.142: Contents of working-memory at performance step nr. 68.

As seen from listing nr.3, the read-range parameter of the current compound-function (CP_3) is (9 6 7 4) – this specifies the amount of information taken from the source melody, in this case, exclusively working-memory, in respectively the dimensions of pitch, velocity, duration and inter-onset-time.

In observation of the read-pointers parameter in listing nr. 3, (24 9 28 7), it means we start reading pitch-interval data at the twenty-fourth event of working-memory i.e., the fourth event in bar 6 (pitch B3). The read-range parameter is 9, which yields eight pitch intervals. The eight pitch intervals read from working-memory are as follows: (-8 8 -7 0 0 7 -5 -5) – the last two intervals are read from the start of working-memory because the read-pointer is taken modulo the length of the number of events in working-memory.

These intervals are reflected in a modified form (inverted) and with a number of intervals added by the second processing function (extend-melody) in the compound-function in use: (2 -2 3 0 0 -3 5 5 1 0 0). Data for the other dimensions are collected using the respective read-pointers and read-ranges. For example, the first duration collected from working-memory starts at the 28th event or the second event of bar 7. Consequently, this duration value appears as the duration of the first event in voice 1 of the reference-melody.

The data in voice 1 now serves as input to the grouping and de-grouping algorithms as a function of the levels and sign of the activation of the remaining agents in the present agents cluster, similar as in the preceding examples. Notice that the activation of agent 8 is only 4 – therefore it will not contribute additional events. Also notice that all potential events are actually realised because both agent activation and energy are relatively high. Finally, notice that all events in the parallel voice synchronise in the absence of delays in the (de)grouping algorithms parameter specifications.

This example shows how fragmented data from four dimensions in the source melody (working-memory) merges into a new machine response. Parametric aspects of working-memory are thus reflected in the resulting melody eventually further modified by the CF's processing functions. Therefore, the machine response echoes certain features of the source melody in non-trivial ways.

Chapter 11: Conclusion

11.1 Introduction: Thesis Review and Discussion

In a television interview on Dutch TV, while referring to the massive variations in size and shape of shells of land snails[6] Harvard palaeontologist Stephen Jay Gould defined beauty as an enjoyment of evolution-based variations and change; "For this is beauty for evolutionary biologists because we love diversity" (Kayser 2000). The important point is that diversity happens to be manifest on a given *continuous* scale stretching in many dimensions. In Oscar, maximisation of diversity (chapter 1, section 1.2.1) is supported explicitly from the appreciation of change as a first principle (algorithmic activity driven by changes rather than absolute data levels) and by the application of evolutionary transformers to maintain diversity in the system's compound-function pool. In addition, the continuous measurement of human-machine melodic *similarity* is a direct implementation of diversity aware processing.

This thesis has highlighted a networked software entity built on the assumption that rewarding musical improvisation flourishes from the approval of *diversity*. The general methodology of Artificial Life was explored to initiate evolution and adaptation in real-time man-machine improvisation. The implementation and the ensuing experiments provided us with a platform to try-out complex ideas in a concrete way rather than by abstract contemplation. Quite often, the relationship between internal and external activity remained utterly blurred, yet a sense of non-trivial communication emerged from the confrontation of the system's autonomy facing external disturbance. This observation conforms with what follows; Dutch virtuoso improviser Misha Mengelberg commenting on his work with fellow musician Han Bennink: "I would not know what Bennink means with his music, but

[6] Shells of *Cerion Excelsior*.

when our misunderstandings are combined we think sometimes that things are fitting, sometimes complementary" (cited in Bailey 1980, p.144). Interaction with Oscar may be described along similar lines. However, great effort was taken in an attempt to disclose the underlying processes that give rise to such types of non-standard interaction.

The work reported in this thesis consisted of bringing life-like qualities to the field of interactive composing following a general Artificial Life inspired approach. In contrast to more traditional approaches, the system developed here deals with continuous social pressures and the construction of machine opinions of how to respond to internal motivations in relation to external influences. The main forces shaping coherent overall behaviour were obtained by having the three key activities (perception, action and motivation synthesis) spread out in a networked architecture.

In order to test the viability of the approach, a large number of systematic experiments were carried out. The experimental data allowed comparing the impact of various systems parameter settings in detail.

This research required the development of a new interaction model to adequately support the intended life-like interaction protocol as outlined in chapter 3. The model was designed to foster the spontaneous development of complex, internal spatiotemporal patterns *and* to support the accommodation of qualitative, external human participation. Once implemented, the model allowed for empirical testing; it was discovered that machine output was indeed strongly influenced by the motivation generator. The model developed strong preferences of how to respond to impinging sensory information and to develop the appropriate musical processing functions in order to successfully comply to a given machine motivation. The general use of evolutionary algorithms allowed for gradual optimisation of all key system

components. However, it was discovered that evolution viewed as *a process of the accumulation of small change*, as formalised by Dawkins (1986), is really an idealised theory. Experiments revealed highly irregular changes in an otherwise fairly incremental fitness history, which hints to the conclusion that evolution can indeed be considered a complex dynamical system in itself. The eminent connection between the concepts of playing "on the edge" in open improvisation and computation "on the edge" of chaos were outlined in chapter 1, section 1.3.8.

11.2 Contributions to Knowledge

The contributions to knowledge of this thesis are summarised below in four sections. The first section indicates how this thesis answered the two overarching questions posed in chapter 1, section 1.2.1. Then, we briefly review how Oscar meets the four main evaluation criteria introduced in section 1.2.2, also in Chapter 1. In the third section we demonstrate how Oscar fulfils a lacuna in the field of interactive music systems' design, by showing how it compares to existing systems in terms of the comparative design criteria (or framework) introduced in Chapter 2, section 2.4. Finally, we briefly revisit Oscar's musical output, by commenting on the rendering of music from its behaviour.

11.2.1 Answers to Overarching Research Questions

The overarching questions this thesis aimed to answer were:

A) Is it possible to develop an interactive system that is able to modify itself in coherent, but non-trivial ways, as the result from interactions with the external world (e.g., with a human interactor)?

B) Would such system be able to offer an ongoing interaction platform, which remains interesting over extended time spans?

The answer to the first question is affirmative because Oscar's networked architecture implicitly supports internal changes while its intended functionality remains intact. As a first example, consider the listening section, it reconfigures its sensor-network as to become gradually more adapted to a given human interactor. These changes are non-trivial because they occur for the purpose of optimisation and are driven by an implicit fitness function e.g.; the maximisation of listening sensitivity. Instances of this process are seen in figures 4.32 to 4.35. As a second example, consider the player agents; they also reconfigure themselves continuously into structural clusters as a function of mutual social affinities. Figures 7.14 and 7.15 clearly show non-trivial changes occurring from one time frame to the next. The experiments documented in chapter 10, section 10.5 and 10.6, document the effect of these structural changes on the actual music produced. As a third example, consider the successful manipulation of the compound-function pool in experiment e3 as shown in figures 10.3 and 10.4; the genetic programming technique evolves fitter functions over time. Also, the data demonstrates how the compound-function pool articulates particular expertise for the purpose of integration and expression over time. As a final example, let us address the macroscopic oscillations revealed by the polynomials in experiments e7 to e10. For instance, consider figure 10.35 (exploitation vs. exploration pressures in experiment e7), figure 10.49 (drives-pool output levels in experiment e7) or figure 10.51 (drives pool understanding level in experiment e7) – they all show non-trivial, gradually oscillating tendencies rather than random drifting.

The answer to the second questions is also affirmative because the degree of interestingness is related to the notion of persistent rewarding interaction. Interaction remains interesting over longer time spans because the evolutionary approach

cultivates fitness levels according to information gained during the time span of one epoch of actual human-machine interaction. Then, the process of reproduction merges aspects of the functionality that led to the current situation, breeding new software modules incorporating features of the past with novelty. This process repeats forever because a single optimal solution does not exist *and* input from the human performer remains totally unpredictable. Therefore, the breeding procedure is a process of both perpetual renewal and perpetual adaptation. Human-machine interaction remains interesting over longer time spans because it renews itself in a continuous process blending recognition and surprise. A wider discussion on the link between coherence and variation in behaviour follows in section 11.2.2 where it is considered in the light of our criteria for evaluation.

11.2.2 Matching the Evaluation Criteria

11.2.2.1 Maximization of External (Human) Influence

According to this criterion, the system should maximise human influence in the process of machine listening, be sensitive to a broad variety of features in the input signal and a global indication of "listening" should materialize from those features. Listening is viewed as an active process of gradual optimisation, maximizing sensitivity and diversity. The networked listening module (network A in figure 1.1, chapter 1) implements listening as a dynamic process consisting of two interacting components: a sensor-network and a patch.

The experiments documented in figures 4.32 to 4.35 (chapter 4) present clear evidence of incremental fitness levels that reflects a steady increase in listening sensitivity over time. In addition, the experiments documented in figures 4.36 and 4.37 equally show that neurons in the sensor networks, on the average, exhibit

incremental firing rates over time. In other words, sensor networks manage to become more sensitive as they evolve, thus optimizing responsiveness to external input.

A wide variety of sensors were developed in order to accommodate a broad range of features in the MIDI input stream. Many sensors do not handle static categorisation algorithms but are adaptive and thus also contribute to the objective implied in this criterion i.e., the optimisation of sensitivity to human influence. Figures 4.3a to 4.3f document how sensor networks develop context sensitive behaviour – the sensor discrimination window (reflecting sensitivity) adapts as a function of the dynamics of the input signal.

