5. Problems of stimulus equivalence [1]

5. Introduction

A fundamental difficulty when using neural networks applied to pattern recognition, is the problem of stimulus equivalence - the invariance of symbolic information independent of transformation within a search space. For example, symbolically the letter A remains an A irrespective of its position, size or orientation within an image field. Adult humans can generally recognise patterns accurately, despite transformations and distortions. Classically it has been hypothosised that this ability is due to a normalisation process that occurs before the classification process starts.

One criticism of the anarchic paradigm has been that there is no clear mechanism by which such normalisation can occur since at first sight unstructured systems such as nets do not provide any mechanism for attention - the focussing of analysis onto a subset of a larger pattern.

This chapter will describe three anarchic mechanisms for solving problems of invariance, Hinton Mapping, Fukushima's Neocognitron and Stochastic Searching, together with a brief discussion of two other methods - Pattern Correlation and Fourier analysis. These are introduced briefly below and in more detail in subsequent sections.

The simplest solution to problems of stimulus equivalence is to use model correlation (5.1). Many traditional search methods, looking for the best fit of a specific data model within a given search space, involve this form of sequential comparison of elements from the model with elements from the search space, at each possible mapping of the model into the search space.

A parallel implementation of correlation, found in the Neocognitron (Fukushima, 1988), is knowledge replication (5.2). Instead of comparing the model at each position in the search space, pattern information is stored in a set of functional units mapped over some or all of the area.

An entirely different method, which works well for problems of translational stimulus equivalence, is the analysis of the Fourier Transform of the model. The amplitude spectrum forms a pattern independent of model position within the search space (5.3).

Another solution is Hinton Mapping (Hinton, 1981). This involves using a set of functional units with the ability to select and apply different mappings of the model into the search space (5.4). The ability to consider a set of mappings into the search space forms the basis of stochastic searching (5.5), a technique developed by the author.

1 From J.M.Bishop, (1989), *Anarchic Techniques for Pattern Classification*, PhD Thesis, Chapter 5, University of Reading, UK.

Stochastic searching is of lower computational order than traditional sequential search methods, such as correlation. It involves the selection of a set of potential mappings into the search space and comparison of data in the search space, pointed to by these mappings, with data from the model. The selection of potential mappings is progressively biased towards those which produce positive comparisons between elements of the model and the search space. These elemental comparisons can either be simple tests for equivalence, if one specific pattern is being searched for, or n-tuple RAM neurons can be used if a fuzzy match is required.

A thorough description of stochastic searching is given, using experimental data to highlight its advantages and disadvantages over traditional search methods. A brief description is also given of ongoing research projects in this area.

5.1 Model Correlation

The simplest method to determine the location of a model within a given search space is to cross correlate the two patterns (Usher, 1984). The cross correlation of two patterns measures the similarity between them. If the patterns are signals defined by the functions f_a (t) & f_b (t) then their cross correlation function f_{ab} is:

$$f_{ab} = \int_{+\infty}^{-\infty} f_a(t)\, f_b(t + r) = f_a * f_b$$

The representation f (t + r) denotes f (t) time shifted by the value (r), where f (t + r) is the function f (t) advanced by (r) and f (t - r) is the function delayed by (r). The mechanism of correlation is that a function f_a (t) is multiplied by a shifted version of a second function f_b (t) and the result integrated over all time. This gives a specific value for f (r) corresponding to the shift used, where f (r) defines the set of values for all possible shifts. Order matters in cross correlation; f_a (t) * f_b (t) implies that f_b (t) is to be shifted.

If the model is correlated with search space over all possible positions of the model in there search space, then the highest peak in the correlated response $f(r)$, gives the most likely location of the model.

Computationally this method is simple to program, but computationally intensive. A simple implementation of correlation matchine on a best fit search problem is:

```
FOR i := min_search TO max_search DO BEGIN                    {loop a}
  cnt := 0;

  FOR j := min_model TO max_model DO                          {loop b}
    IF search_space [i+j] = model [j] THEN INC (cnt);

  IF cnt > best_cnt THEN BEGIN
    best_cnt := cnt;
    best_pos := j;
  END;
END;
```

It can be seen that **loop b** is executed once every time **loop a** is executed, hence the number of iteration of **loop b** is:

```
= (max_model - min_model)  steps
= SIZE (model)  steps
= (a)  steps.
```

The number of times **loop a** is executed is;

```
= (max_search - min_search)  steps
= SIZE (search) steps
= (b) steps.
```

Hence the order of the algorithm is $O(a \times b)$. If the ratio of (a) to (b) is kept constant then:

$$\frac{a}{b} = k$$

$$a = k \times b$$

Hence, keeping this ratio fixed implies that the Order of the algorithm is:

$$= O(k \times b \times b)$$

$$= O(b^2)$$

In a practical implementation of the technique, k can be reduced but the underlying algorithm bounding order remains constant. One such reduction is to stop **loop b** when a count of errors exceeds the current minimum.

```
FOR i := min_search TO max_search DO BEGIN                          {loop a}
  errors := 0;

  FOR j := min_model TO max_model DO BEGIN                          {loop b}
    IF search_space [i+j] <> model [j] THEN INC (errors);
    IF (errors > least_errors) THEN GOTO 1;
  END;

  least_errors := errors;
  best_pos := j;

1: END;
```

It can be shown on average that the abort from **loop b** occurs around $(\frac{a}{2})$ steps, and substituting $(\frac{a}{2})$ for (a) still results in the algorithm being bounded, $O(b^2)$.

5.2 Knowledge Replication

Knowledge replication is analogous to a parallel form of Model Correlation. Given the problem, to locate a data model within a search space of size [M], using correlation one stored representation of the model is scanned sequentially over the search space. However, using Full Knowledge Replication, [M] models are stored entirely mapping the search space. These representations can be compared with data from the search space in parallel, hence the technique offers high performance. The comparison process can be carried out using either simple template matching or a neural net.

The disadvantage of the above technique is its large memory requirement. Full Knowledge Replication, although offering high performance, is very memory intensive. If the search space is a 512*512 image and the model a sub image of 256*256 pixels, then using template match comparison, 1.72 G bits need to be stored; using an n-tuple net with tuple size of eight for the comparison, 550 G bits are required! These memory requirements are only for a one class discriminator system, and hence for the forseeable future the technique is not practical.

Although full Knowledge Replication is not practical, reduced replication is being used in some systems such as Fukushima's Neocognitron (1988). The Neocognitron is a hierarchical neural network for visual pattern recognition, consisting of multiple layers of cells (figure 5.1).

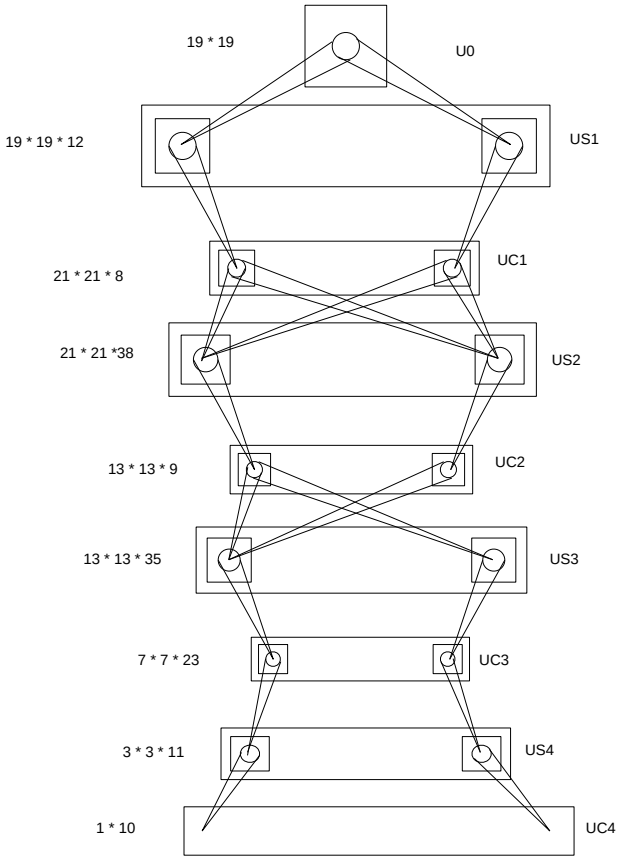


Figure 5.1. Hierarchical structure of Neocognitron, (from Fukushima, 1988)

In each layer cells are bound together into local cell clusters called *cell planes*. Each cell plane extracts one specific feature by the use of knowledge replication.

The Neocognitron acquires the ability to classify patterns by learning, and can be trained on any pattern. After learning, pattern recognition is based on similarity in shape between patterns and is not affected by changes in size, position or deformation.

The hierarchical architecture of the Neocognitron extracts local features at the lowest levels and combines them into more global features at the higher levels. Each cell plane is representative of the object set it is required to classify. At low levels there is a relatively large degree of knowledge replication, as there are many cells within each cell plane responding to the same feature, whereas at the final level there is no replication since there is only one cell in each cell plane.

The strucure of the Neocognitron is that there is an initial layer of *input cells*, followed by alternate layers of *S-cells* and *C-cells*. S-cells are the feature extracting cells and respond strongly to correctly aligned local features within their receptive field (figure 5.2).

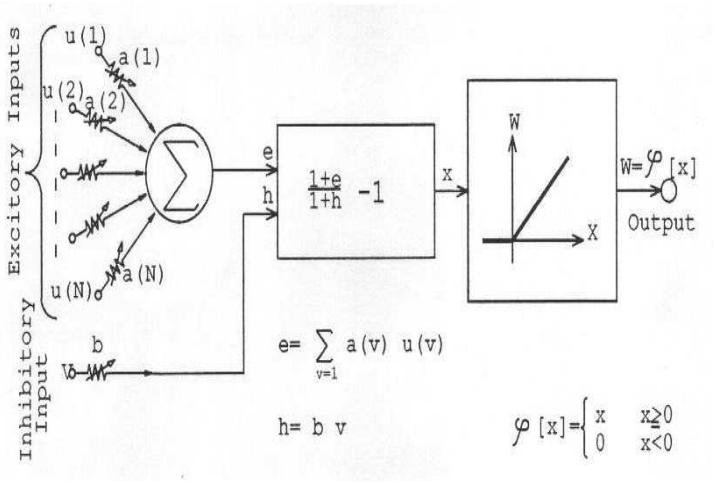
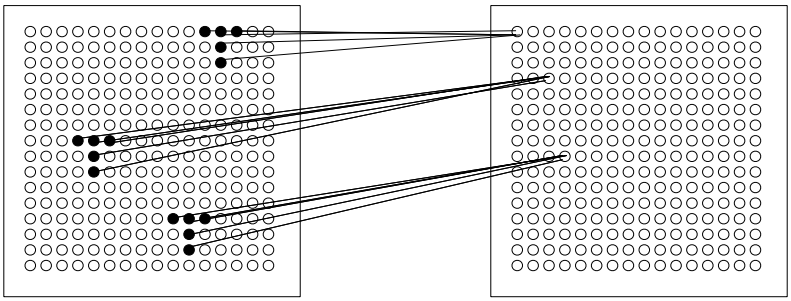


Figure 5.2. Structure of S-Cell, (from Fukushima, 1988)

The connections (weights) to the S-cells are variable and are reinforced with training. The C-cells are used to provide a degree of stimulus equivalence and connections between S-cells and C-cells are fixed and unchangeable. Each C-cell receives input from a group of S-cells *which extract the same feature*, but in slightly different positions (figure 5.3). As information filters from the lower layers to the higher, the density of cells decreases, since the cells at the higher stage usually receive input over a larger receptive field.



Cell plane Knowledge Replicarion in Neocognitron

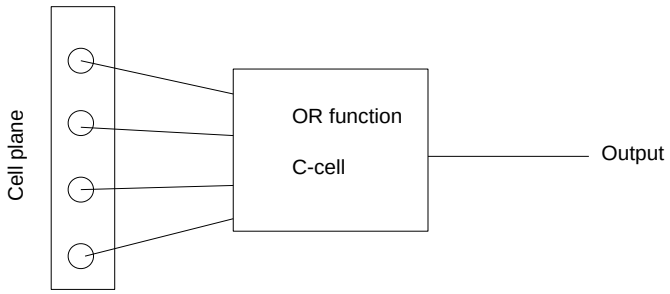


Figure 5.3. Fukushima' s C-cell

The operation of tolerating a relatively small translational error at each stage, rather than all at once, enables the network to recognise distorted input patterns.

5.3 Fourier Transforms

A different way to achieve a degree of stimulus equivalence is to use Fourier Transforms. These convert a finite function of time f (t) into its frequency components, known as the spectrum F (f) of the function.

$$\textit{The forward transform } F(f) = \int_{-\infty}^{+\infty} f(t)\, e^{-j\omega t} dt$$

$$\textit{The reverse transform } F(t) = \int_{-\infty}^{+\infty} F(f)\, e^{-j\omega t} df$$

The reverse transform illustrates that a function can be represented as a sum of cisoids $e^{j\omega t}$. In a real function the positive and negative sides of the integral combine to give terms leading to cos wt, hence the integral implies that any function can be expressed as a sum of cosine waves with a weighting factor F(f) determining the relevant amplitude and phase of the constituent cos functions. For a 2D object, the Fourier integrals are two dimensional.

Fourier analysis is useful in problems of translational stimulus equivalence, (Usher, 1984), because the amplitude of the Fourier Transform of a bounded 2D signal remains constant, independent of position within the signal bounds, (see figure 5.4).