The patch object in combination with the sensor-network can be viewed as a qualitative oscillator consistent with the view that listening is a dynamic process. The relationships operational in a patch provide a qualitative, non-linear reduction of sensor information in many dimensions into a single output vector. This vector holds signed quantities reflecting the impact of changes (in contrast to static information samples) in the neural firing patterns of the sensor-network. Therefore, the patch object presents a continuous, qualitative and unified perspective on a hybrid mix of complex, parallel channels of influences as accommodated by the sensor-network.

It should be mentioned that the genetic optimisation process acting on the relationships inside a patch generated unanticipated results; the patch fitness levels did not show the expected gradual incremental data profile. A plausible conclusion is that (1) the effect of multiple relationships accumulates in non-linear ways leading to unpredictable behaviour and (2) the genetic process is itself inherently irregular and deviates, from its idealised conception i.e., as a process of gradual optimisation. However, in summary, the intent put forward in the first criterion has been accomplished successfully.

11.2.2.2 Blending of Mutual Influences

The second criterion states that temporal complexity and novelty should emerge by blending influence from a human performer and pressure generated by internal system components. The system's internal organization must accommodate mutual influences between musical objects in a distributed system – the articulation of larger, multi-layered musical output structures should emerge from the interactions between more basic musical processing units. In essence, when creating a response, the system must negotiate influence from internal and external information resources.

A distributed, A-life oriented model was developed, taking inspiration from models of social interaction between people (Dewdney 1987, Gold 2007) and physical interaction between molecules (Hofstadter 1995). It implements the player agency developed in this thesis. Agents are organised in a spatial 2D world, they are sensitive to three information resources (1) the external human performer, (2) neighbour agents that are spatially close to the agent and (3) information that is held privately inside every agent. All three resources are in fact competing to be applied and the dynamics of this process is one reason for the great variety in the systems' musical output. The principle of multiple influences (chapter 6, section 6.4) is implemented as an algorithm to critically merge influence from two sources in view of the construction of a machine response. The algorithm allows for quantitative control of the amount of information taken from the respective sources (figure 6.7) as shown, for example in detail, in the musical scores documenting experiments e7 and e8 (chapter 10).

The pilot experiments documented in figures 7.16 and 7.17 (chapter 7) reveal interesting spatiotemporal structures, the agency shows self-organising behaviour and oscillations of variable complexity, from high-periodicity oscillations to complex

quasi-periodic oscillations to chaos. These patterns are not directly sonified since – as explained in the introduction – we avoid the notion of mapping. The agency continuously generates variable arrangements of agents (referred to as clusters) and thus supports the synthesis of novelty in a coherent and organised way.

The player agency further implements the blending of influences from two different perspectives. Firstly, the patch object sends activation signals to the agents; their levels will influence the interpretation of a given agents cluster and influence the synthesis of complex, rhythmically interlocking polyphonic structures. Secondly, the complex-functions pool distributes musical processing functions to every agent according to a distribution scheme conditioned by the system's global motivation. This type of double influence (in terms of activation and function) is depicted in chapter 7, figure 7.1.

The global musical result is thus qualitatively conditioned by (1) internal social forces guaranteeing perpetual renewal of the agency's internal structural organization, (2) the sharing of partial data from competing information resources and (3) a method to influence the strength and the quality (musical functionality) of the contribution of every agent to the developing musical fabric. Therefore, the objectives implied by the second criterion have been met.

11.2.2.3 Automatic Generation of Internal Motivations

This criterion requires the system to initiate internal motivations autonomously, motivations that are instrumental in *influencing* the interaction climate in specific ways.

The drive object (chapter 8) implements an answer to this requirement by formalising two primary machine motivations: integration and expression. A drive

interprets external changes using a collection of non-linear relationships; the results accumulate as two competing pressures forcing the selection of one of the motivations. Motivations call for musical processing functions in the compound-function pool according their implied objective, integration or expression. As the musical distance between man and machine is traced continuously, we have a means to evaluate the efficiency of particular functions in view of the service they provide in relation to the goal they are aiming to attend. This method circumvents the fitness bottleneck e.g., the necessity to provide explicit fitness ratings to evaluate particular intended systems behaviour.

In addition, the learning algorithm operational in a drive collects evidence on how appropriate the current drive actually is in supporting the implied systems goal, i.e., human and machine aiming for the same orientation (either both integration or both expression). The systematic experiments e7 to e10 (chapter 10) examine the impact of learning. From the study of the correlations between figures 10.47, 10.48, 10.83 and 10.84 it was concluded that learning exercises a stabilizing effect on the integration and expression levels in a given drive, drives without learning produce wider and irregular oscillations than drives that include a learning component. The same series of experiments offer compelling evidence that Oscar adapts in order to maximize human/machine agreement (table 10.9). Further evidence is collected from the correlation analysis as documented in chapter 10, section 10.4.8; the analysis reveals a strong relationship between negative global orientation values (expression) and positive values for system common understanding – therefore, the system develops an interaction climate characterised by agreement rather than conflict.

Histogram analysis of the compound-function pool application density, for the same experiments, shows a more peaking profile when learning is on. This means that

Oscar managed to evolve a sharper opinion about what processing functions to prefer. Therefore, learning promotes the welcome yet unexpected appearance of temporal stylistic biases that most probably encourages heightened awareness and momentum in the human interactor.

The relationship between the drives pool and the compound-function pool also influences global systems behaviour in interesting ways. For example, referring to figures 10.55 and 10.56 (drives output levels) and figures 10.91 and 10.92 (compound-function pool fitness histories) we notice a consistent correlation between the polynomials in the respective data sets; the changes in the competing levels in drives pool levels are complementary to the changes in the CF-pool. This observation illustrates the tight interaction between machine motivation and machine musicality over extended time spans.

The organisation of the timing instructions active in the scheduling algorithm (as detailed in chapter 9) also reflect the selection mechanism active in a drive because timing depends on the drive's current orientation; for example, when orientation equals integration, the scheduler aims to start playing in exact synchronisation with the predicated start of the pending human input sequence (further details are in chapter 9, section 9.5).

The combined activity and interaction between drives and compound-functions offers a robust mechanism to influence the nature of human-machine interaction in interesting ways; this method offers considerable variety in machine responses while still sufficiently constraining systems complexity by way of feedback provided by the learning algorithm. In conclusion, the third criterion is attended as the drive object implements machine motivations that successfully merge autonomy and behavioural directedness.

11.2.2.4 Emergent Functionality vs. Variable Behaviour

Oscar should consistently offer coherent emergent functionality *as a whole* while still showing evidence of variable behaviour evolving in the long run. The networked model (figure 1.1) was suggested to accommodate this wish for overall behaviour merging operational consistency and stylistic variety.

The experiments in chapter 10 document the behavioural scope of the system given different conditions such as the impact of learning, breeding and systems inclination (social or selfish). For example, consider figure 10.3, it shows the relationship and progression of the fitness levels for integration and expression in a given CF-pool. It offers an unambiguous example of consistency and change in a single object that progress over 10 evolutionary epochs. Remarkably, the evolutionary process manages to steadily accumulate expression fitness over 10 breeding steps (while still being exposed to unpredictable input) after which a total breakdown in fitness arises.

Consider the emergent relationships between drives global orientation and compound-function pool fitness histories in experiments e7 and e8. In experiment e7, the relationships in the drive mainly evolve expression rather than integration as seen in figure 10.41. The compound-function pool manages to evolve musical processing functions that are primarily fit for the purpose of expression as documented in figure 10.55. This reflects autonomous emergent functionality without any need of external guidance.

The data gathered from many experiments as visualised in chapter 10 reveal a complex peaking data profile. However, the polynomials disclose various kinds of "behavioural waves". Many comparative polynomial visualisations show evidence of the competitive interaction between complementary data levels such as human vs.

machine agreement (figures 10.81 and 10.82), drives' integration and expression levels (figures 10.85 and 10.86) and CF-pool fitness histories (figures 10.91 and 10.92). These waves can be interpreted as emergent phenomena disclosing very low-level frequency oscillations in the systems' constituent networks.

Reconsider figures 10.111 to 10.114, they provide a comparative analysis of the results of the four experiments in terms of correlation between the drives global orientation and the system common understanding. With the exception of figure 10.112, all show positive correlation between agreement and expression – agreement suggests itself given oscillatory behaviour in the dimension of system global orientation. There seems to be no correlation given a system orientation of integration and negative common understanding; i.e., a situation of conflict. As explained in chapter 10, this observation proves that Oscar has more difficulty to develop an attitude of integration than an attitude of expression. Figure 10.112 shows uncorrelated data given conflict and integration. This figure, documenting an experiment with selfish inclination and learning enabled, offers the most explicit phase portrait; just two behavioural regimes. Firstly, nearly perfect correlation given positive common understanding (agreement) and negative global orientation (system prefers expression). Secondly, we observe practically no correlation at all given negative common understanding (conflict) and positive global orientation (system prefers integration). In other words, the system oscillates between behaviour based on relatively linear relationships and chaotic behaviour, in effect, another example of how Oscar merges consistency and variation.

The score examples and their analysis documented in chapter 10 provide further evidence of the mix of (1) coherent musical behaviour as articulated by the social

affinities in the player agency and (2) unexpected patterns computed from the appreciation of external input or the occurrence of evolutionary breeding.

Evolution thus both acts as a means to optimise the four system components subject to genetic optimisation *in addition* to providing potentially pivotal moments during interaction where the respective newly bred populations are introduced. The present implementation also shows that small populations offer enough critical mass to evolve prospective systems objects. In conclusion, the goal implied in this criterion has been achieved.

11.2.3 Contextual Evaluation and Fulfilment of a Lacuna in the Field

	Oscar
Paradigm	System
Idiomatic inclination	Non-idiomatic
Objective	Improvisation
Learning capability	Yes
Evolution capability	Yes
Complexity	Emergent
Autonomy	Motivation generator
Agents paradigm	Player agency
Generative paradigm	Evolutionary
Sensing approach	Adaptive, evolutionary

Table 11.1: Tabulation of comparative design criteria for Oscar (relating to the information in tables 2.3 and 2.4).

Referring to the comparative design criteria developed in chapter 2, this section (1) presents a brief critical discussion of Oscar in relation to the systems presented in table 2.3 and table 2.4 and (2) arguments how Oscar fulfils a lacuna in the field of interactive computer music systems.

11.2.3.1. Critical Evaluation of Contextual Systems

Cypher's design methodology concentrates on symbolic representations while Oscar resorts to mixture of symbolic and subsymbolic computing. It is instructive to compare the two systems as both support non-idiomatic interactions. In contrast to Oscar, sensors in Cypher do not evolve and interactions are hand-designed protocols; a human designer implements prospective action-reaction schemes. Such schemes may be saved and loaded from disk in order to provide Cypher with a specific responsive musical identity. Oscar has no access to such pre-defined conditioning information, an interactive session starts from scratch and *all* internal information is acquired and updated within the process of interaction itself.

Oscar minimises static internal representations (memory structures) and avoids symbolic reasoning although not its complete elimination. Oscar maximises functional representations throughout, for example, by using the agility of non-linear relationships (chapter 8). Experiments e9 and e10 in chapter 10 reveal the unplanned emergence of purposeful behaviour. A strong positive correlation exists between the drives-pool integration and expression levels (figures 10.85 and 10.86) and system common understanding (figures 10.99 and 10.100). The extended incremental history of both data profiles shows that man and machine may indeed evolve towards the same type of interaction activity, in this particular case, both parties primarily aim to express rather than to integrate.

Still, Cypher (and Voyager) can be characterised as relatively "open" systems as they do not promote stylistic biases and offer interesting augmented complexity from the combinatorial explosion of their respective rule bases. In effect, the generation of complex behaviour in the systems in table 2.2 is a consequence of the activation of "designed" action-reaction associations. Input signals may trigger responses whose

complexity results from mapping many parallel rules to many sound conditioning parameters also addressed in parallel. The results often sound "complex" and "unpredictable" because the mapping logic is too complex to be deciphered in real-time by a human observer. Notice that the notion of complexity has a different meaning considering the systems in table 2.3; complexity in the A-life oriented systems typically results from self-organization.

In contrast to Oscar, Cypher does not learn nor evolve, so it cannot modify the way it performs from the consequences of the act of interaction itself. Voyager includes simple statistical learning (histograms) in its listening section. The Continuator uses a far more sophisticated method based on augmented Markov chains (chapter 2). However, learning remains confined to the acquisition of information of how the external human interactor behaves. In contrast to Oscar, learning is not used to improve systems behaviour over time. For example, experiments e9 and e10 prove that the inclusion of a learning component has a positive impact on the synthesis of machine motivations.

A certain type of automatist production is possible in Cypher; in the absence of human input, Cypher will generate musical variations by feeding its output back into its listening section. In contrast to Oscar, true autonomy is not achievable because Cypher does not generate internal criteria conditioning the progression of its internal decision-making.

A discrete number of performance modes in Cypher also follow from the application of specific action-reaction scripts and the nature and configuration of the musical processing functions. By *discrete* it is understood that interaction develops as conditioned activity in a succession of time frames, in fact, reminiscent of the narrative interaction schemes devised by Subotnick and Coniglio (Rowe 1992). Oscar

does not impose specific segmented performance modes but leaves the impression of particular transitions as the emergence of state transitions in the behaviour of its internal networks. By definition, such transitions are not planned but materialize spontaneously from a myriad of forces that happen to animate a given interaction context. As a result, Oscar articulates time in a continuous fashion rather than in isolated segments. A turning point during human-machine negotiation signalling surprise and potential confusion in the human interactor is essentially considered emergent functionality. For example, consider the sudden introduction of a different clustering regime in the agency as reflected at measure 147 of the score in chapter 10, section 10.6. The unpredictability of such potentially pivotal moments in man-machine interaction where man and machine are questioning their mutual behaviour is considered a vital attribute of rewarding interaction.

The Continuator mainly follows a call-and-response paradigm that cannot, by definition, exhibit unanticipated behaviour.

Strictly speaking, both GenJam implementations also severely limit the operational freedom of a human interactor since explicit stylistic databases confine general systems behaviour. Whatever the input of the human interactor, GenJam will produce consistent stylistic output. One might even conclude that the exploratory nature and the "divergence" implied in the application of genetic algorithms in GenJam is essentially in conflict with the "convergence" imposed by its stylistic biases. GenJam version 2 circumvents the fitness bottleneck by eliminating fitness altogether, so that version no longer qualifies as a genetic system. Oscar offers an original solution to the problem of attributing fitness to machine generated melodies by suggesting an *implicit* fitness rating. The success of particular musical processing functions is tied to the context in which they are used; one tracks changes in musical

distance between man and machine and fitness is then related to the current machine objective, integration or expression, as described in chapter 8.

Both Oscar and the Social Robots project employ a type of reinforcement learning, Oscar to reward successful motivations, the Social Robots project to reward successful imitations.

The A-life oriented systems in table 2.3 explore a form of emergence. Diseases Squared is unique as it models a form of virtual biology involving organisms evolving over many generations. Interesting dynamics emerges primarily because the disease and the virtual organisms co-evolve and continuously adapt in order to survive. In addition, a wide variety of organisms emerge through the process of bio-diversity forced by evolutionary pressure. Both Swarm systems and Audible Ecosystems rely on self-organization. Swarm follows the flocking model formalised by Reynolds (1987). Musical output results from the application of a 3-part mapping function incorporating functionality for listening, reasoning and responding. Human input may function as a temporary focus for the flock. The design rationale of the Swarm systems implies that the relationships between input and output remain sufficiently transparent to the human interactor, which in turn implies responsive rather than interactive behaviour. In contrast, Oscar allows for the on-line development of open-ended man-machine relationships with many degrees of understanding, awareness and potential confusion in many dimensions. Emergence in Audible Ecosystems follows from the exploration of non-linear feedback of sound as a function of the acoustics of a given performance space and the use of feedback in its sound processing functions. Eden and the Cellular Automata projects suggest complex behaviour in virtual worlds organised as a discrete, regular topology of cells. Emergence here issues from the consistent application of a local rule interpreting the neighbourhood of every cell. As

explained in chapter 1, section 1.3, in Oscar, emergent functionality is operational for the purpose of machine listening, playing machine responses and in the motivation generator. The experiments in chapter 10 and the interpretation of the experimental data in section 10.4.6 reveals interesting correlations between the various datasets – this leads to the conclusion that intended functional behaviour indeed emerges from interactions between the three basic networks in our systems architecture (figure 1.1).

11.2.3.2. Fulfilment of a Lacuna in the Field

As explained in chapter 1, section 1.3.1, autonomy is a highly desired property of interactive music systems because (1) it guarantees the generation of unexpected responses that are however tightly connected to the current context in musically interesting complex ways, (2) it allows interactive systems to develop unpredictable yet coherent behaviour from the synthesis of internal motivations.

As opposed to the systems in tables 2.2 and 2.3, Oscar offers unique functionality on the following levels:

1) A combination of learning and evolution to support a form of true autonomy; the motivation generator.

2) The use of genetic programming to evolve the appropriate musical processing functions to conform to a specific motivation.

3) Sensing approach: sensor diversity, adaptation and optimisation.

Referring to the data in tables 2.2 and 2.3, three systems feature a form of autonomy; Voyager, Social Robots and Audible Ecosystems. In Voyager, autonomy is considered very low since the system only provides a weak impression of autonomy because seemingly, goal-directed behaviour issues from the conditioning of its probability distributions. The various data arrays in Voyager are continuously

modified by incoming data. Therefore, when these arrays are addressed to supply data for creation of machine responses, an apparent link can be detected between what the system hears and how it reacts. However, global behaviour in Voyager does *not* progress in any particular direction because there is no higher-level arrangement to specify motivations. The Social Robots project offers a conceptually higher form of autonomy. As the robots dynamically interact, they adjust their data structures in a wish to develop a repertoire of common songs. The requirement to build up a common repertoire is a goal specification; such higher-level goals are absent in Voyager. The robots manage to solve the problem autonomously using reinforcement learning. In terms of emergence and autonomy, Voyager and Audible Ecosystems are related, however, Audible Ecosystems is more multifaceted since its complexity follows from the confrontation of an "internal" complex dynamical system (its digital signal processing networks) and an "external" system made up of the complex acoustics of a given space. Systems like Voyager and Audible Ecosystems are appreciated from the experience of the complexity of their behaviour as it develops in real-time. However, this behavioural complexity does not offer a hint of the expression of higher-level machine motivations: the general feeling is that of relatively unpredictable, evolving episodic complexity – there is no pressure to accomplish a specific agenda; consequently there is no motivation. In conclusion, of the ten contextual systems (tables 2.2 and 2.3) only the Social Robots develop autonomous behaviour (however, without the participation of human interactors), therefore, the notion of autonomy is generally not addressed in the current practice of interactive music systems design. Oscar fulfils a lacuna in this sense because, as explained in chapter 8, the system develops by itself a policy to regulate its own behaviour from the consideration of two competing internal motivations: integration

and expression. Oscar evolves populations of musical processing functions (chapter 6) fit to accomplish its current motivation; co-evolution of motivation and function is a unique feature of the system developed here.