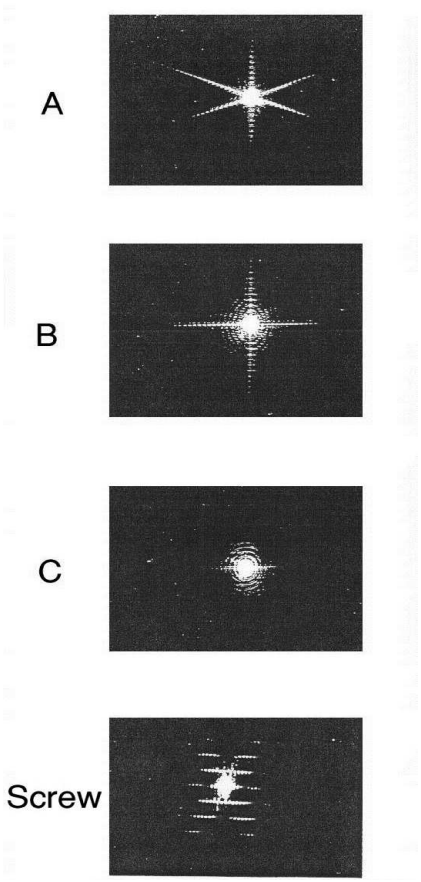


Figure 5.4. Optical Fourier Transform of three ASCII characters {A, B, C} and the profile of a screw

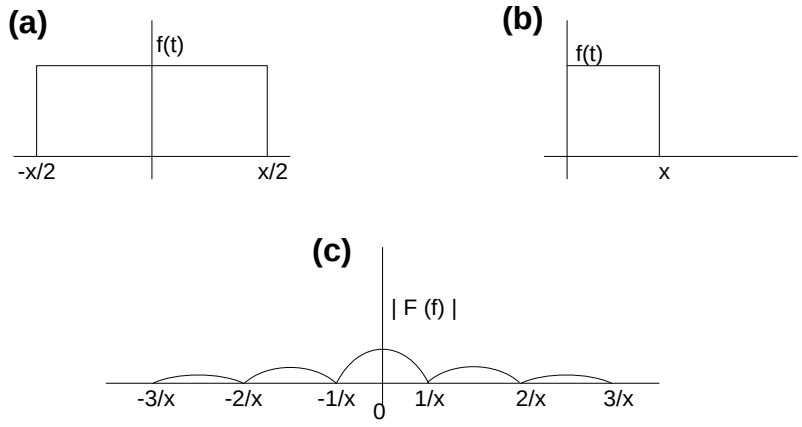


Figure 5.5. Spectrum of a Rectanglular pulse

Consider the [x] width pulse at $(-\frac{x}{2})$ (see figure 5.5a):

$$\begin{aligned} F(f) &= \int_{-\infty}^{+\infty} f(t)\, e^{-j\omega t} dt \\ &= \int_{-\frac{x}{2}}^{+\frac{x}{2}} A e^{-j\omega t} dt \\ &= \left[A \times \frac{-1}{j\omega} \times e^{-j\omega t} \right]_{-\frac{x}{2}}^{+\frac{x}{2}} \\ &= 2 \times \frac{A}{w} \times sin(\frac{\omega x}{2}) \\ &= Q \end{aligned}$$

Consider the [x] width pulse at (0), (see figure 5.5b):

$$\begin{aligned} F(f) &= \int_{-\infty}^{+\infty} f(t)\, e^{-j\omega t} dt \\ &= \int_{0}^{+x} A e^{-j\omega t} dt \\ &= \left[A \times \frac{-1}{j\omega} \times e^{-j\omega t} \right]_{0}^{x} \\ &= 2 \times \frac{A}{w} \times sin(\frac{\omega x}{2}) \times e^{-\frac{j\omega x}{2}} \\ &= Q \times e^{-\frac{j\omega x}{2}} \end{aligned}$$

This relation states that a delay (or advance) of $(\frac{x}{2})$ simply adds a phase term $e^{-\frac{j\omega x}{2}}$ to the spectrum. However, it can be shown that a 2D rotation of the signal also causes a similar rotation of the amplitude spectrum.

Computationally Fourier Transforms are relatively demanding - a one dimensional discrete FFT is of computational order $O(n \times log\, n)$. However, the transforms can be carried out virtually instantaneously by using coherent laser light.

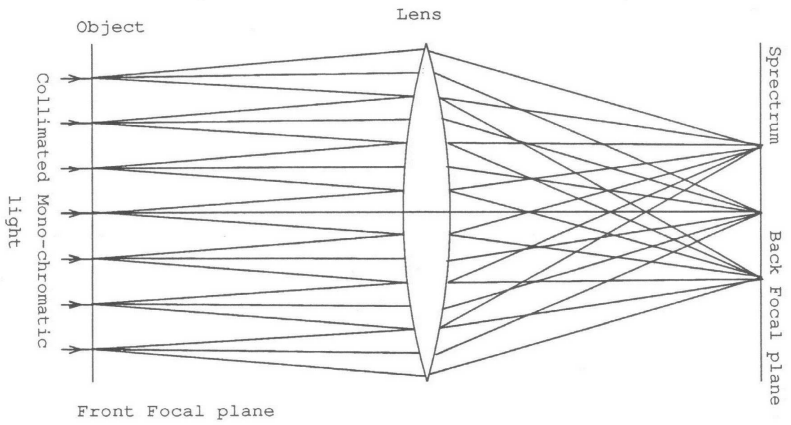


Figure 5.6 Optical Fourier Transform

Given the system in figure (5.6) it can be shown that the light amplitude distribution in the back focal plane is the Fourier Transform of the light amplitude distribution of the object, when the object is placed in the front focal plane. This method of using Fourier Transforms has a number of advantages over conventional mechanisms:

- It is effectively time-independent, so object and spectrum can be observed together.
- Positive and negative frequency values always appear since the spectrum is symmetrical.
- Imaginary objects can be produced by making them have non-zero phase.
- The spectrum is accessible and can be physically operated on, allowing the analogue of electrical filtering to be carried out.

This technique has been used for character recognition by scanning the spectrum with a rotating slit. The light transmitted by the slit is transduced by a photodetector, the output of which is a repetitively varying waveform which can be analysed into harmonics of the rotational frequency. The relative amplitudes of the harmonics characterise each letter in any position. The technique was also used at Aldermaston in a study into automatic fingerprint recognition.

5.4 Hinton Mapping

Hinton mapping is a technique for solving the stimulus equivalence problem that was described by Hinton in 1981. His system consists of two sets of feature detectors, *canonical* and *retinocentric*. Mutually excitory connections link the canonical detectors with higher order detectors such as letter detectors, in the manner of the *interactive activation and competition* (iac) model (Rhumelhart & McClelland, 1986a). Hinton describes a method for mapping retinocentric feature patterns into canonical feature patterns. Although in general in three dimensions there are six degrees of freedom, a simplified version using a normalisation process which corrects for a one dimensional rotational transformation is described here.

Central to Hinton Mapping are *fixed mapping cells* which implement possible mappings into the search space. There is one such cell for each possible mapping into the search space. Interactive activation and

competition between cells ensure that the mapping which yields the best fit of the model into the search space will receive the most activation. After a few iterations, unless the model is symmetric, this mapping will become dominant.

Considering the situation in which a letter has been rotated clockwise by ninety degrees. A dynamic mapping system is required that will transmit activation from the activated retinocentric feature detectors to the relevant canonical detector, offset by a counter clockwise rotation of ninety degrees.

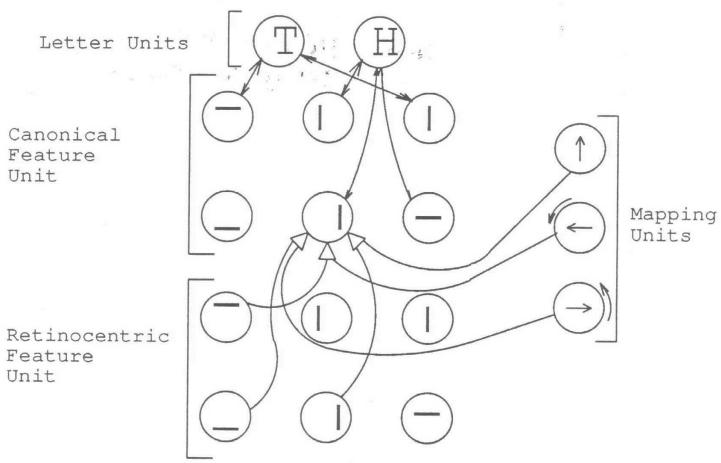


Figure 5.7 Hinton Mapping, (from Rumelhart & McClelland, 1986)

In figure (5.7) three different mapping cells are shown, producing 0^0, $+90^0$ & -90^0 rotational shifts. Activity on mapping cell [x] produces one of two multiplicative inputs to a canonical cell, the other being from the corresponding retinocentric cell (RCU) offset by [x] degrees. Thus for a canonical cell to receive input activation, a retinocentric cell must be active and a mapping cell corresponding to the rotation of the RCU feature to the canonical feature. If just one of the cells is one, either a mapping or RCU, then no activation will pass to the canonical cell. Thus the mapping cell effectively selects dynamic mappings from input space to normalised space.

If the correct rotational mapping is known in advance then, using this technique, it is possible to activate the corresponding mapping cell to map from retinal to normalised coordinate space. Object recognition thus becomes:

- Select a mapping, perhaps on the basis of prior knowledge.
- Activate corresponding mapping cell.
- Analyse degree of match between retinocentric stimuli and higher order features.

This process is repeated until the best match is found. The above technique is simply a sequential implementation of the standard normalisation and compare procedure and does not offer any improvement in performance over traditional methods.

The advantage of the above technique is that by using Hinton mapping cells it is possible to perform the search for the correct mapping in parallel. This is accomplished by using a further set of multiplicative connections to choose the correct mapping for a pattern given the retinal input and correct normalised pattern of activation.

Simultaneous activation of a central feature and a retinal feature is evidence that the mapping is correct. Thus, by allowing central and retinal cells that correspond under a specific mapping to project back to a multiplicative connection on the mapping cell, the activation of the mapping cell can be reinforced. Spurious conjunctions will occur, but correct mapping cells will in general receive more activation than the incorrect ones (unless there is ambiguity due to symmetry). By allowing mapping cells to compete, so that the ones receiving the most activation win, the network will eventually converge onto the correct mapping.

Initially when the retinal input is presented, all the mapping cells are partially active. Each retinal feature then stimulates partial activation to the canonical features corresponding to it over all mappings. The correct mapping allows the correct canonical cell to be partially active. Interactive activation between the canonical cells and the higher order cells will then reinforce the correct mapping relative to the background noise. This process alone can be sufficient for correct identification. However, the multiplicative connections between the canonical cells and the RCUs on the mapping cells force further suppression of the noise. This process swiftly converges onto the correct interpretation of the pattern, as well as the correct canonical representation and correct mapping, purely from the retinal input.

The main problem with Hinton Mapping is that it requires one mapping cell for each potential mapping into the Search Space. Searching for a model of size [N] in a Search Space of size [M], requires [M] mapping cells and this value increases polynomially with the number of degrees of freedom. Stochastic Searching evolved from a study of Hinton Mapping, but uses only [N] mapping cells.

5.5 Stochastic Searching; general

Stochastic Searching is a search technique which uses a random diffusion process to find the *best fit* of the data model in the search space. The data model and the search space are defined as comprising of Atomic Data Units (ADUs) between which comparisons can be made. For example, when searching for the best fit of a string of numbers within a larger string, the ADUs may be the digits that comprise the strings.

Stochastic searching uses an array of *dynamic mapping cells* to test possible mappings into the search space. There is one such cell for each ADU of the model. A random diffusion process is used to ensure that mappings which yield successful comparisons between ADUs in the model and ADUs from the search space, are spread across to other cells.

Stochastic searching is inherently parallel in nature, where the algorithm is *probabilistically* bounded by $O(log\, N)$, where N is the ratio of the size of the model to the size of the search space. However, if the algorithm is implemented on a standard sequential computer, then this order is linear - $O(N)$. These orders have been established empirically as it has not been possible to derive a complete, accurate, description of the algorithm behaviour.

It will be shown how factors influencing algorithm performance include:

a: The Time Ratio, (TR)

This is the ratio of model size [N] to search space size [M].

$$TR = \frac{N}{M}$$

b: Comparison Error Probability, (CRP)

This is the probability that an ADU comparison using a specific mapping will give a misleading result. That is, the comparison may be positive but the mapping used may not be optimal, or the comparison may be negative although the mapping being tested is the best.

5.5.1 Algorithm Description

Stochastic searching is an iterated two stage process, involving the testing of mappings and the diffusion of successful mappings across to other cells, the general structure of which is:

```
INITIALISE (mappings);

REPEAT

      TEST (mappings);

      DIFFUSE (mappings);

UNTIL TERMINATION;
```

This standard format of the algorithm is described below (a worked example is given in Appendix one); another variation, Stochastic Search Nets, is given in (5.8).

Mappings is a data structure describing the [N] mapping cells which define potential mappings of the object into the search space. Each mapping cell has a tag field to store the result of the **test** using this mapping.

Initialise sets the mappings to initial values, according to some predefined schema, usually randomly if no extra information is known about the problem.

Test checks each mapping into the search space by making a local comparison between an *X-randomly selected* ADU of the model and the corresponding ADU from the search space. X-Random selection is defined as the process of sampling an element from a set, *without replacement*.

Diffuse spreads mappings marked positive by test across to other cells.

Termination uses some criterion on the mappings to decide whether the algorithm has found the best mapping of the model into the search space.

5.5.1.1 Initialisation

```
INITIALISE (VAR MC : Mapping_Cells);
BEGIN

  FOR i := 1 TO N DO
    MC [i].map := RANDOM(M);

END {INITIALISE};
```

The initialisation of the mapping cells is determined by any a-priori knowledge of the problem that can be usefully applied.

If stochastic searching is to be used in an environment where there is no a-priori knowledge of the search space, then the *new map* function consists of assigning all mapping randomly, as above, (or at regular intervals through the search area). Advance knowledge of the search space enables mappings to be suitably biased. Two common examples of a-priori knowledge are;

1: The Time Ratio of the search is greater than one.

This occurs if the search space contains less mappings than the number of ADUs in the model. eg. classifying a model into one of [k] sets, where [k] is less than the size of the model.

In this situation, it can be guaranteed that the initialisation procedure can seed at least one of the mapping cells with the correct (best fit) mapping.

2: The previous position of the model is known.

This is only of use in scenarios where the model is unlikely to move in large discontinuous jumps. eg. Successive images of a ball or an aircraft in flight.

In this situation the initial mappings can be randomly biased towards the last known position of the model.