Let us address the sensing approach in tables 2.2 and 2.3. We focus on the notions of adaptation, optimisation and diversity of perspectives in sensing. Sensing in Cypher is considered "minimally adaptive" since only the pitch discrimination window modifies itself in relation to the data it is currently processing (chapter 2, section 2.2.1). In Cypher, sensor hierarchies are created explicitly and remain fixed as the system is running. Oscar creates listening networks of variable complexity and orientation (selection of specific sensors) and applies a genetic algorithm to optimise listening sensitivity in the long run (chapter 4). Sensing in Audible Ecosystems is "implicitly adaptive" since its rationale is to explore sonic complexity generated by audio feedback; the system is intended to function "on the edge of chaos" (chapter 1, section 1.3.2), therefore its sensors must adapt in order to cope with momentary under- and over stimulation. Only Oscar integrates the following listening qualities deemed vital for interactive music systems: (1) diversity in listening perspective by using different sensor categories (chapter 4, section 4.4), (2) short term adaptation in low level, single event, single dimension sensors (section 4.5.1.1) and (3) long term optimisation of listening networks using a genetic technique (section 4.8).

In conclusion, Oscar contributes original methods on two levels: firstly, the support of genuine autonomous behaviour by evolving musical processing functions in relation to machine motivations, and secondly, the introduction of diversity, adaptability and optimisation in the listening process.

11.2.4 Musical Output

This section briefly reviews how musical output embodies Oscar's behaviour; we address the musical score analysis in experiment e7 (chapter 10, section 10.5) and experiment e8 (section 10.6). Analysis confirms that Oscar accommodates rewarding interaction (chapter 1, section 1.3.2); the system develops independent complex autonomous behaviour while, at the same time, activity by a human performer leaves a discernable impact as the music unfolds. For example, the score in experiment e7 clearly reflects the effect of the consecutive changes in clustering activity inside the player agency while aspects of human input (such as particular pitch intervals and rhythmic patterns) appear in modified form in the score. The principle of multiple influences (chapter 6, section 6.4) effectively implements our wish to merge aspects of internally generated complexity and external influence in the musical rendering process.

The score excerpts of experiment e8 are analysed in terms of the global effect of ten systems quantities; we provide a comprehensive analysis of typical systems behaviour at three moments in time. We show how a particular temporal system state – reflected in specific numerical data subject to analysis – translates to unambiguous musical output. Excerpt nr. 1 (section 10.6.1) details the effect of the agents' clustering, the selection of responsive or autonomous mode as a function of human-responsiveness, the selection of source material as function of the input-deliberation-vector and the effects of the agents' energy and activation on the creation of the reference-melody. Excerpt nr. 2 demonstrates the effect of the compound-function currently assigned to the agents in the playing cluster and how the current agents' activation level contributes additional, parallel events to the reference-melody. The mixed effect of grouping and de-grouping algorithms merges in a coherent though

complex polyphonic musical fabric as shown in figure 10.140. Excerpt nr. 3 details a situation where human-responsiveness is very high, this generates exclusive pressure to address only working memory (i.e. the current musical context made available by the human interactor) to create a reference-melody. Figure 10.141 shows a dense score; all potential events are realised since the energy and activation levels of all contributing agents is very high. In addition, the score in figure 10.141 provides evidence that complex machine responses may be generated reflecting source material in musically interesting, non-trivial ways. In conclusion, we have shown that the music-rendering approach in Oscar contributes to the achievement of rewarding interaction as defined in chapter 1, section 1.3.2.

11.4 Recommendations for future research

As made clear in the introduction, it is still difficult to provide a unified, stable definition of the relationships between the nature of interaction and improvisation. However, while drawing on Oscar as a pragmatic platform for research on the brink of interaction and improvisation, we discovered many interesting traits in systems behaviour of Oscar's internal networked organization in confrontation with an external interactor. The various other systems studied in this thesis offered an articulated context for discussion and evaluation though, most importantly, they substantiate the vast cultural and scientific diversity in approaching interaction and improvisation.

One direction for further research would be to evolve much larger populations of objects than presented here and perhaps evaluate them in parallel using multiprocessor hardware. Conceivably, a system of improved plasticity would result when agents would be sensitive not just to social forces from other agents and activation by a human performer but could also develop a capacity to be sensitive to

the variable complexity of the spatiotemporal structures emerging from their spatial organization.

The introduction of direct audio sensing rather than indirect sensing via MIDI could possibly offer a tighter, more intimate connection between human and machine players. For instance, one could apply our working principles, including learning and evolution, to a distributed population of real-time digital signal processing modules. For example, the complementarities of human input and machine output on the audio sample level could result in more profound musical human-machine associations. In addition, when a machine develops adequate sensitivities (for the purpose of analysis and production) (1) to appreciate microscopic changes in audio and (2) a sense for understanding larger macroscopic structures, it would bring machine musicianship closer to the human experience of music.

However, in the spirit of A-life and as stated in the introduction, the work reported here does not aspire to impersonate any known formal model of what musical improvisation should be although our system is definitely rooted in the dominantly Western opinion of non-idiomatic musical improvisation. In this light, it would be utterly meaningful to have experimental interactive music systems that develop sensitivities for music emanating from a wider diversity of world cultures.

Innovative forms of improvisation are further conceivable when the principles of social affinities amongst agents and evolution are implemented in global networks perhaps involving forms of multi-modal audiovisual interaction. At that point, improvisation becomes a dynamic tool supporting the critical analysis of the organization of human culture far beyond the appreciation of music as a human discipline addressed in isolation. As further research in interactive music systems may be instrumental to come across innovative forms of human-machine negotiation from

technological and aesthetic points of view, it may equally generate social awareness from a much wider perspective.

Glossary

Adaptation

Adaptation refers to the process whereby an organism gradually becomes better tailored facing the pressures of its environment. Adaptation affects both the morphology and the behaviour of an organism. Adaptation is thought of in two ways; (1) adaptation by way of long term evolution over many evolutionary epochs of a given species and (2) adaptation by way of learning during the life span of an organism. Both instances of adaptation are operational in Oscar's *drive* object.

Agent

In general, agent refers to a computational structure (materialised in software or hardware) endowed with local intelligence. Agents may act with certain autonomy within a specialised domain of expertise represented as procedural knowledge in the form of simple rules. However, when many agents are configured in an agency, sophisticated overall behaviour results because the execution of simple, local rules give way to global, complex emergent functionality. The terminology for software agents followed from the cognitive theory developed by Marvin Minsky in his book *The Society of Mind.* Much fundamental work in hardware agents refers to the domain of mobile robotics. This thesis views agents as virtual players configured in a player agency. Agents express social relationships to one another, which makes them move in two-dimensional space. In addition, agents assemble themselves in spatiotemporal "clusters"; structures that we interpret as *emergent* structural couplings between simple, basic building blocks.

Artificial life

The Biology inspired, scientific discipline of Artificial Life (A-life) is concerned with the study, analysis and design of systems exhibiting self-organisation and *adaptive* behaviour arising spontaneously by *emergence* rather than explicit design.

Autonomous behaviour

Autonomous systems develop for themselves, the laws and strategies according to which they regulate their behaviour. Autonomous systems contrast with automatic systems that behave only according prescribed rules. Autonomous systems are typically grounded in a given dynamic environment and adapt their internal structure in order to guarantee sustained functionality. In the present thesis, the co-evolution of machine *motivations* and *compound functions* provides for autonomous systems behaviour.

Complexity

Complexity is a qualitative feature of systems composed of many locally interacting components that organise themselves into spatiotemporal structures of critical composition – structures in between relative regularity and total irregularity: a point of balance known as "the edge of chaos". Complexity gives rise to *emergence.*

Compound-function

A compound-function consists of a series of simple musical processing functions, all holding parametric specifications. The individual functions and their associated arguments are subject to genetic optimisation using a method of *genetic programming.*

Crossover

Crossover and *mutation* are the two basic genetic operators. In the context of *genetic algorithms*, crossover creates a new *genotype* from sub-sequences of genotype extracted from both parents. In the context of *genetic programming*, crossover typically creates a new nested functional tree structure that merges programming functionality from two parent structures.

Derivative

Derivative is defined in terms of its conventional meaning in mathematics. For example, the first derivative of a number sequence is equivalent to its rate of change, that is, the value and sign of the difference (intervals) between consecutive values.

Distributed system

A distributed system consists of a collection of interrelated computational components, often referred to as *agents*.

Drive

A drive represents a machine *motivation* in a computational structure. A drive holds two 12-bit binary vectors – one for each basic motivation; *integration* and *expression*. Every "on" bit instructs the drive to accommodate specific external changes such as changes in (1) quality and (2) quantity of the human input stream in addition to (3) changes in musical distance between the most recent human and machine produced melodies. Changes are interpreted by means of internal *relationships*.

Emergence

Emergence generally denotes the creation of global complexity from simple interactions between basic building blocks, complexity that cannot be explained from consideration of those building blocks in isolation. Within this thesis, emergence is a multi-facetted concept, however, it is characterised according to two fundamental meanings; emergence as morphology and emergence as functionality. Firstly, emergence refers to the creation of complex, spatiotemporal structures arising from the simple interactions between computational objects known as *agents*. For example, player *agents* assemble into spatiotemporal clusters according to mutual social affinities. Secondly, emergence refers to the creation of spontaneous global functionality arising from favourable interactions between different system modules. The apparent cooperative behaviour between the creation of machine motivations and the creation of musical processing functions to fulfil those motivations is an example of emergent functionality.

Entropy

Entropy is generally referred to as a measure of the degree of disorder in a given system though our definition is more specific. We use entropy to measure continuous melodic complexity based on the degree of predictability of a given musical event to the next. The entropy is then proportional to the amount of surprise it generates.

Exploitation

Oscar evolves populations of four types of objects: *sensor networks* and *patch* objects for the purpose of listening, *drive* objects that represent machine motivations (*integration* or *expression*) and musical processing functions that provide assistance in attaining a particular momentary *motivation*. All objects keep a fitness level documenting past performance efficiency. Exploitation means that object selection is

totally based on the appreciation of current fitness levels – without considering objects that did not manage to accumulate efficiency in the past. Exploitation is complementary to *exploration.*

Exploration

In contrast to *exploitation*, explorative object selection from a given population prefers to promote untried objects hoping they might contribute particular intended functionality.

Expression

Expression is a basic machine *motivation* and is complementary with *integration.* Expression means that the system is aiming to convey a private musical personality, irrespective of the present musical context (as present in *working memory*) suggested by the human performer.

Evolutionary computing

A computational method inspired by biological evolution as found in nature. Evolutionary computing aims to solve problems through the application of evolution rather than explicit design. Two main approaches exist: *genetic algorithms* and *genetic programming.*

Fitness

In nature, the fitness of an organism is proportional to how well it adapts to its environment. In computer simulations of life-like processes, for instance when using *genetic algorithms*, the meaning of fitness is arbitrary and decided by the system's designer.

Genotype

Genotype is internal, coded information found in the cells of living organisms. Genotype holds a series of critical instructions that are inherited in the process of reproduction. The genotype instructions are interpreted by cells to create a physical manifestation: a *phenotype.*

Genetic algorithm

A genetic algorithm is a computer representation mimicking genetic optimisation as witnessed in nature. A given problem is analysed and typically represented as a binary sequence (known as a *genotype*) documenting all possible parameter settings of the implied search space. In nature, survival and reproduction are the implicit *fitness* criteria. In computer simulations, fitness may depend on an arbitrary assigned goal.

Genetic programming

Genetic programming views algorithms and their implementation in computer programs as subject to genetic manipulation using the standard operators of *crossover* and *mutation.*

Identity

In the context of this thesis, the identity of a system is related to the apparent complexity of its behaviour. Identity is reflected from the perception of specific spatiotemporal patterns documenting a non-random, seemingly *emergent* goal-directed behaviour.

Integration

Integration (complementary to *expression*) is a basic machine motivation, its strength being represented by a scalar quantity (0 to 100). When the system aims for integration, the objective is to close the musical distance between a human suggested context (as present in *working memory*) and the current machine output. The system has access to a population of *compound-functions* that offer musical functionality to fulfil a particular goal such as integration.

Interaction

Human-machine interaction is defined as a process of reciprocal action, a process of mutual influence. This definition puts human and machine on equal levels of authority; machine behaviour cannot be controlled but only influenced. The aesthetic orientation of this thesis is interaction as *non-idiomatic improvisation.*

Interactive composing

Interactive composing is a two-stage process: one first creates a program equipped with listening and playing modules, the functionality of both being determined by the logic embedded in the respective software modules. At a later stage, a live musician interacts with the program, the musical flow and dynamics of the interaction being conditioned by how man and machine provide mutual influence. Relationships between human input and machine responses are usually organised as *mappings*.

Mapping

In *interactive composing*, mapping refers to the creation of an associative connection between human actions and machine responses. Mappings are typically built from explicit rules (if-then structures), the if-part identifies a feature of the input

signal, the then-part triggers consequent machine activity such as playing a specific sound. Mapping methods are considered problematic because they are not adaptive; they only offer static call-and-response behaviour and cannot adapt to large changes in context. In addition, mappings cannot deal with input material that was not anticipated by the programmer.

Melody

Melody is a computational object holding a series of time-stamped MIDI events. Events are identified by five parameters: start-time, pitch, velocity (loudness), duration and MIDI channel. The melody object is a subclass of the Sequence object found in Common Music (Taube 2005).

Motivation

Oscar maintains two competing levels of machine motivations: *integration* and *expression*. The winning level determines the current machine motivation. Oscar keeps a population of motivations represented as *drive* objects.

Mutation

Mutation is a basic genetic operator. In the context of *genetic algorithms*, a small amount of mutation is applied to genotypes in order to maintain sufficient variation in a given population, that is, as a means to avoid convergence to uniformity.

Network

A network is an assembly of (often simple) computational components that communicate through non-linear connections.

Non-idiomatic improvisation

Also called "open-improvisation", non-idiomatic improvisation has no stylistic commitment. In contrast, idiomatic improvisation typically takes place within a confined musical framework of pre-established directives, for instance, a lead sheet or turn-taking in standard jazz improvisation. Non-idiomatic improvisation typically aims to propose maximum diversity consisting of many concurrent musical options. Musical structure is said to surface from improvised interactions rather than resulting from a formal, predefined framework of musical references.

Object-oriented programming

With object-oriented programming (OOP), a computer program is conceptualized and built as a collection of dynamic software components referred to as "objects". Objects mutually communicate by way of exchanging messages, hence the concept of computing by "message passing". Object-oriented programming has many advantages over functional programming, for instance, procedures and data are captured in a single software structure. An important feature of OOP is inheritance whereby software complexity is managed as a hierarchy of classes; specialised subclasses inherit expertise from higher super classes. Oscar is written in a symbolic language (LISP) and in a style of object-oriented programming.

Pattern

The word "pattern" is primarily used with the meaning it receives in the scientific domain of complex dynamical systems. Such systems typically consist of many small, interacting components that assemble themselves into typical spatiotemporal patterns. Specific regularities in a given pattern articulate the *identity* of the complex dynamical system they reflect.

Personality

The personality of a system is equivalent to its *identity*. From a general systems point of view, personality is tightly linked to the way a system modifies its behaviour and/or morphology over time. On a smaller scale, the musical "personality" (or stylistic character) of a given melody issues from the nature of the specific basic building blocks used to generate that melody; i.e. pitch intervals, list of velocity values, durations and inter-onset-times.

Phenotype

Phenotype is the physical appearance of an organism, the structure of its metabolism and behaviour. Phenotype is complementary with *genotype*.

Prediction

Oscar is equipped with a prediction algorithm in order to collect evidence at what point in time the human performer will either (1) start playing a future musical statement (if he is currently silent) or (2) at what point in time the human performer will finish the current input sequence (in case he is currently playing). This information is instrumental in scheduling machine responses as a function of the current orientation of the *drive* presently in use.

Real-time system

Real-time performance means that no perceptual delay exists between the computation of a musical response and that response being performed.

Reinforcement learning

Reinforcement learning (RL) is a non-supervised machine learning method based on systematic trial-and-error. An *agent* makes random actions and gathers feedback, a

scalar reinforcement signal, as to how successful that action was in achieving an implied goal. The agent must develop a policy (a mapping between agent states and actions) that optimises the reinforcement signal in the long run.

Relationship

In the context of the present thesis, a relationship is defined as a non-linear coupling between changes in an input quantity and the level of an output quantity. Four different types of relationships exist (fully documented in chapter 5). Many individual, concurrent relationships are typically assembled into larger structures. The patch and drive objects are examples of such computational structures. The basic objective of a relationship is to accommodate the impact of external changes into an internal numerical representation.

Rewarding interaction

Rewarding interaction results when a live performer can exercise a discernable impact on a given system *and* when that system holds enough dynamic relationships amongst its components to guarantee *complex* behaviour.

Segmentation

Segmentation refers to the intricate activity of extracting individual sub-sequences from a continuous data stream. Segmentation in real-time systems is difficult because specific sequence boundaries depend on a variable musical context. Principles of Gestalt psychology are typically used to collect evidence for the existence of particular discontinuities.

Self-organization

Self-organisation generally refers to the spontaneous, seemingly purposeful formation of complex spatiotemporal patterns in distributed systems consisting of many interacting components. Self-organisation is studied in many domains, from biology and physics to cybernetics. A system must hold a sufficient number of components for self-organisation to occur and the effect of behavioural changes must feedback into the global system. Self-organisation often gives rise to *emergence*.

Sensor

A sensor is a software module that interprets an input value and, for instance, decides whether that value is considered high or low. Many sensors take a decision relative to a given context; the dynamic range of the context provides a threshold for making a decision. The complexity of sensors varies widely, from simple single event, single dimension sensors to complex higher-level sensors such as the ones addressing entropy in a given signal. The global effect of many sensors is typically accommodated in a sensor network.

Sequence

Sequence refers to a chronological collection of musical events. The terminology is mostly used to signify short series of musical events produced by man or machine. For example, the last melody played by the human performer as captured by the segmentation algorithm is referred to as "the last sequence".

Similarity

Similarity refers to a comparative relationship between any two objects of the same class. For instance, melodic similarity is a quantity that specifies the

resemblance between any two melodies, normalised between zero (no relation) to 100 percent (identical melodies). Oscar explores consecutive changes in similarity between melodies produced by man and machine in order to derive tendencies of man and machine coming together or drifting apart.

Short-term memory

Short-term memory (STM) holds the most recent sequence of musical events produced by the human interactor. A sequence of variable length is extracted from a continuous MIDI stream using an adaptive segmentation algorithm. Oscar holds two instances of STM (STM1 and STM2) in a wish to capture the two most recent input sequences. Comparing both STM provides information about the dynamics of human input and feeds particular *sensors*. STM is a subclass of the *melody* object.