5.5.1.2 The Test Phase

```
PROCEDURE TEST (VAR MC : Mapping_Cells);
BEGIN

  Hits := 0;

  FOR i := 1 TO N DO
    WITH MC [i] DO BEGIN
      index := map;

      element := X-Random (N);

      IF (Model [element] = SearchSpace [index + element]) THEN BEGIN
        cFire := TRUE;
        Hits := Hits + 1; END
      ELSE
        cFire := FALSE;
    END;

END {TEST};
```

The test phase of the algorithm checks the mappings pointed to by the mapping cells. The test can be either a *rigid* or a *fuzzy* comparison of an ADU(s) from the model with an ADU(s) from the search space.

In a rigid comparison, the test for each mapping cell comprises of the selection of [j] ADUs from the model, ($1 \leqslant j \leqslant N$), which are compared with [j] ADUs from the search space, pointed to by that mapping cell. The result of the test is either a unanimous or a majority decision of the [j] ADU

comparisons.

One method of implementing a fuzzy comparison is to use an n-tuple RAM(s) to test the pattern pointed to by the given mapping. The fuzzy test may use one 2^j bit RAM to cover the [j] sample points of the search space or use [m] smaller RAMs as described in chapter three (3.1 & 3.2).

In both rigid and fuzzy methods care must be taken to ensure that when testing a specific mapping, each iteration of the algorithm uses a different segment of the model. Thus when testing mapping [m] using segment [i] of the model, [j] ADUs of the model from position [i] are compared with [j] ADUs from the search space at position [m+i], where [i] is chosen using X-Random selection. This ensures that the system does not get caught in a 'local minimum' where the specific mapping gives the wrong result (see section 5.5.b).

X-Random selection is a technique for randomly sampling a search space, where each point of the search space is guaranteed to be sampled once only - sampling without replacement.

The result of the test operation for each mapping cell is recorded for use by the diffusion phase.

5.5.1.3 The Diffusion Phase

```
PROCEDURE DIFFUSE (VAR MC : Mapping_Cells);
BEGIN

  FOR i := 1 TO N DO
    WITH MC [i] DO
      IF (cFire = FALSE) THEN BEGIN
        p := RANDOM (N);

        IF (MC [p].cFire = TRUE) THEN
          map := Sampling Distribution [p].map
        ELSE
          map := RANDOM (M);
      END;

END {DIFFUSE};
```

The diffusion phase ensures that potential successful mappings are spread across the mapping cells. Each mapping cell is redefined as follows;

1 A check is made to see if the result of the previous **test** using the current mapping was successful - if it was that mapping remains assigned to that cell - if it was unsuccessful, then it is changed.
2 To change a mapping, a new mapping is selected from the current list of mapping cells. In the standard form of Stochastic Searching this selection is made at random as there is no communication between mapping cells. Other implementations, such as Stochastic Nets (5.8) use inter cell communication to ensure the spread of potentially successful mappings.
3 If the new mapping, sampled from the list of mapping cells, was also marked unsuccessful then a completely new mapping is made into the search space using the *new map* function (see initialisation).

5.5.1.4 Termination Condition

```
FUNCTION TERMINATION;
BEGIN

  IF (Hits > base_threshold) THEN BEGIN
    stability_time := stability_time + 1;

    Termination := (stability_time = stable);
  END ELSE
    stability_time := 0;
  ENDIF;

END {Termination};
```

The simplest termination condition that can be used is to sum the number of mappings that are the same. The algorithm terminates when this sum [Q] exceeds a given threshold. A problem with this technique is that it requires a threshold to be set before use. If the model is very distorted, or there is more than one similar instantiation of the model in the search space, then the sum may never exceed this threshold and hence the algorithm will never terminate.

A more sophisticated termination condition is to use a stability criterion. That is, when [Q] remains constant over a number of iterations the algorithm is said to have reached equilibrium and hence may be terminated. This condition, though, is also not sufficient to guarantee correct operation, since until a mapping starts to diffuse across the cells, [Q] will be fairly constant at a value of zero or one.

A general technique that is used successfully is thus a combination of a threshold and the stability criteria. An initial activation threshold is set, which when reached triggers the start of stability monitoring.

5.5.2 Algorithm Analysis

Experimentation has shown that the Stochastic Search process is a Linear time algorithm, compared to the optimised Sequential Search method which is of Polynomial Time. Thus for a wide variety of object and Search Space sizes, the simple Stochastic Search algorithm is more efficient.

The analysis of the Stochastic Search can be divided into the analysis of two functions. The first is a function describing the number of iterations required for one or more cells to obtain the best mapping of the object into the Search Space - The Time to First Hit, (TH). The second is a function describing the number of iterations required to diffuse the correct mapping across all mappings cells - The Time for Diffusion, (TD).

Detailed computational analysis of the Stochastic Search technique is non trivial and to date has not been completed. This is due to difficulty in formally analysing the Time for Diffusion.

5.5.2.1 The Time to First Hit, (TH)

Given random initialisation and new mapping, (TH) can be analysed by simple statistical methods. In general new mappings into the Data Set are selected randomly. One iteration consists of [N] such random selections of mappings from a Search Space of [M] ADUs, of which one mapping is correct (the best fit). The probability of a correct mapping being found in one of [N] mapping cells, at iteration [i],

given a Search Space of size [M], is calculated as follows:

$$Q(i) = P\ (Success\ of\ ith\ selection\ in\ first\ iteration)$$
$$Q(i) = \frac{1}{M}$$

$$R(i) = P\ (Failure\ of\ ith\ selection\ in\ first\ iteration)$$
$$= 1 - Q(i)$$
$$= 1 - (\frac{1}{M})$$
$$= \frac{(M - 1)}{M}$$

$$X = P(Number\ of\ successes\ in\ first\ iteration\ of\ N\ trials)$$
$$= R^N$$
$$= \left(\frac{M - 1}{M}\right)^N$$

$$X(i) = P(Number\ of\ successes\ after\ i\ iterations)$$
$$X(i) = X^i$$
$$= \left(\frac{M - 1}{M}\right)^{Ni}$$

$$P(i) = P(at\ least\ one\ success\ after\ i\ iterations)$$
$$= 1 - X(i)$$
$$= 1 - \left(\frac{M - 1}{M}\right)^{Ni} \quad (1)$$

For a given P, the above can be re-written to define the iteration, *I*, at which the probability that the correct mapping has been found is P:

$$I = \frac{ln(1 - P)}{\left(N \times ln(\frac{M-1}{M})\right)} \quad (2)$$

From equation (1), it can be seen that P asymptotically approaches one. Hence the algorithm is *probabilistically* bounded, as this probability is never exactly one. It is also apparent that as the Time Ratio (N/M) approaches zero, the Time to first hit (TH) will grow exponentially (figure 5.8 & figure 5.9).

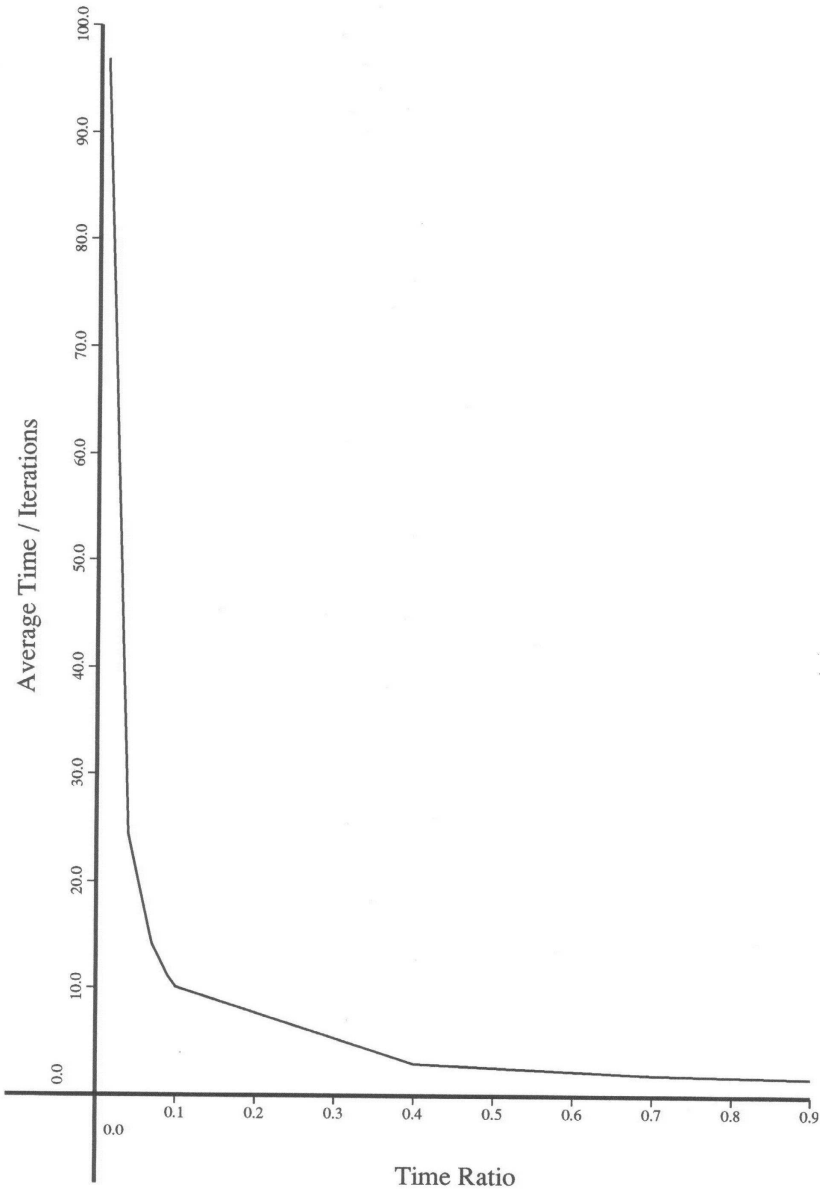


Figure 5.8 Variation in, 'Time to first hit'

5.5.2.1 Time of Diffusion, (TD)

It is desired to obtain an expression giving the probability that any mapping cell is TRUE (contains the correct/best mapping) at iteration (t = i), given one mapping cell from [N] is correct at (t = 0). However, to date it has not been possible to establish an accurate, complete description of the Diffusion process. However, an approximation to Diffusion behaviour can be obtained, using *expected value* estimations. Given that an event [X] occurs with probability p, over n trials the expected value for the number of occurrences of [X] is np.

Theorem:
Let P_m denote the probability that any cell is TRUE then:

$$P_m = 1 - \left(\frac{N-1}{N}\right)^{2^m} \tag{3}$$

Proof (*by induction on m*):

For (m = 0):

$$\begin{aligned} P_0 &= \frac{1}{N} \\ &= 1 - \frac{N-1}{N} \\ &= 1 - \left(\frac{N-1}{N}\right)^{2^0} \end{aligned}$$

Thus the statement is true for (m = 0). Assuming the statement is TRUE for (m=i), for some non negative integer:

$$\begin{aligned} \textit{Then } P_i &= 1 - \left(\frac{N-1}{N}\right)^{2^i} \\ Q_i &= 1 - P_i \\ &= \left(\frac{N-1}{N}\right)^{2^i} \end{aligned}$$

But $Q_{(i+1)}$ = (*The probability that the cell was false on iteration i*) × (*the probability that any other cell was false on that iteration*):

$$\begin{aligned} &= Q_i \times Q_i \\ &= \left(\left(\frac{N-1}{N}\right)^{2^i}\right)^2 \\ &= \left(\frac{N-1}{N}\right)^{(2^{(i+1)})} \\ \textit{Hence } P_i &= 1 - \left(\frac{N-1}{N}\right)^{(2^{(i+1)})} \end{aligned}$$

Hence the statement is true for (m = i + 1) and hence by induction the statement is true for all m, (*where m is a non negative integer*).

To derive [m], given P:

From (3),

$$P = 1 - \left(\frac{N-1}{N}\right)^{2^m} \quad \left(P_0 = \frac{1}{N}\right)$$

$$
\begin{aligned}
(1 - P) &= \left(\frac{N - 1}{N}\right)^{2^m} \\
2^m &= \frac{ln\,(1 - P)}{ln\left(\frac{N-1}{N}\right)} \\
m &= \frac{\frac{ln(1-P)}{\left(ln\frac{N-1}{N}\right)}}{ln\,(2)}
\end{aligned}
\tag{4}
$$

To test the accuracy of (3), a simple program was used to establish diffusion probabilities over a set of values for N, which could be compared with those from the estimated value model.

```
PROCEDURE TEST_DIFFUSE;
  VAR
    lastMap, MAP : ARRAY [1..N] OF BOOLEAN;
    AVERAGE : ARRAY [1..maxIters] OF INTEGER;
    cnt, iters : INTEGER;

  BEGIN

    FOR i := minMap TO N DO
      MAP [i] := FALSE;  {set one mapping cell TRUE }
    MAP [RANDOM (N)] := TRUE; (@ iteration zero }

    iters := 0;
    REPEAT
      iters := iters + 1;
      lastMap := MAP; cnt := 0;

      FOR i := minMap TO N DO
        IF lastMap [i] THEN
          cnt := cnt + 1
        ELSE
          MAP [i] := lastMap [ RANDOM (N) ];
    UNTIL (cnt = N);

    AVERAGE [iter] := AVERAGE [iter] + 1;

  END;
```

For each N, the weighted average of probabilities, averaged over one thousand trials, was calculated giving an approximation to expected diffusion time (when $P_i = 1$). The comparison of probabilities from the above process with those from the expected value model, show that the expected value model, although it does not accurately decribe the system, is of the same computational order $O\,ln\,(N)$, (see tables 5.1, 5.2 and figure 5.10a).