Working memory

Working memory (WM) is a computational FIFO (first-in, first-out) data structure holding the last (typically 32) MIDI events produced by the human performer. Therefore, WM reflects the current musical context suggested by that performer. Many sensors address the contents of WM in an attempt to extract specific features. WM is a subclass of the *Melody* object.

References

Abelson, H Sussman, G and Sussman, J 1996, *The Structure and Interpretation of Computer Programs*, The MIT Press, Cambridge.

Assayag, G, Dubov, S and Delerue, O 1999, 'Guessing the composers mind; applying universal prediction to musical style', in *Proceedings of the International Computer Music Conference,* Beijing, pp. 496-499.

Assayag, G, Dubnov, S 2004 'Using Factor Oracles for Machine Improvisation', *Soft Computing*, vol. 8, no. 9, pp. 604-610.

Baggi, D 1991, 'Neurswing: An Intelligent Workbench for the Investigation of Swing in Jazz', *IEEE Computer*, vol. 24, no. 7, pp. 60-64.

Bailey, D 1980, *Improvisation: its nature and practice*, Moorland Publishing.

Barlow, C 1986, *Journey to Parametron*, Feedback Editions, Cologne.

Baron, R and Byrne, D 1997, *Social Psychology*, Allyn and Bacon, Needham Heights.

Beer, R 1990, *Intelligence as adaptive behaviour*, Academic Press, Cambridge.

Behrman, D 1981, *On The Other Ocean/Figure In a Clearing*, Lovely Records, New York.

Beishon, J and Peters, G (eds) 1976, *Systems Behaviour,* 2nd edition, The Open University Press, Harper & Row Publishers, London.

Belet, B 1994, 'Integrating real-time interactive software synthesis, pre-processed synthesis, MIDI data and acoustic piano in composition and live performance', in *Society for electro-Acoustic Music in the United States*, Middlebury.

Berger, J 2004, 'Who cares if it listens? An essay on creativity, expectations and computational modelling of listening to music' in *Virtual Music: Computer Synthesis of Musical Style*, D Cope (ed.), The MIT Press, Cambridge, pp. 263-281.

Berry, W 1989, *Structural Functions in Music*, Dover Publications, New York.

Beyls, P 1988, 'Musical Morphologies from Self-Organizing Systems', *Interface, Journal of New Music Research*, vol. 19, no. 2-3, pp. 205-218.

Beyls, P 1991, 'Chaos and Creativity. The Dynamic Systems approach to Musical Composition', *Leonardo Music Journal*, vol. 1, no. 1, pp. 31-36.

Beyls, P 1991, 'Self-Organising Control Structures using Multiple Cellular Automata', *Proceedings of the International Computer Music Conference*, Montreal, pp. 254-257.

Beyls, P 1997, 'A Survey of Agents Based Real-time Interactive Systems', *Proceedings of the International Computer Music Conference,* Thessaloniki, pp. 379-382.

Beyls, P 1997, 'Aesthetic Navigation: Musical Complexity Engineering Using Genetic Algorithms,' *Proceedings of JIM97*, GRAME, Lyon, pp. 97-105.

Beyls, P 2003, 'Selectionist Musical Automata: Integrating Explicit Instruction and Evolutionary Algorithms', *Brazilian Computer Music Conference*, University of Campinas, Campinas, pp. 67-75.

Beyls, P 2005, 'A molecular collision model of musical interaction', *Proceedings of the Generative Arts Conference*, Milan Polytechnic University, Milan, pp. 375-386.

Beyls, P 2005, 'Evolving Adaptive Sensors in a Synthetic Listener', *Proceedings of the International Computer Music Conference*, Pompeu-Fabra University, Barcelona, pp. 563-566.

Beyls, P 2007, 'Interaction and Self-Organization in a Society of Musical Agents', *Music-AL*, CD-ROM, 9th European Conference on Artificial Life, Lisbon.

Biles, J 1994, 'GenJam: A Genetic Algorithm for Generating Jazz Solos', *Proceedings of the International Computer Music Conference*, Aarhus, pp. 131-137.

Biles, J, Anderson, P and Loggi, L 1996, 'Neural network fitness functions for a musical IGA', *Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA'96) and Soft Computing (SOCO'96)*, Reading, pp. 339-344.

Biles, J 1998, 'Interactive GenJam: Integrating Real-time Performance with a Genetic Algorithm', *Proceedings of the International Computer Music Conference*, Ann Arbor, pp. 232-235.

Biles, J 2001, 'Autonomous Genjam; eliminating the fitness bottleneck by eliminating fitness', *GECCO 2001 Workshop*, San Francisco, viewed May 5, 2006, < http://www.it.rit.edu/~jab/GECCO01/index.html>.

Biles, J 2002, 'GenJam: Evolutionary Computation Gets a Gig', *Proceedings Conference for Information Technology Curriculum*, Rochester Institute of Technology, New York, viewed May 5, 2006, <http://www.it.rit.edu/~jab/GenJam.html>.

Biles, J 2005, *Evolutionary Music Bibliography to accompany the Evolutionary Music Tutorial given at GECCO 2005,* Rensselaer Institute of Technology, viewed August 13, 2007, <http://www.it.rit.edu/~jab/EvoMusic/EvoMusBib.html>.

Biles, J and Miranda, E (eds) 2007, *Evolutionary Computer Music,* Springer.

Bisshof, J, Gold R and Horton J 1978, 'Music for an interactive network of microprocessors', *Computer Music Journal*, vol. 2, no. 3, pp. 24-29.

Blackwell, T 2001, 'Making Music with Swarms', MSc. Thesis, University College London.

Blackwell, T and Branke, J 2001, 'Multi-swarm optimization in dynamic environments', in *Lecture Notes in Computer* Science, *Applications of Evolutionary Computing*, Springer, pp. 489-500.

Blackwell, T and Bentley, P 2002, 'Improvised Music with Swarms', *Proceedings of Congress on Evolutionary Computation*, Honolulu, pp. 1462-1467.

Blackwell, T and Young, M 2004, 'Self-organised music', *Organised Sound,* vol. 9, no. 2, pp. 123-136.

Blackwell, T and Young, M 2007, '*Live Algorithms for Music network – Final Report March 2007*', viewed March 15, 2008, <http://igor.gold.ac.uk/~mas01tb/papers/AISB.pdf >

Bonabeau, E and Theraulaz, G 1995, 'Why Do We Need Artificial Life?' in *Artificial Life: An Overview*, C Langton (ed.), The MIT Press, Cambridge, pp. 303-325.

Borgo, D 2002, 'Synergy and Surrealestate: The Orderly Disorder of Free improvisation', *Pacific Review of Ethnomusicology*, vol.10, Fall 2001/Spring 2002, pp. 1-24.

Borgo, D and Goguen, J 2004, 'Sync or Swarm: Group Dynamics in Musical Free Improvisation', short abstract, *Conference of Interdisciplinary Musicology (CIM04),* Graz, pp. 52-53.

Borgo, D and Goguen, J 2005, 'Rivers of Consciousness: The Nonlinear Dynamics of Free Jazz', L Fisher (ed.), *Jazz Research Proceedings Yearbook,* vol. 25, pp. 46–58

Brooks, R 1991a, 'Intelligence without reason', *Proceedings of 12th International Joint Conference on Artificial Intelligence*, Sydney, pp. 569–595.

Brooks, R. 1991b, 'Intelligence without representation', *Artificial Intelligence*, vol. 47, no. 1-3, pp. 139-160.

Brooks, R and Stein, L 1994, 'Building Brains for Bodies', *Autonomous Robots*, vol. 1, no. 1, pp. 7-25.

Bryson, J. 1995, 'The reactive accompanist: adaptation and behavior decomposition in a music system', in L Steels (ed.), *The Biology and Technology of Intelligent Autonomous Agents*, Springer, pp. 365-376.

Burk, P, Rosenboom D and Polansky L 1987, 'HMSL: overview (version 3.1) and notes on intelligent instrument design', *Proceedings of the International Computer Music Conference*, Urbana – Champaign, pp. 220-227.

Burton, A and Vladimirova, T 1999, 'Generation of Musical Sequences with Genetic Techniques', *Computer Music Journal,* vol. 23, no. 4, pp. 59-73.

Cambouropoulos, E 2001, 'The Local Boundary Detection Model (LBDM) and its Application in the Study of Expressive Timing', *Proceedings of the International Computer Music Conference*, Havana, pp. 17-22.

Citron, J 1985, *Cybernetic Music*, Tab Books Inc. Blue Ridge Summit.

Chadabe, J 1975, 'The voltage-controlled Synthesizer', in J Appleton and R Perera (eds), *The Development and Practice of Electronic Music,* Prentice-Hall, Inc., Englewood Cliffs, pp. 138-188.

Chadabe, J 1977, 'Introduction to the PLAY program', *Computer Music Journal*, vol. 2, no. 1, pp. 55-66.

Chadabe, J 1989, 'Interactive composing; an overview', in *The Music Machine*, C Roads (ed.), The MIT Press, Cambridge, pp. 143-148.

Chadabe, J 1997, *Electric Sound*, Prentice Hall, New York.

Chadabe, J 2002, 'The limitations of mapping as a structural descriptive in electronic instruments', *Proceedings of NIME 2002*, Dublin, pp. 1-5.

Chadabe, J 2004, Electronic Music and Life, *Organised Sound*, vol. 9, no. 1, pp. 3-6.

Cope, D 1991, *Computers and Musical Style*, The Computer Music and Digital Audio Series, Volume 6, A-R Editions, Inc., Madison.