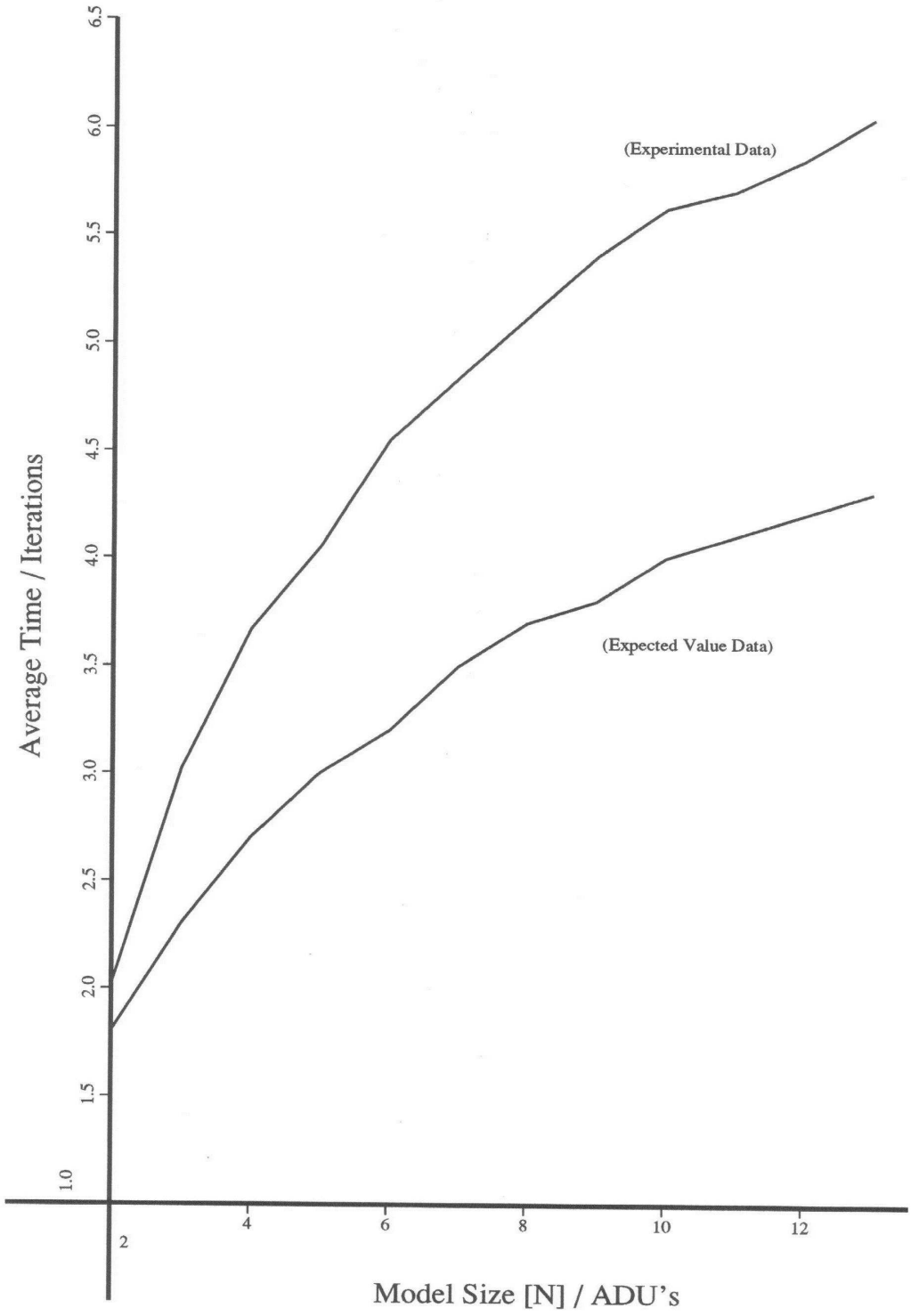


Figure 5.10a Comparison with Expected Value diffusion model

	i = 1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**mean**
N = 2	*0.49*	*0.75*	*0.86*	*0.93*	*0.97*	*0.98*	*0.99*	*1.00*	*1.00*	*2.03*
3	*0.11*	*0.46*	*0.71*	*0.85*	*0.92*	*0.96*	*0.98*	*0.99*	*1.00*	*3.02*
4	*0.02*	*0.24*	*0.53*	*0.76*	*0.88*	*0.94*	*0.98*	*0.98*	*1.00*	*3.67*
5	*0.04*	*0.13*	*0.40*	*0.67*	*0.83*	*0.92*	*0.97*	*0.99*	*1.00*	*4.05*
6	*0.00*	*0.04*	*0.26*	*0.55*	*0.75*	*0.88*	*0.95*	*0.98*	*0.99**	*4.55*
7	*0.00*	*0.02*	*0.18*	*0.43*	*0.71*	*0.87*	*0.94*	*0.97*	*0.99**	*4.84*
8	*0.00*	*0.01*	*0.12*	*0.34*	*0.63*	*0.82*	*0.92*	*0.96*	*0.98**	*5.12*
9	*0.00*	*0.00*	*0.06*	*0.28*	*0.56*	*0.79*	*0.91*	*0.96*	*0.99**	*5.40*
10	*0.00*	*0.00*	*0.04*	*0.23*	*0.51*	*0.73*	*0.87*	*0.93*	*0.98**	*5.62*
11	*0.00*	*0.00*	*0.02*	*0.19*	*0.46*	*0.72*	*0.87*	*0.94*	*0.97**	*5.70*
12	*0.00*	*0.00*	*0.01*	*0.14*	*0.42*	*0.69*	*0.86*	*0.94*	*0.97**	*5.85*
13	*0.00*	*0.00*	*0.01*	*0.11*	*0.36*	*0.64*	*0.82*	*0.93*	*0.97**	*6.04*
25	0.31	0.56	0.78	0.89	0.97	0.98	1.00	1.00	1.00	7.45
40	0.05	0.32	0.60	0.83	0.94	0.97	0.99	1.00	1.00	8.30
55	0.02	0.14	0.44	0.71	0.89	0.96	0.97	0.99	1.00	8.86
70	0.00	0.06	0.31	0.61	0.81	0.90	0.96	0.98	0.99*	9.32
85	0.00	0.01	0.18	0.50	0.70	0.91	0.96	0.99	0.99*	9.71
100	0.00	0.00	0.13	0.47	0.70	0.87	0.95	0.99	1.00	9.88

Table 5.1. Experimental probabilities that a cell will be TRUE at iteration [i] ($2 \leqslant i \leqslant 13$). Results in italics averaged over 1000 trials, other results averaged over 250 trials.

	i = 1	2	3	4	5	6	7	8	9	mean
N = 2	0.50	0.75	0.94	1.00	1.00	1.00	1.00	1.00	1.00	1.81
3	0.33	0.56	0.80	0.96	1.00	1.00	1.00	1.00	1.00	2.35
4	0.25	0.44	0.68	0.90	0.99	1.00	1.00	1.00	1.00	2.74
5	0.20	0.36	0.59	0.83	0.97	1.00	1.00	1.00	1.00	3.01
6	0.17	0.31	0.52	0.77	0.95	1.00	1.00	1.00	1.00	3.28
7	0.14	0.26	0.46	0.71	0.91	0.99	1.00	1.00	1.00	3.53
8	0.12	0.23	0.41	0.66	0.88	0.99	1.00	1.00	1.00	3.71
9	0.11	0.21	0.38	0.61	0.85	0.98	1.00	1.00	1.00	3.86
10	0.10	0.19	0.34	0.57	0.81	0.97	1.00	1.00	1.00	4.02
11	0.09	0.17	0.32	0.53	0.78	0.95	1.00	1.00	1.00	4.11
12	0.08	0.16	0.29	0.50	0.75	0.94	1.00	1.00	1.00	4.28
13	0.08	0.15	0.37	0.47	0.72	0.92	0.99	1.00	1.00	4.30

Table 5.2. Expected Value probabilities that a cell will be TRUE at iteration [i] ($2 \leqslant i \leqslant 13$).

The accuracy of (3) and (4) is limited by the behaviour of the system when the probability at iteration (i) differs from its predicted expected value by an error E. The results show that the error due to this effect is significant and causes the expected value model to underestimate Diffusion time.

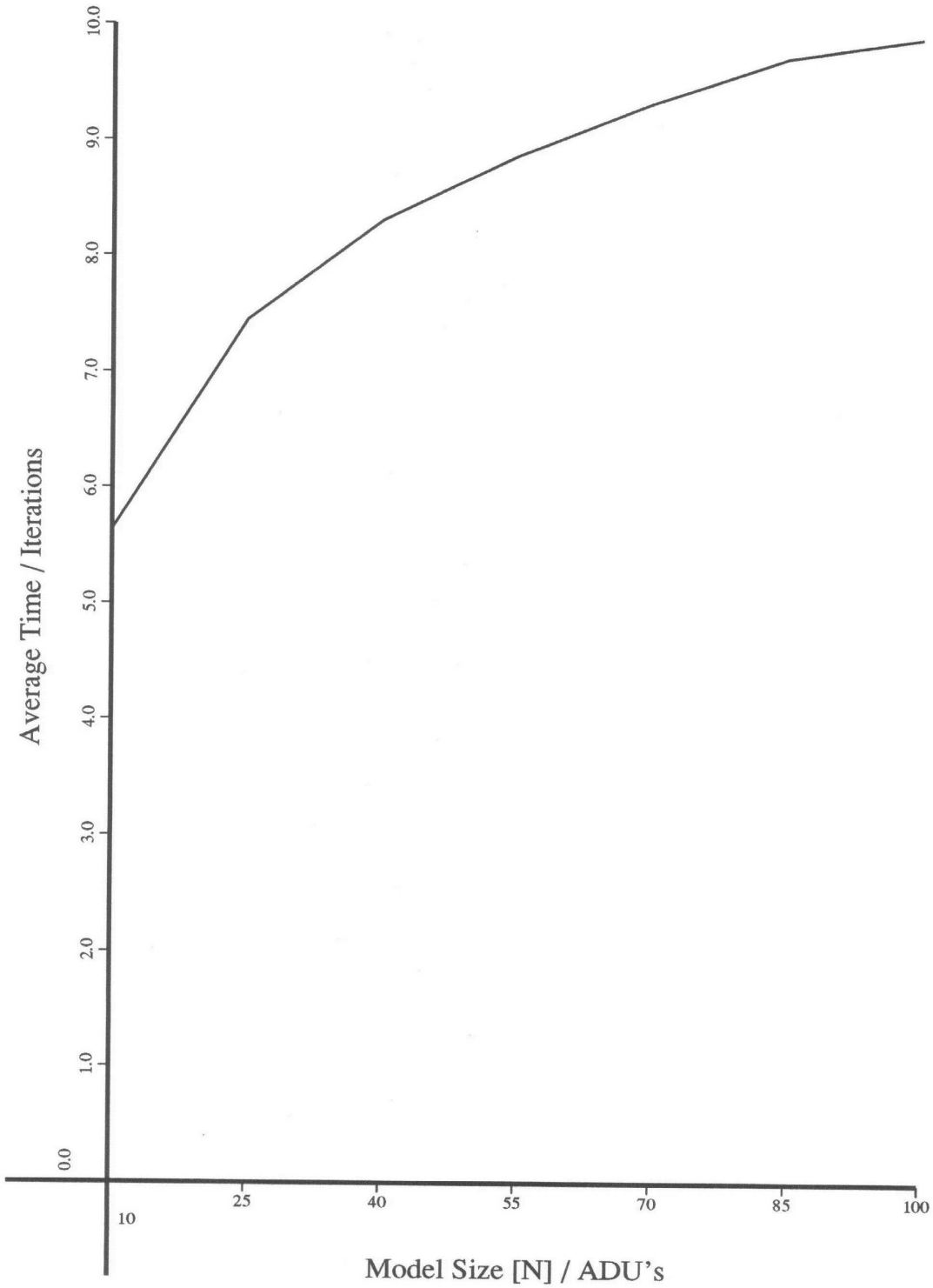


Figure 5.10b Experimental diffusion behaviour

To obtain a better description of Diffusion time, a graph was plotted over the range of N ($10 \leqslant N \leqslant 100$)

(figure 5.10b) and an approximation to Diffusion behaviour obtained by regression analysis.

$$TD = 5.63 + 1.85 \times (ln(X) - ln(10)) \quad (5)$$

Although a complete, accurate, algebraic analysis of TD has not been found, a recursive function describing the probability of [n] cells from a set of [N] mapping cells being true at iteration [i], has been derived for ($2 \leqslant N \leqslant 13$) - the *Recursive Model*. Its use is limited by its application of the factorial function on numbers of size [N], and by the recursive structure of the algorithm whose computation time expands exponentially with [N].

Given [nNow] cells correct at iteration zero, PROB returns the probability of [nStop] cells being correct at iteration [iStop].

Definitions

```
N :              Number of mapping cells
nNow :           Number of cells correct at current iteration
iNow :           Current iteration number
pNow :           Probability of nNow cells being correct at iteration iNow
```

Function Description

```
PROB (N, nNow,nStop, iNow,iStop: INTEGER; pNow: REAL): REAL;
  BEGIN

  IF (iNow = iStop) THEN
    IF (nNow = nStop) THEN
      prb := pNow
    ELSE
      prb := 0
  ELSE BEGIN
    p := nNow / N;
    q := 1 - p;
    pNow := 0;
    FOR i := 0 TO (nStop - nNow) DO BEGIN
      pI := pNow * nCr(N - nNow, i);
      pI := pI * POWER(p, i) * POWER(q, (N - nNow + i));
      prb := prb+PROB(N,(nNow+i),nStop,(iNow+1),iStop,pI);
    END {i};
  END {if};

  PROB := prb;
END; {prob}
```

The recursive model does accurately describe experimental behaviour, as shown in table (5.3) and figure (5.11).

	i = 1	2	3	4	5	6	7	8	9	mean
N = 2	0.50	0.75	0.87	0.94	0.97	0.98	0.99	1.00	1.00	2.00
3	0.11	0.46	0.71	0.85	0.93	0.97	0.98	0.99	1.00	3.00
4	0.02	0.22	0.54	0.76	0.88	0.94	0.98	0.99	0.99*	3.62
5	0.00	0.10	0.38	0.65	0.83	0.92	0.96	0.98	0.99*	4.13
6	0.00	0.04	0.23	0.55	0.77	0.89	0.95	0.98	0.99*	4.55
7	0.00	0.02	0.17	0.45	0.70	0.86	0.94	0.97	0.99*	4.85
8	0.00	0.01	0.11	0.37	0.64	0.83	0.92	0.97	0.99*	5.11
9	0.00	0.00	0.06	0.29	0.58	0.79	0.91	0.96	0.98*	5.34
10	0.00	0.00	0.04	0.23	0.52	0.76	0.89	0.95	0.98*	5.54
11	0.00	0.00	0.02	0.18	0.47	0.72	0.87	0.95	0.98*	5.72
12	0.00	0.00	0.01	0.14	0.41	0.68	0.85	0.94	0.97*	5.88
13	0.00	0.00	0.01	0.10	0.36	0.65	0.83	0.93	0.97*	6.03

Table 5.3. Recursive Model probabilities that a cell will be TRUE at iteration [i] (2 ⩽ N ⩽13).

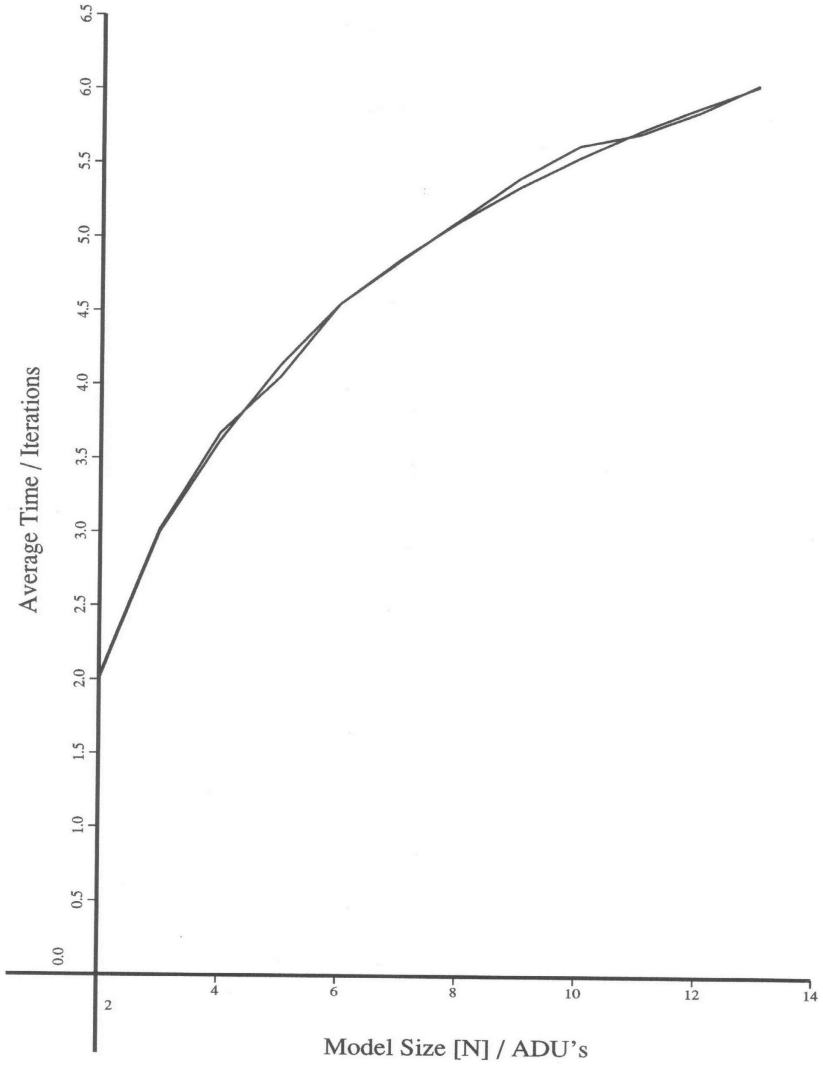


Figure 5.11 Comparison with recursive diffusion model

5.6 Investigation of Basic Stochastic Search Algorithm

Two factors directly influence the performance of stochastic search, the Time Ratio (TR), (section 5.5a), and the probability of an erroneous comparison (section 5.5b). The effect of these factors is described in the following sections.

5.6.1 Time Ratio less than one (TR < 1)

To investigate TR less than one, a random search space [ss] was generated comprising of an array of M digits. A data model [md] of N integers was extracted from positions [x] to [x + N - 1] (N < M). The problem was to determine [x], given the search space with the model embedded in it. This is a problem analogous to finding the position of a grey level encoded object in a grey level image. The experiment consisted of calculating the time to determine [x] given [md] and [ss].

The experiment was performed on a SUN workstation and the results were averaged over one thousand trials, to limit the effect of statistical fluctuations. M was varied logarithmically over a range [100] to [15411], and TR varied linearly over the range [0.1 .. 0.9]. The time in seconds and the number of iterations required to find [x] were noted.

Although the stochastic search is of lower computational order than the optimised sequential search, computational overheads cause it to be less efficient for some values of M and N, when used on a sequential computer. To establish this range, the performance of the stochastic search was compared with that from an optimised sequential search, the results are listed in table (5.4).

	TR = 0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
M = **100** iters:	18.8	13.9	13.2	12.1	11.2	10.9	10.4	9.64	8.45
stoc:	0.03	0.05	0.06	0.08	0.08	0.09	0.10	0.11	0.10
seq:	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00
205 iters:	20.8	16.1	13.5	13.3	12.4	11.9	11.4	10.8	9.58
stoc:	0.06	0.12	0.13	0.16	0.18	0.21	0.23	0.25	0.24
seq:	0.01	0.02	0.03	0.03	0.03	0.03	0.03	0.02	0.01
422 iters:	22.6	16.3	14.6	13.9	13.7	13.1	12.6	11.9	10.9
stoc:	0.17	0.22	0.28	0.36	0.42	0.48	0.54	0.56	0.55
seq:	0.05	0.10	0.11	0.13	0.15	0.15	0.13	0.10	0.05
866 iters:	22.4	17.1	15.9	15.1	14.8	14.1	13.7	13.0	12.0
stoc:	0.30	0.50	0.64	0.81	0.96	1.09	1.23	1.32	1.29
seq:	0.23	0.42	0.53	0.55	0.62	0.54	0.51	0.41	0.20
1779 iters:	23.1	18.1	16.6	15.8	15.9	15.2	14.6	14.1	13.0
stoc:	0.68	1.03	1.45	1.78	2.12	2.48	2.73	2.93	2.94
seq:	0.90	1.62	2.08	2.48	2.55	2.66	1.94	1.79	0.95
3654 iters:	23.2	20.0	17.5	17.2	17.0	16.2	15.8	15.0	14.1
stoc:	1.41	2.19	3.14	3.93	4.74	5.49	6.01	6.57	6.88
seq:	3.67	7.47	9.02	10.2	11.2	10.2	8.04	6.77	3.79
7504 iters:	23.5	20.2	19.1	18.2	17.6	17.3	16.8	16.2	15.1
stoc:	2.96	5.12	6.88	8.69	10.4	12.2	13.7	14.8	15.4
seq:	15.7	28.1	38.5	40.3	43.1	45.6	38.7	26.7	17.2
15411 iters:	26.5	21.1	20.3	19.2	18.7	18.2	17.9	17.3	16.1
stoc:	6.38	10.8	14.8	18.8	23.1	26.8	30.2	32.3	34.2
seq:	73.7	117	168	174	184	186	172	134	69.3

Table 5.4. Performance of Stochastic Search and Optimised Sequential Search Algorithms, (CPU time in seconds; and iterations of the stochastic search).

For each value of TR tested, there is a value of [M], the *breakthrough value*, after which the stochastic search became more efficient (for this particular implementation). This value can be established graphically, (see figure 5.12). A list of values of breakthrough values, for the range of TR tested, is given in table (5.5) and plotted in figure (5.13).

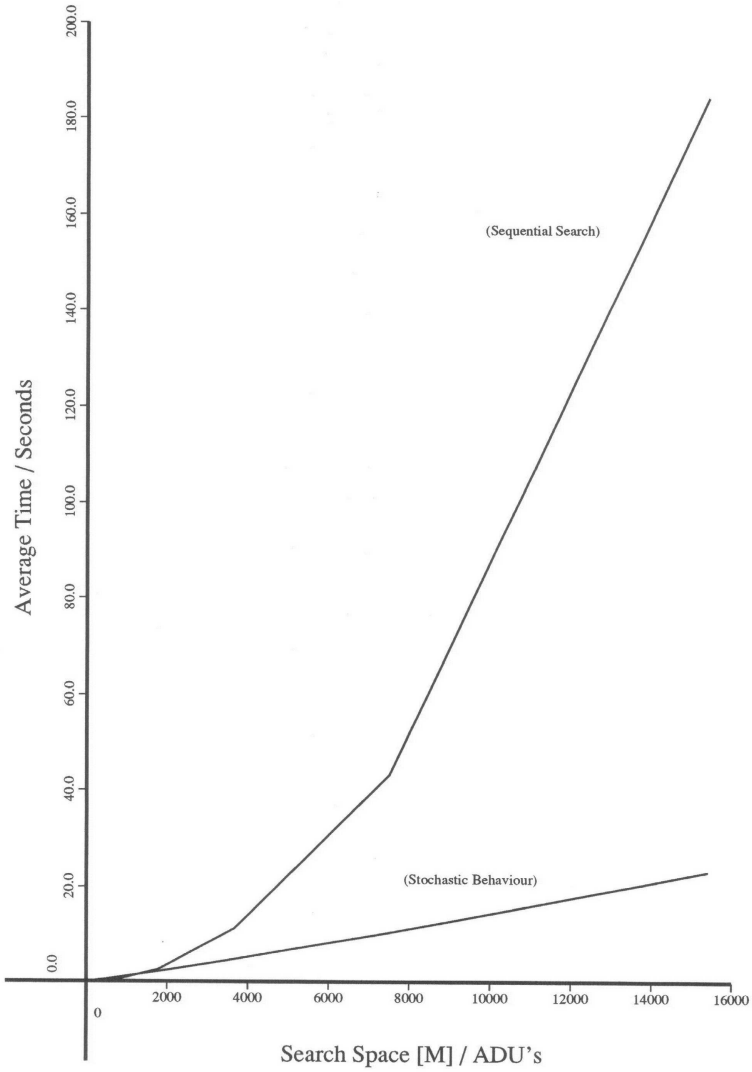


Figure 5.12 Stochastic & sequential search (TR = 0.5)

TR	**Minimum [M]**
0.1	2280
0.2	1570
0.3	1850
0.4	2100
0.5	2390
0.6	2880
0.7	4350
0.8	6470
0.9	9670

Table 5.5. Results showing the minimum size of search space [M], when the stochastic search, (running on a SUN Workstation), uses less CPU time than the optimised sequential search, for a given Time Ratio, TR.

From figure (5.13) it can be seen that stochastic search becomes less efficient at low values of TR. This is due to the time to first hit becoming dominant, (see figure 5.9a). The use of a different strategy other than random mapping, (such as linear spacing of samples), into the search space may reduce this problem.

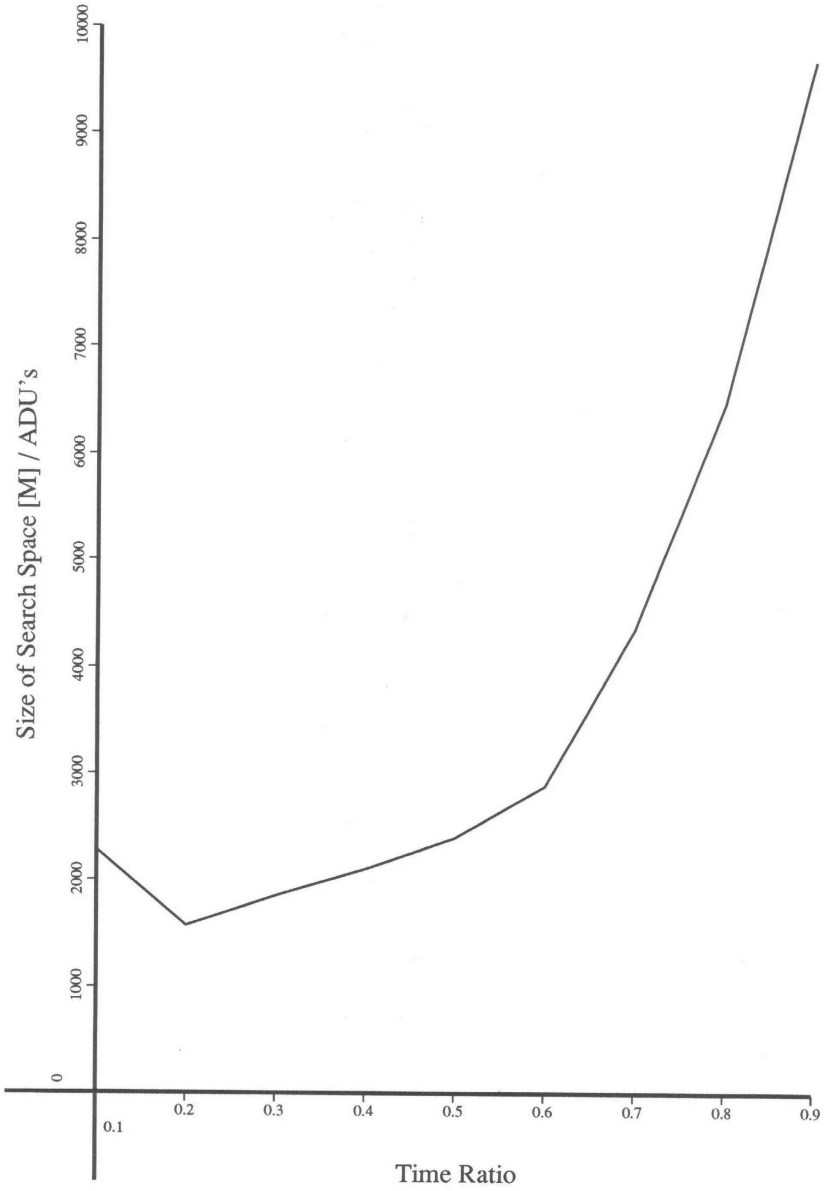


Figure 5.13 Breakthrough values of [M] for stochastic search

Figure (5.13) shows that stochastic searching, using a standard sequential implementation, also becomes less efficient at high values TR (TR < 1), where the computational overheads, implicit in the stochastic search, dominate compared to the simpler sequential search.