Cope, D 1996, *Experiments in Musical Intelligence.* The Computer Music and Digital Audio Series, Volume 12, A-R Editions, Inc., Madison.

Cope, D 2005, *Computer models of Musical Creativity*, The MIT Press, Cambridge.

Dannenberg, R 1989, 'Real-time scheduling and computer accompaniment', in *Current Directions in Computer Music Research*, M Matthews and J Pierce (eds), The MIT Press, Cambridge, pp. 225-261.

Dannenberg, R.,Thom B, and Watson, D 1997, 'A Machine Learning Approach to Musical Style Recognition', *Proceedings of the International Computer Music Conference*, Thessaloniki, pp. 344-347.

Dawkins, R 1986, *The Blind Watchmaker*, Longman Publishing.

Dawkins, R 1996, *Climbing Mount Improbable*, WW Norton and Co.

De Bono, E 1990, *Lateral Thinking*, Penguin Books

Dennett, D 2003 *Freedom Evolves*, Viking Penguin Putnam Inc. New York

Dewdney, AK 1987, 'Computer Recreations: Diverse personalities search for social equilibrium at a computer party', *Scientific American*, September 1987, p. 104.

Di Scipio, A 1994, 'Formal Processes in Timbre Composition, Challenging the Dualistic Paradigm of Computer Music', *Proceedings of the International Computer Music Conference*, Aarhus, pp. 202-208.

Di Scipio, A 2003, 'Sound is the Interface, Sketches of a Constructive Ecosystemic View of Interactive Signal Processing', *Proceedings of the Colloquium on Musical Informatics*, Firenze.

Di Scipio, A 2005, 'Due di Uno. A composition dedicated to Horatio Vaggione', *Contemporary Music Review*, vol. 24, no. 4+5, pp. 383-398.

Dorin, A 2005, 'Artificial Life, Death and Epidemics in Evolutionary, Generative Electronic Art, Proceedings of 3rd European Workshop on Evolutionary Art, Applications of Evolutionary Computing, Lausanne, pp. 448-457.

Dorin, A 2006, 'The Sonic Artificial Ecosystem', *Proceedings of the Australian Computer Music Conference*, Adelaide, pp. 32-37

Dubnov, S, Assayag, G. Lartillot, O and Bejerano, G 2003, 'Using Machine-Learning Methods for Musical Style Modelling', *IEEE Computer,* vol. 36, no. 10, pp. 73-80.

Eysenck, H 1973, *The Inequality of Man*, Temple Smith, London.

Fahlman, S 1990, 'The recurrent cascade-correlation architecture', *Proceedings of the 1990 conference on advances in neural information processing systems*, Denver, pp. 190-196.

Fateman, R 1988, *Common LISP, The Reference*, Addison-Wesley Publishing Company, Inc., Reading.

Fiume, E and Van de Panne, M 1993, 'Sensor-Actuator Networks', *Proceedings of SIGGRAPH 93*, ACM Computer Graphics, pp. 335-342.

Fober, D, Letz, S and Orlarey, Y 2004, 'MidiShare, une architecture logicielle pour la musique', in *Informatique Musicale*, Hermes, pp. 175-194.

Franco, S 1974, *Hardware design of a real-time musical system*, PhD Thesis, University of Illinois at Champaign-Urbana.

Franklin, S and Graesser, A 1996, 'Is it an agent or just a program? A taxonomy for autonomous agents', *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*, Springer.

Friedman, G 1959, 'Digital simulation of an evolutionary process', in *General Systems, Yearbook of the Society for General Systems Research*, vol. 4, L Von Bertalanffy and A Rapoport (eds), Society of General Systems Research, Stanford, pp. 171-184.

Friberg, A, R. Bresin, L Fryden and Sunberg, J 1998, 'Musical Punctuation on the Microlevel: Automatic Identification and Performance of Small Melodic Units', *Journal of New Music Research*, vol. 27, no. 3, pp. 271-292.

Fry, C 1980, 'Computer Improvisation', *Computer Music Journal*, vol. 4, no. 3, pp. 48-58.

Gold, R 2007, *The Plenitude. Creativity, Innovation and Making Stuff*, The MIT Press, Cambridge.

Goldberg, D 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.

Gould, SJ 1996, *Full house*, Random House, New York.

Harth, E 1995, *The Creative Loop. How the Brain makes a Mind,* Penguin Books.

Hiller, L. and Bean, C. 1966, 'Information theory analysis of four sonata expositions', *Journal of Music Theory*, vol. 10, pp. 96-137.

Hofstadter, D 1995, *Fluid Concepts and creative analogies, Computer Models of the Fundamental Mechanisms of Thought*, Harper Collins.

Holland, J 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Holland, J, Booker L and Goldberg, D 1989, 'Classifier systems and genetic algorithms', *Artificial Intelligence,* vol. 40, no. 1-3, pp. 235-282.

Holland, J 1995, *Hidden Order, How Adaptation Builds Complexity*, Helix Books, Reading.

Holland, J 1998, *Emergence*, Helix Books, Reading.

Horner, A and Goldberg, D 1991, *Genetic Algorithms and Computer-Assisted Music Composition*, Technical Report CCSR-91-20, Centre for Complex Systems Research, University of Illinois, Urbana-Champaign.

Hunt, A 2002, 'The importance of parameter mapping in electronic instrument design', *Proceedings of NIME 2002*, Dublin, pp. 149-154.

Huron, D 2006 *Sweet Anticipation, Music and the Psychology of Expectation*, The MIT Press, Cambridge.

Impett, J 2002, 'Interaction, Simulation and Invention: a Model for Interactive Music', in *Proceedings of the ALLMA 2002 Workshop on Artificial Models for Musical Applications*, Editoriale Bios, Cosenza, pp. 108-119.

Jacob, B 1995, 'Composing with genetic algorithms', *Proceedings of the International Computer Music Conference*, Banff, pp. 452-455.

Kaelbling LP, Littman, ML and Moore, A 'A Reinforcement Learning; A Survey', *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285.

Kauffman, S 1995, *At Home in The Universe; Self-organization and Selection in evolution*, Oxford University Press Inc, New York.

Kayser, W 2000, *De Mythe*, Wim Kayser interviews Stephen Jay Gould on VPRO Television, Holland, March 26, 2000.

Koza, J 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* The MIT Press, Cambridge.

Kugler, P (ed.) 1989, *Self-organization in Biological Work Spaces*, Elsevier Science Publishers BV.

Langton, C (ed.) 1997, *Artificial Life: an overview*, The MIT Press, Cambridge.

Laske, O 1996, 'Knowledge Technology and the Arts: A Personal View', in *Computers and Mathematics with Applications*, vol. 32, no. 1, pp. 85-88.

Lehrdall, F and Jackendoff, R 1983, *A Generative Theory of Tonal Music*, The MIT Press, Cambridge

Lewis, G 1999, 'Interacting with latter-day musical automata', *Contemporary Music Review*, vol. 10, no. 3, pp. 99-112.

Lewis, G 1992, *Voyager*, CD Disk Union Avan-CD 014.

Lewis, G 1995, 'Singing the Alternative Interactivity Blues', *Western Front*, vol. 7, no. 2, pp. 18-22.

Lewis, G 2000, 'Too many notes: computers, complexity and culture in Voyager', *Leonardo Music Journal*, vol. 10, pp. 33-39.

Lewis, G 2004, 'Gittin' To Know Y'all: Improvised Music, Interculturalism and the Radical Imagination', *Critical Studies in Improvisation*, vol. 1, no. 1, viewed June 2, 2006, <http://gir.uoguelph.ca/index.php/csieci/article/view/6/14>.

Loy, G 1985, 'Musicians make a standard: the MIDI phenomenon', *Computer Music Journal,* vol. 9, no. 4, pp. 8-26.

Machover, T and Chung, J 1989, 'Hyperinstruments; musically intelligent and interactive performance and creativity systems', *Proceedings of the International Computer Music Conference*, Columbus, pp. 186-190.

Machover, T 1992a, 'Hyperinstruments: A Composer's Approach to the Evolution of Intelligent Musical Instruments', in *Cyberarts*, William Freeman (ed.), pp. 67-76.

Machover, T 1992b, *"Classic" hyperinstruments - 1986-1992 - A composer's approach to the evolution of intelligent musical instruments*, viewed August 1, 2006, <http://brainop.media.mit.edu/Archive/Hyperinstruments/classichyper.html>.

Maes, P 1994, 'Modelling Adaptive Autonomous Agents', *Journal of Artificial Life*, vol. 1, no. 1-2, pp. 746-753.

Mandelis, J 2002, 'Adaptive Hyperinstruments: Applying Evolutionary Techniques to Sound Synthesis and Performance', *Proceedings of the NIME 2002 Conference*, Dublin, pp. 192-193.

Maturana, H and Varela, F 1984, *The Tree of Knowledge, The Biological Roots of Human Understanding*, Shambhala Publications Inc., Boston

Manzara, L, Witten, I and James, M 1992, 'On the Entropy of Music: An Experiment with Bach Chorale Melodies', *Leonardo Music Journal*, vol. 2, no. 1, pp. 81-88.

McCormack, J 2001, 'Eden: an Evolutionary Sonic Ecosystem', in *Advances in Artifical Life*, Kelemen, J and Sosik, P (eds), Springer, pp. 133-142.

McCormack, J and Dorin, A 2001, 'Art, Emergence and The Computational Sublime', in Dorin (ed.) *Second Iteration: a Conference on generative Systems in the Electronic Arts*, CEMA, Melbourne, pp. 67-81.