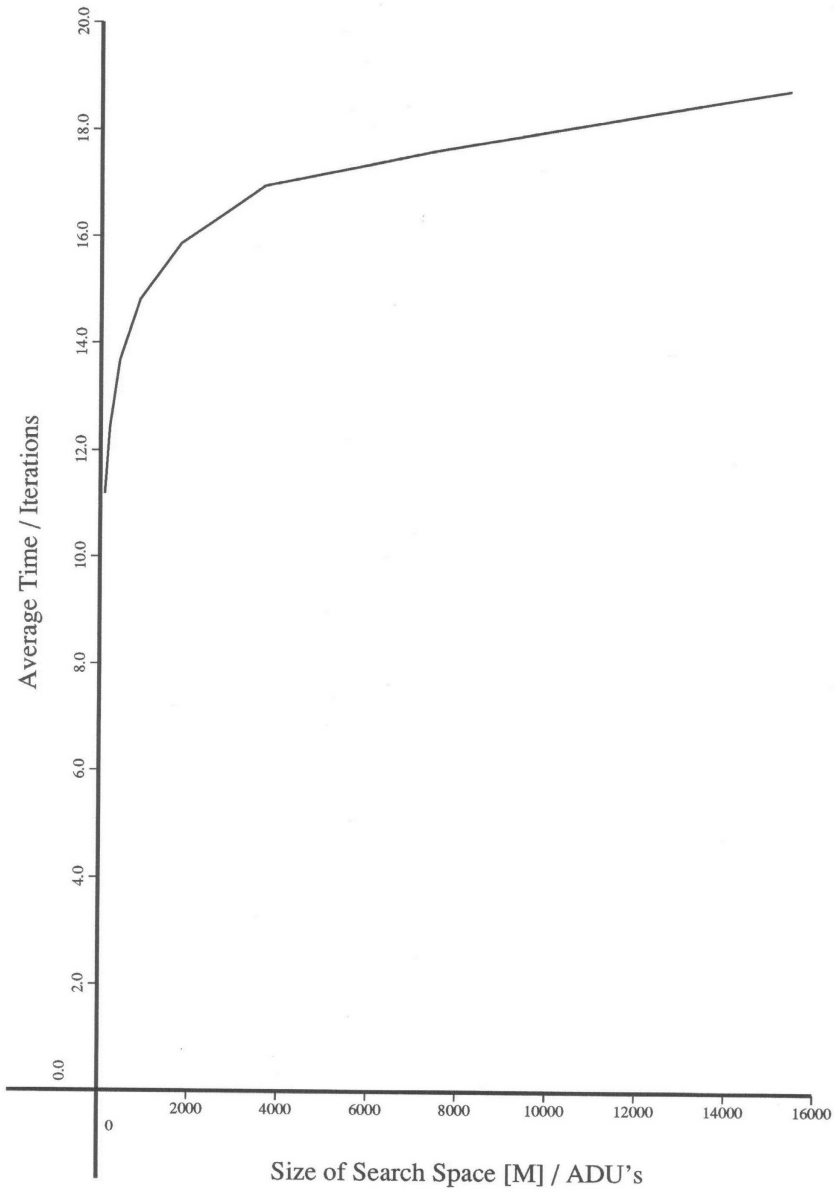


Figure 5.14 Performance of stochastic search, (TR = 0.5)

A graph showing the number of iterations required for termination plotted against M, for a range of TR, is given in figure (5.14). This shows that a parallel implementation of the algorithm would be probabilistically bounded by $O(ln(M))$ for a given TR.

5.6.2 Time Ratio greater than one (TR > 1)

To investigate TR greater than one, a series of [M] random patterns were generated - the search space [ss], each comprising of an array of [N] digits. The problem was to identify the pattern [x] which best fitted a data model [md] of N integers. This is a problem analogous to classifying an object of size [N] ADUs into one of [M] sets (N > M). The experiment consisted of calculating the time to determine [x] given [md] and [ss].

The experiment was performed on a SUN workstation and the results were averaged over one hundred trials, to reduce the effect of statisitical fluctuations. N was varied logarithmically over a range [100] to

[65536] and TR varied over the values [3.33, 1.96, 1.11]. The time in seconds, and the number of iterations, required to find [x] were noted, (see table 5.6).

	TR = 3.30	**1.96**	**1.11**
N = 100	M = 30	51	90
Iterations	15.3	16.3	17.9
Time	0.20	0.22	0.25
N = 505	M = 151	262	454
Iterations	15.4	16.3	17.1
Time	1.15	1.25	1.34
N = 2560	M = 768	1330	2304
Iterations	17.6	18.5	19.6
Time	7.17	7.64	8.24
N = 12952	M = 3885	6730	11656
Iterations	20.2	21.2	22.0
Time	43.4	46.3	48.5
N = 65536	M = 19660	34053	58982
Iterations	22.7	23.6	24.5
Time	255	269	284

Table 5.6. Performance of Stochastic Search for various Time Ratios greater than one, ($1.11 \leqslant TR \leqslant 3.30$), giving CPU timings (on a SUN workstation) and iterations of the algorithm.

NB. Due to the large storage that would be required to hold 58982 patterns of 64K bytes, (3.9 GByte) these experiments were simulated. This was accomplished by randomly generating [x] in the range [1..M], then recording how quickly the search converged onto [x], given a specific sample error probability.

```
TEST (VAR MC : Mapping_Cells);
BEGIN
  FOR i := 1 to N DO
    IF (MC [i].map = [x]) OR (ERROREOUS) THEN
      MC [i].cFire := TRUE
    ELSE
      MC [i].cFire := FALSE;
END;
```

The function ERRONEOUS returns a random boolean value, determined by the sample error probability. This simulates the possibility that any given individual atomic comparison may give a misleading result.

Figure (5.15a) shows the relative performance of the stochastic search in seconds, for the time ratio (TR = 3.3), for the range ($100 \leqslant M \leqslant 65536$). From the graph it can be seen that this relationship is linear in [M].

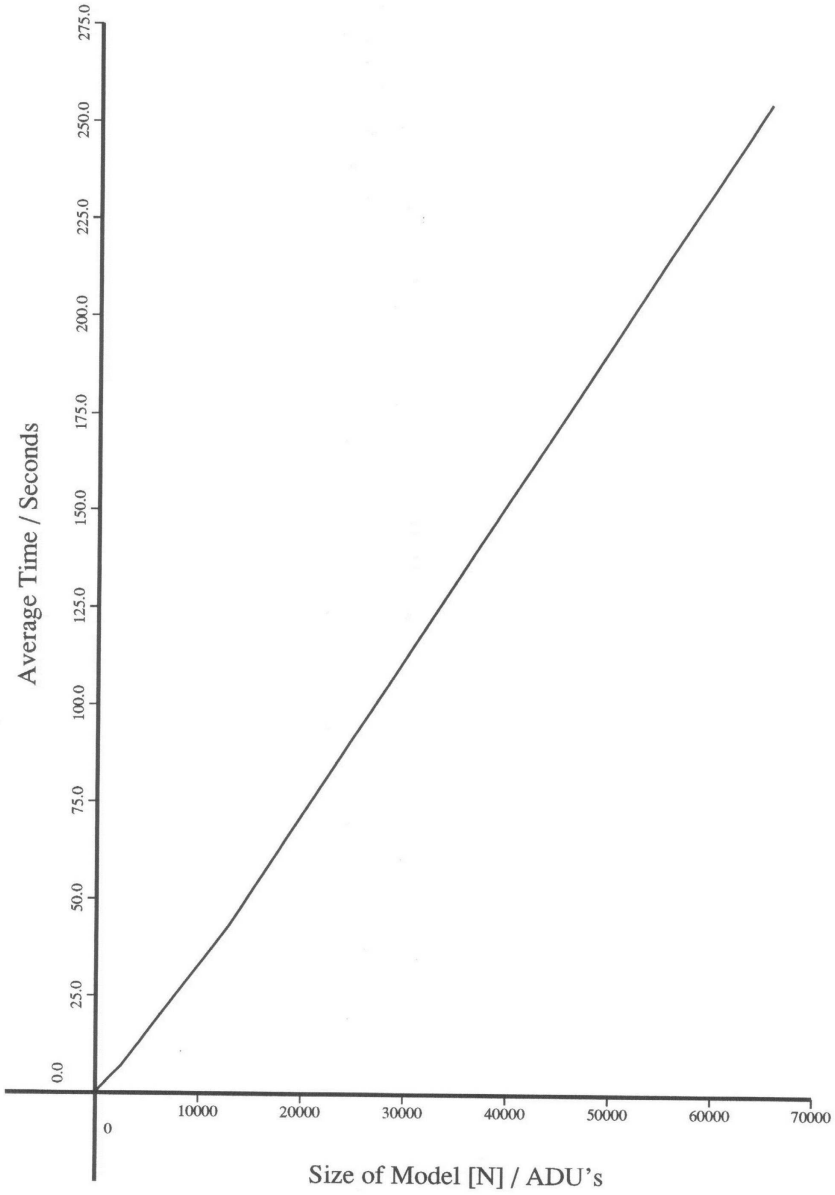


Figure 5.15a Performance of stochastic search, (TR = 3.3)

This investigation is analogous to the problem of finding the best fit of a grey level image from one of [M] sets. The performance of the algorithm suggests possible uses in applications such as fingerprint analysis, since the best fit of an [256*256] image from one of 58982 instances, was found in around 5 minutes! This performance was achieved on a conventional uni-processor workstation, with the program coded in a standard high level language!

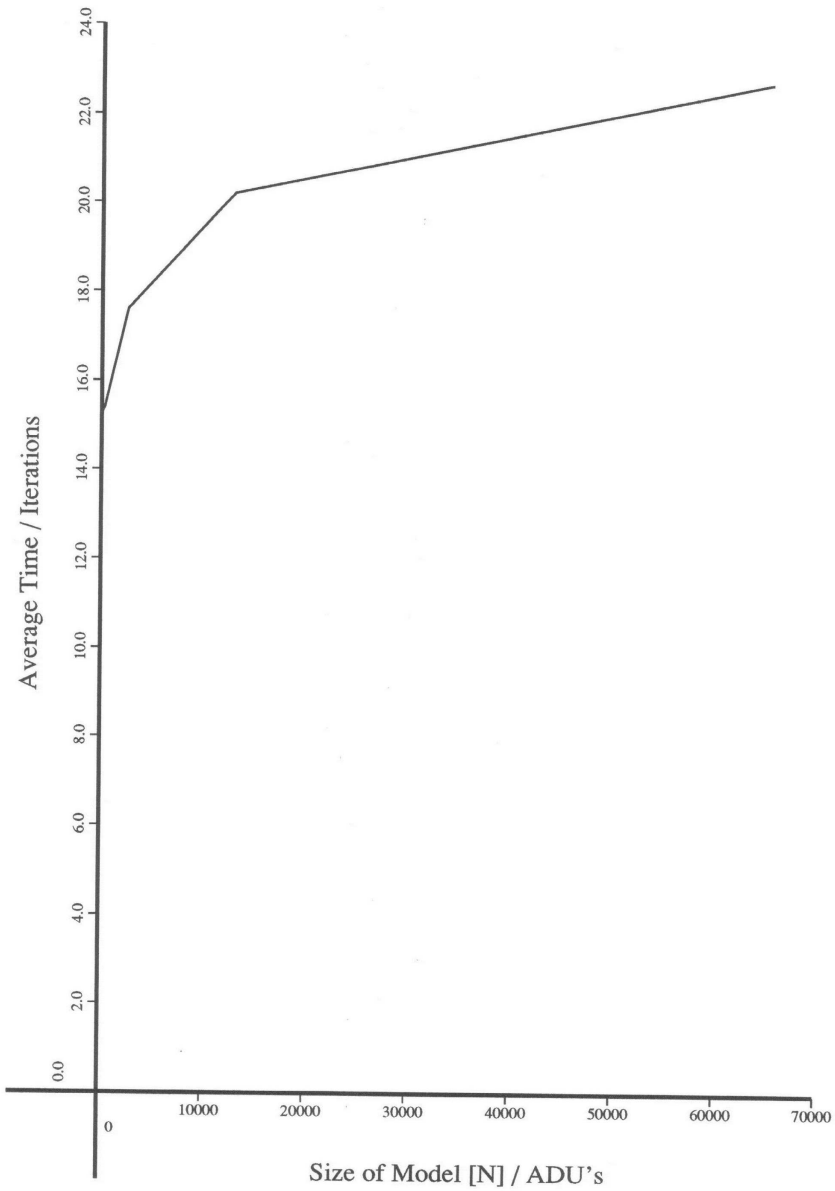


Figure 5.15b Performance of stochastic search, (TR = 3.3)

Figure (5.15b) plots the relative performance of the stochastic search in iterations, for the time ratio, (TR = 3.3), over the logarithmic range ($100 \leqslant M \leqslant 65536$). From the graph it can be seen that this relationship is logarithmically linear in [M], from (M = 505) onwards. The initial non-linearity is possibly due to an amplification of statistical fluctuations due to the small model and search space sizes.

5.6.3 The Effect of Noise, Resolution and Similarity

The effect of the following factors on algorithm performance was investigated using a time ratio greater than one.

1: Noise

This is the degree of distortion of the model within the search space. A value of zero implies that the model is not distorted at all, a value of one indicates that each element of the model has been distorted.

2: Resolution

Resolution describes the range of symbols over which an atomic comparison takes place. A resolution of two indicates that there are two states each ADU can be in (eg. a boolean), a resolution of ten indicates there are ten (eg. a decimal digit).

3: Similarity

This is the degree to which each of the [M] elements of the search space are the same as each other. A similarity of one indicates that the Hamming Distance between each element is zero, a similarity of zero indicates that there is no similarity between the elements of the search space.

5.6.3.1 Effect of Noise

The effect of noise on time to termination of the stochastic search algorithm was investigated. A system was configured with a time ratio of [1.25], an atomic resolution of ten with zero forced similarity between object class instances.

The effect of noise was investigated over the range [0% .. 50%] of the model size. The experiments were averaged over one hundred trials and are listed in table (5.7) and plotted in figure (5.16). The results show an approximately logarithmic relationshiop between noise and time to termination.

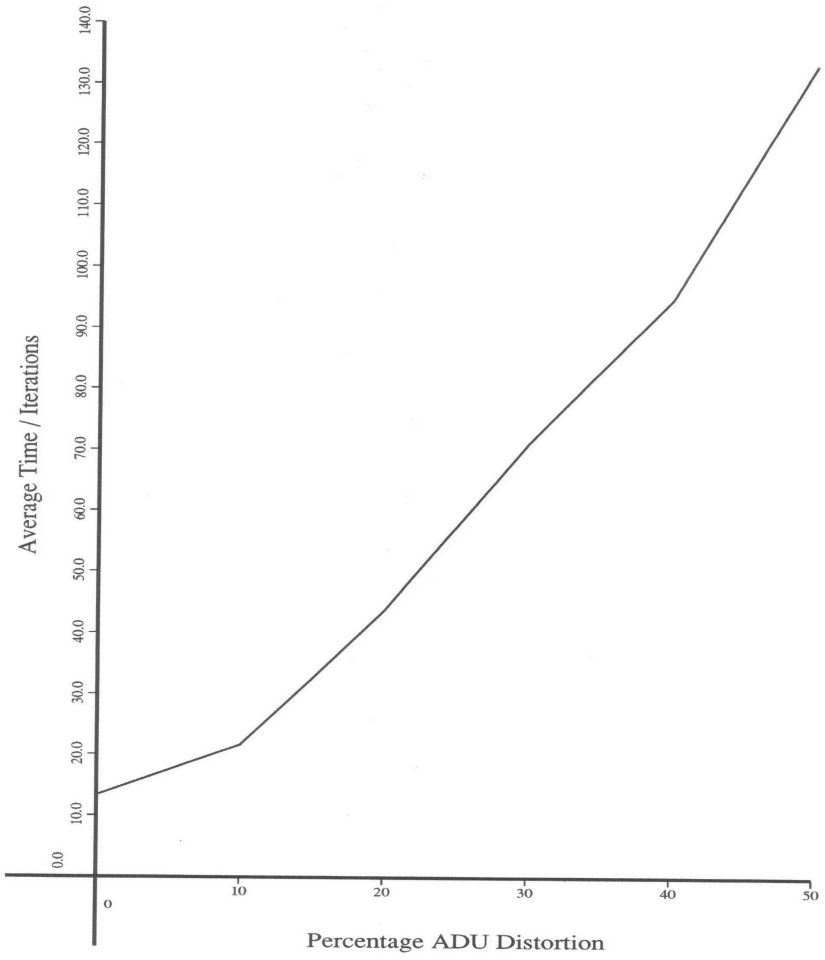


Figure 5.16 Performance of stochastic search, (TR = 1.25)

ADU Distortion %	Time (in iterations)
0	13.28
10	21.58
20	43.72
30	71.12
40	94.79
50	133.17

Table 5.7. Effect of ADU Noise on Stochastic Search, (TR = 1.25; Resolution = 10; Similarity = 0).

5.6.3.2 Effect of ADU Resolution

The effect of Resolution on time to termination of the stochastic search algorithm was investigated. A system was configured with a time ratio of [1.25], noise of ten percent with zero forced similarity between object class instances.

The effect of variation in ADU resolution was investigated over the range [2..24]. The experiments were averaged over one hundred trials and are listed in table (5.8) and plotted in figure (5.17). The results show a rapid fall in time of termination for a resolution between two and seven, from a high of 124 to 27.

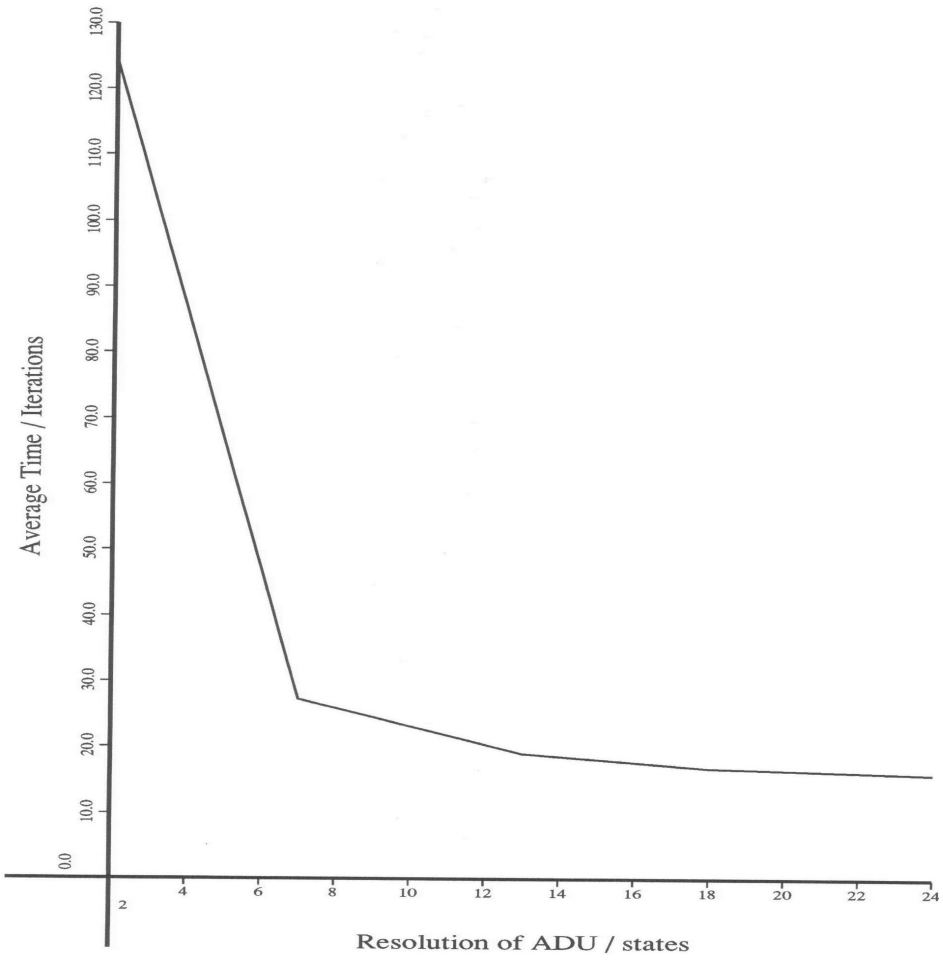


Figure 5.17 Performance of stochastic search, (TR = 1.25)

ADU Resolution	Time (in iterations)
2	124.45
7	27.24
13	18.95
18	16.81
24	15.90

Table 5.8. Effect of ADU Resolution on Stochastic Search, (TR = 1.25; Noise = 10%; Similarity = 0).

5.6.3.3 Effect of ADU Similarity

The effect of ADU Similarity on time to termination of the stochastic search algorithm was investigated. A system was configured with a time ratio of [1.25], noise of zero and an ADU resolution of ten.

The effect of variation in ADU similarity was investigated over the range [0..90]. The experiments were averaged over one hundred trials and are listed in table (5.9) and plotted in figure (5.18). The results show a rapid rise in time to termination as object instance similariy approaches 100%.

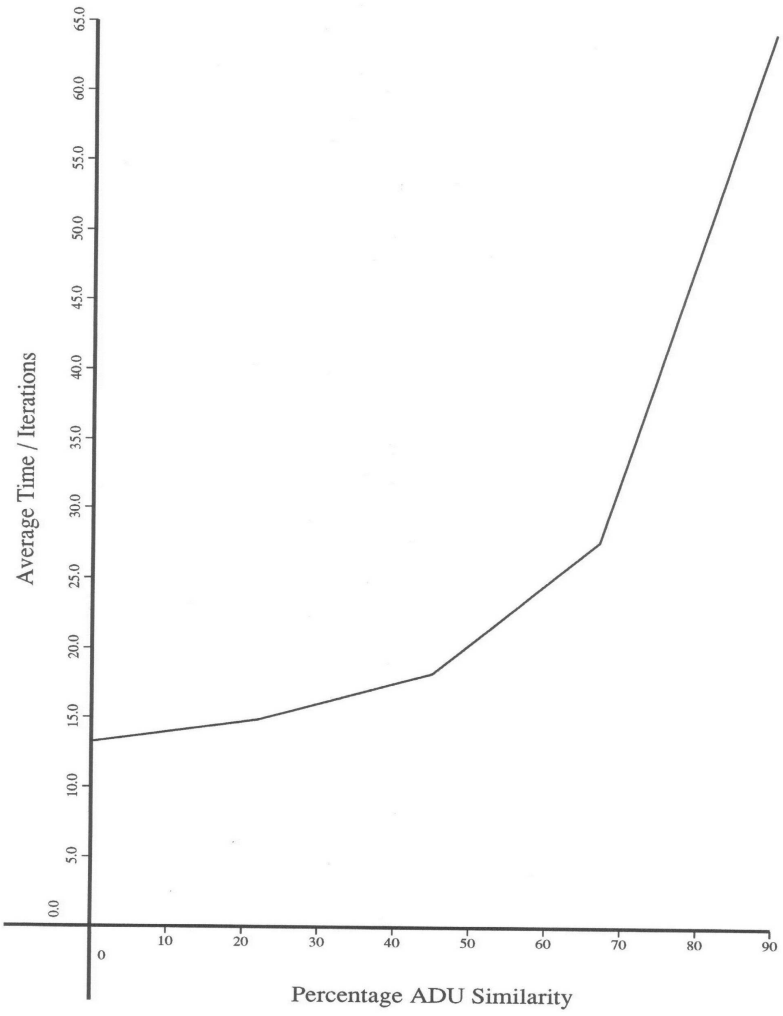


Figure 5.18 Performance of stochastic search, (TR = 1.25)

ADU Similarity	Time (in iterations)
0	13.23
22	14.86
45	18.23
67	27.64
90	64.12

Table 5.9. Effect of ADU Similarity on Stochastic Search, (TR = 1.25; Noise = 0%; Resolution = 10).

5.6.3.4 The Influence of Data Structure on Algorithm Performance

As described in earlier sections, there are several 'environmental' factors which influence algorithm performance, such as Noise (5.7.3) and Time Ratio (4.7.2). However the structure of the data can also adversely influence the performance of the algorithm, since it determines the probability of an Erroneous Test Comparison (5.5b).

Data effects are very apparent in the problem of locating a uniformly structured shape - for example, locating a solid black shape on a white background. Problems occur because there will be many positions which may give a positive ADU comparison over a number of cycles. Consider the following, one dimensional, problem of locating a given model with boolean ADUs;

```
Model:                    ++++++++              (size = N)
Search Space:          ———++++++++———           (size = M)
Mappings:                 abcdefgh
```

It can be seen that the correct mapping is [a], however mappings [b..h] may also give a positive comparison dependent on the 'tuple size' of the test. The tuple size is the number of individual ADUs which are compared in one 'Test Comparison'.

Given a tuple size of one, where each test of a mapping cell (MC) involves comparing one ADU from the model with one ADU from the search space, the mapping [b] will produce a positive comparison with probability $\frac{N-1}{N}$, mapping [c], with probability $\frac{N-2}{N}$, etc. Given a tuple size of T, [b] will yield a positive test result with probability $\left(\frac{N-1}{N}\right)^T$ and [c] with $\left(\frac{N-2}{N}\right)^T$, etc. Hence, increasing tuple size reduces exponentially the probability of such an erroneous test comparison.

To investigate data effects practically, a series of experiments were carried out with the aim of establishing the probability distributions of positional error in mapping cells over the iterations of the algorithm. These results can then be compared with those from a more usual scenario, where the data model has some degree of variety (such as a randomly generated data model).

To find the distribution of MC error probabilities over time, it is necessary to store the cell errors at each iteration. For a one dimensional search space, cell error is defined as ABS (Model position - cell mapping). In general, for a model of size [N], errors will be in the range [0 .. (M-1)] if there is 'wrap around' in the search space, or [0 .. (M-N-1)] if not (the N is due to clipping at the search space boundary).

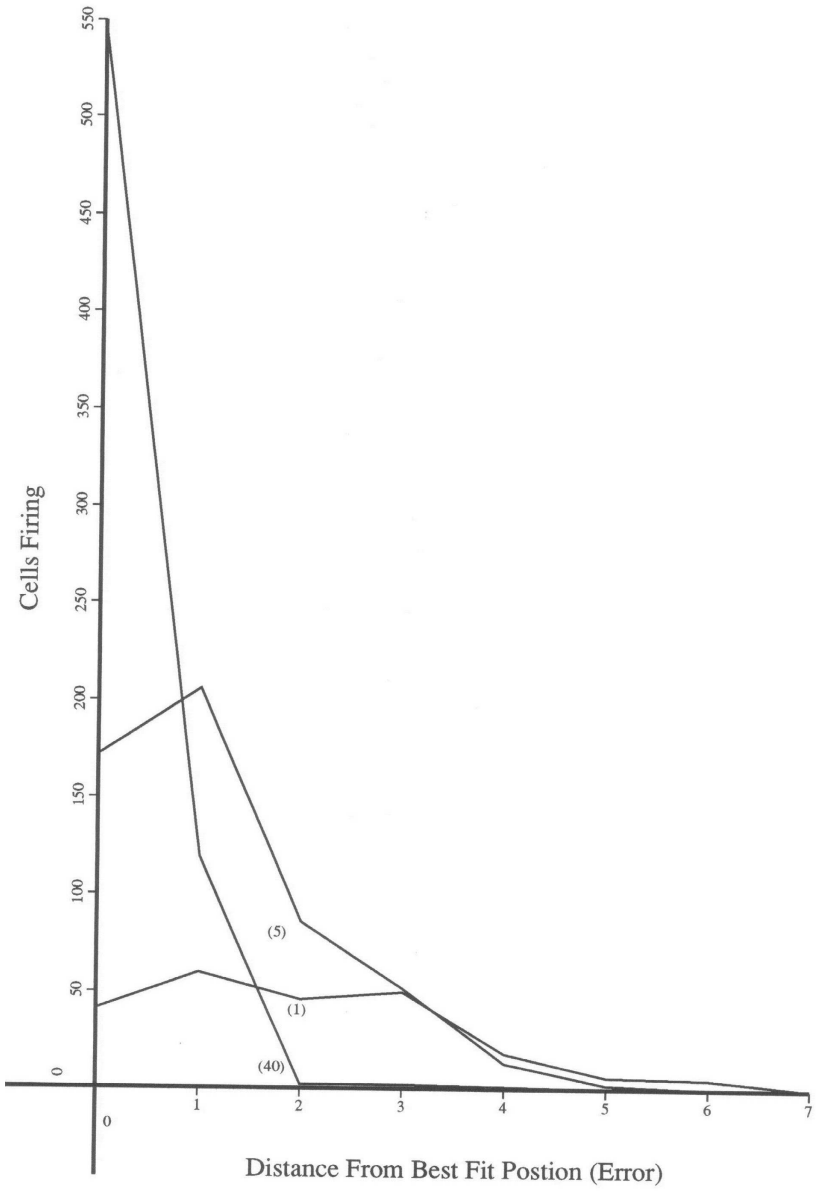


Figure 5.19 Error distribution, (uniform model)

The experiment consisted of recording cell errors over a hundred iterations and averaged over a hundred trials, using a tuple size of one, and a Time ratio of [0.35]. The resulting cell error distributions (figure 5.19) can be compared with those obtained in a conventional scenario, using a random model with ADU resolution ten, figure (5.20). The graphs show that the random model distribution is sharper - the algorithm locates the precise position of the model very quickly. After five iterations the number of cells firing with incorrect mappings is very low (less than fifty). This is compared with the uniform model, where after five iterations 359 cells were still firing with incorrect mappings.