McCormack, J and Dorin, A 2002, 'Ways of Seeing: Visualization of Artificial Life Environments', *Proceedings of Beyond Fitness: Visualising Evolution,* pp. 155-162.

McCormack, J 2003, 'Evolving Sonic Ecosystems', in *The International Journal of Systems and Cybernetics – Kybernetes*, vol. 32, no. 1-2, Emerald, Northampton.

McFarland, D 1992, 'Autonomy and self-sufficiency in robots', AI-Memo 92-03, AI-Lab VUB, Brussels.

Meyer, L 1956, *Emotion and meaning in music*, The University of Chicago Press, Chicago.

Minsky, M 1981, 'Music, Mind, and Meaning' in *Music, Mind, and Brain: The Neuropsychology of Music,* Clynes, M (ed.), Plenum, New York.

Minsky, M 1985, *The Society of Mind*, Simon and Schuster, New York.

Miranda, E (ed.) 2000, *Readings in Music and Artificial Intelligence*, Harwood Academic Publishers, Amsterdam.

Miranda, E 2002, 'Emergent Sound Repertoires in Virtual Societies', *Computer Music Journal,* vol. 26, no. 2, pp. 77-90.

Miranda, E (ed.) 2003, 'Evolutionary Music; At the Crossroads of Evolutionary Computing and Musicology', *Contemporary Music Review*, vol. 22, no. 3, pp. 1-3.

Miranda, E 2003, 'On the Music of Emergent behaviour: What Can Evolutionary Computation Bring to the Musician? *Leonardo*, vol. 36, no. 1, pp. 55-59.

Miranda, E 2003, 'On Making Music with Artificial Life Models', *Consciousness Reframed 2003,* UWCN, Wales, pp. 299-304.

Miranda, E and Biles, J (eds) 2007, *Evolutionary Computer Music,* Springer.

Miranda, E 2008, Emergent songs by social robots, *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 20, no. 4, pp. 1-16.

Mitchell, M 1998, *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge.

Moore, R 1988, The dysfunctions of MIDI, *Computer Music Journal*, vol. 12, no. 1, pp. 19-28.

Morris, R1999 *Artificial Worlds, Computers, complexity and the riddle of life*, Plenum, New York.

Mumma, G 1975, 'Live-Electronic Music', in *The Development and Practice of Electronic Music*, J Appleton and R Perera (eds), Prentice-Hall, Inc., Englewood Cliffs, pp. 286-335.

Narmour, E 1990, *The Analysis and Cognition of Basic Melodic Structures*, University of Chicago Press, Chicago.

Nemirovsky, P and Watson, R 2003, 'Genetic Improvisation Model', EvoMUSART Conference, Springer Lecture Notes in Computer Science, pp. 547-558.

Newell, A and Simon, H 1972, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs.

Pachet, F 2003, 'The Continuator: Musical Interaction With Style', *Journal of New Music Research,* vol. 31, no. 1, pp. 27-37.

Pachet, F 2004a, 'Beyond The Cybernetic Jam: the Continuator', *Computer Graphics and Applications*, vol. 24, no. 1, pp. 31-35.

Pachet, F 2004b, 'Enhancing Individual Creativity with Interactive Musical Reflective Systems' in *Musical Creativity, Current Research in Theory and Practice*, I Deliege and G Wiggins (eds), Psychology Press, pp. 359-375.

Pachet, F 2004c, 'Playing with Virtual Musicians: The Continuator in practice', *IEEE Multimedia*, vol. 9, no. 3, pp. 77-82.

Pachet, F 2004d, 'On the Design of a Musical Flow Machine', in *The Future of Learning, Learning Zone of One's Own*, M Tokoro and L Steels (eds), IOS Press, pp. 113-134.

Packard, N 1988, 'Intrinsic Adaptation in a Simple Model of Evolution', in *Artificial Life*, C Langton (ed.), Addison-Wesley, Reading, pp. 141-155.

Pennycook, B, Stammen, D and Reynolds, D 1993, 'Towards a Computer Model of a Jazz Improviser', *Proceedings of the International Computer Music Conference*, Tokyo, pp. 228-231.

Perkis, T n.d. 'Complexity and Emergence in the American Experimental Music Tradition', viewed June 25, 2006, <http://www.perkis.com/wpc/abisko.pdf>.

Perkis, T 1989, The Hub, Computer Network Music, CD ART 1002 Artifact Recordings, Berkeley.

Pfeifer, R and Scheier, C 1999, *Understanding Intelligence*, The MIT Press, Cambridge.

Phon-Amnuaisuk, S, Tuson, A and Wiggins, G 1999, 'Evolving Musical Harmonization', *International Conference on Artificial Neural Networks and Genetic Algorithms*, Portoroz, pp. 220-224.

Prigogine, I and Stengers, I 1984, *Order Out of Chaos*, Bantam Books Inc.

Pritchett, J 1993, *The Music of John Cage*, Cambridge University Press.

Prusinkiewicz, P and Lindenmayer, A 1990, *The Algorithmic Beauty of Plants*, Springer.

Ray, T 1991, 'An approach to the synthesis of life', in C Langton, C Taylor, JD Farmer and S Rasmussen (eds.), *Artificial Life II*, Addison-Wesley, Reading, pp. 371-408.

Reynolds, G 1987, 'Flocks, herds and schools: a distributed behavioural model', *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM, New York, pp. 25-34.

Riegler, A 2002, 'When is a system embodied?', *Cognitive Systems Research*, no. 3, pp. 339-348.

Rowe, R 1992, 'Interactor 4.0.8', in *Leonardo Music Journal*, vol. 2, no. 1, pp. 122-123.

Rowe, R 1993, *Interactive Music Systems*, The MIT Press, Cambridge.

Rowe, R 2001, *Machine Musicianship*, The MIT Press, Cambridge.

Rumelhart, D and McClelland, J 1986, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations, The MIT Press, Cambridge.

Russell, S and Norvig, P 2003, *Artificial Intelligence, A Modern Approach*, Prentice Hall, Upper Saddle River.

Schillinger, J 1948, *The Mathematical Basis Of The Arts*, Philosophical Library, New York.

Sims, K 1991, 'Artificial Evolution for Computer Graphics', *Proceedings of SIGGRAPH 91*, ACM Computer Graphics, vol. 25, no. 4, pp. 319-328.

Sims, K 1994, 'Evolving 3D morphology and behavior by competition', in *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R Brooks and P Maes (eds.), MIT Press/Bradford Books, pp. 28-39.

Singer, D and Revenson, T 1996, *Piaget Primer*, Penguin Publishing Group.

Smithers, T 1991, 'Taking Eliminative Materialism Seriously: A Methodology for Autonomous Systems Research', in *Towards a Practice of Autonomous Systems*, F Varela and P Bourgine (eds), Paris, pp 31-40.

Steels, L 1994, 'Emergent functionality in robotic agents through on-line evolution', *Proceedings of A-life IV*, pp. 8-16.

Steels, L 1995, 'A selectionist approach to behavioural development', AI Memo 95-06 VUB, Brussels.

Sutton, R and Barto, A 1998, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge

Taube, H 2004, *Notes From The Metalevel. Introduction to Algorithmic Music Composition*, Taylor and Francis, London.

Tenney, J and Polansky, L 1980, 'Temporal Gestalt Perception in Music', *Journal of Music Theory*, vol. 24, pp. 205-241.

Thomson, J and Stuart, H 1986, *Nonlinear Dynamics and Chaos*, John Wiley and Sons, Chichester.

Todd, S and Latham, W 1992, *Evolutionary Art and Computers*, Academic Press, New York.

Todd, P and Werner, G 1999, 'Frankensteinian Methods for Evolutionary Music Composition', in Griffith and Todd (eds) *Musical Networks: Parallel distributed perception and performance,* MIT Press/Bradford Books.

Tokui, N and Iba, H 2000, 'Music Composition with Interactive Evolutionary Computation', *Proceedings of the International Conference on Generative Art*, Milan, viewed January 20, 2005, <http://www.generativeart.com>.

Unemi, T 2002, 'SBEAT3: A Tool for Multi-part Music Composition by Simulated Breeding', *Proceedings of the 8th International Conference on Artificial Life*, Sydney, pp. 410-413.

Unemi, T and Bisig, D 2004, 'Playing Music by Conducting BOID Agents - A Style of Interaction in the Life with A-Life', in J Pollack et al. (eds), *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, The MIT Press, pp. 546-550.

Ventrella, J 1993, 'Attractions and Repulsions, A Gravity-based Animated Microworld for Exploration of Dynamical Systems', viewed August 16, 2007, <http://www.ventrella.com/Ideas/attractions.html>.

Von Bertalanffy, L 1973, *General System Theory, Foundations Development Applications*, Penguin Books.

Vos, P and Van Geenen, E 1996, 'A Parallel-Processing Key-Finding Model', *Music Perception*, vol. 14, no. 2, pp. 185-224.

Waisvisz, M 1985, 'THE HANDS, a set of remote MIDI-controllers', *Proceedings of the International Computer Music Conference*, Burnaby, pp. 313-318.

Waldrop, M 1994, *Complexity, The Emerging Science at the Edge of Order and Chaos*, Penguin Science Books, London.

Walker, H, Martirano, S, and Scaletti, C 1992, 'ImprovisationBuilder: improvisation as conversation', *Proceedings of the International Computer Music Conference*, San Jose, pp. 190-193.

Wolfram, S 1994, *Cellular Automata and Complexity, Collected Papers*, Addison-Wesley.

Worrall, D 2004, 'Complex Systems in composition and improvisation', *Organised Sound*, vol. 9, no. 2, pp. 121-122.