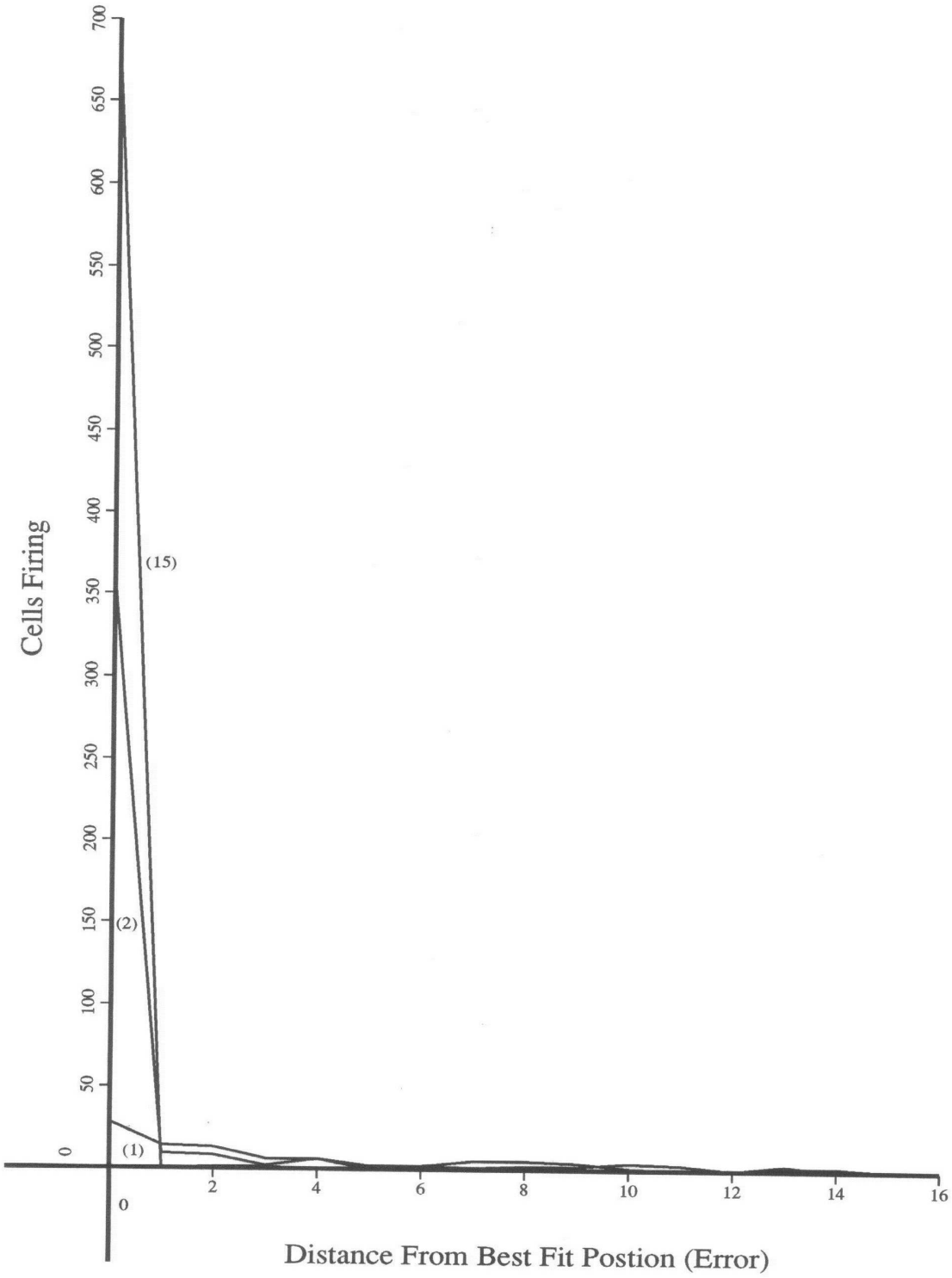


Figure 5.20 Error distribution, (standard model)

From the cell error distributions it is possible to calculate the expected cell error at each iteration. This was evaluated using three different tuple sizes (1,2 & 4) when searching for the random and uniform data models. The results are shown in figures (5.21) and (5.22) for the uniform and random models respectively.

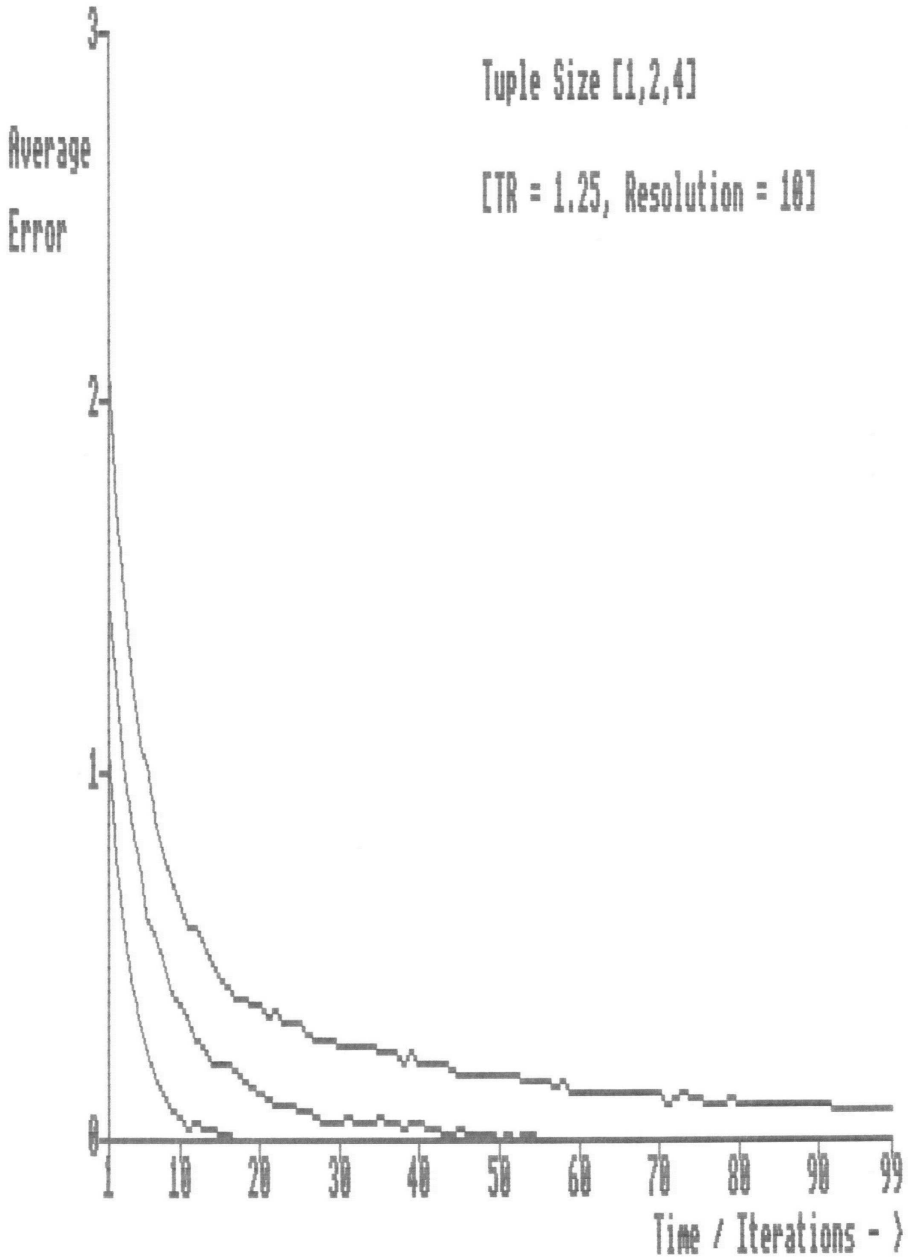


Figure 5.21 Average error, (uniform model)

A comparison of the expected error graphs shows that the uniform model takes much longer to find; using a tuple size of one, there is still a significant average cell error after one hundred iterations. This can be compared with data from the random model when using a tuple size of one, to show that the error has decayed to an insignificant level after only 15 iterations.

Both graphs (5.21) & (5.22) also illustrate the effect of raising tuple size, which reduces the time for convergence. This occurs because, in both cases, increasing the tuple size reduces the probability of an erroneous Test comparison by an exponential factor - as for simple n-tuple nets, *raising the tuple size increases the discrimination of the test comparison.*

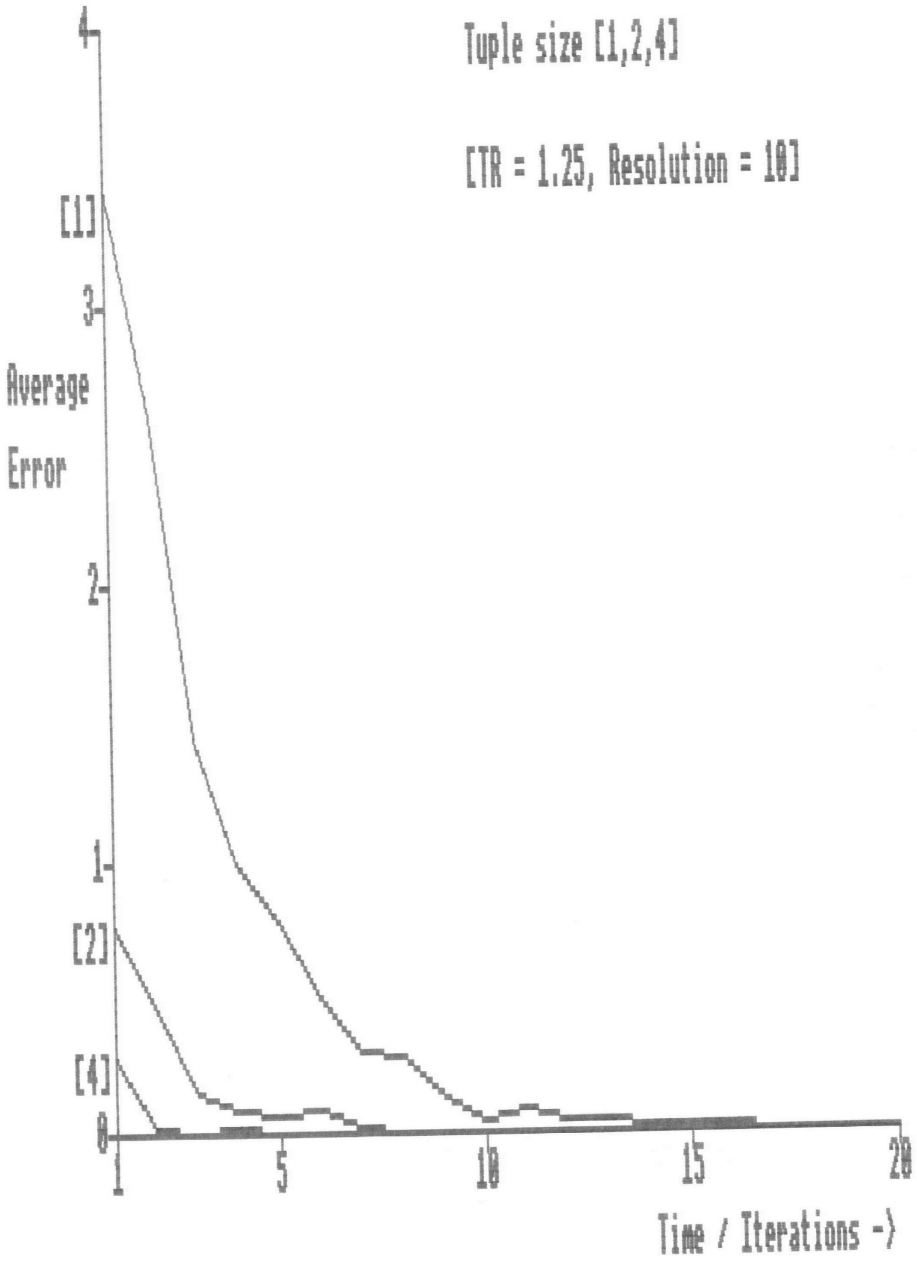


Figure 5.22 Average error, (random model)

Extrapolating these results, figure (5.23) shows diagramatically the probability that a mapping cell will point to a uniform data model with error E, over a number of iterations. This figure illustrates how the stochastic search *homes* onto the data model in a manner analogous to the way a human perceives locating an object from within a scene. The system rapidly locates the object roughly, then gradually places it with increasing accuracy within its field of view.

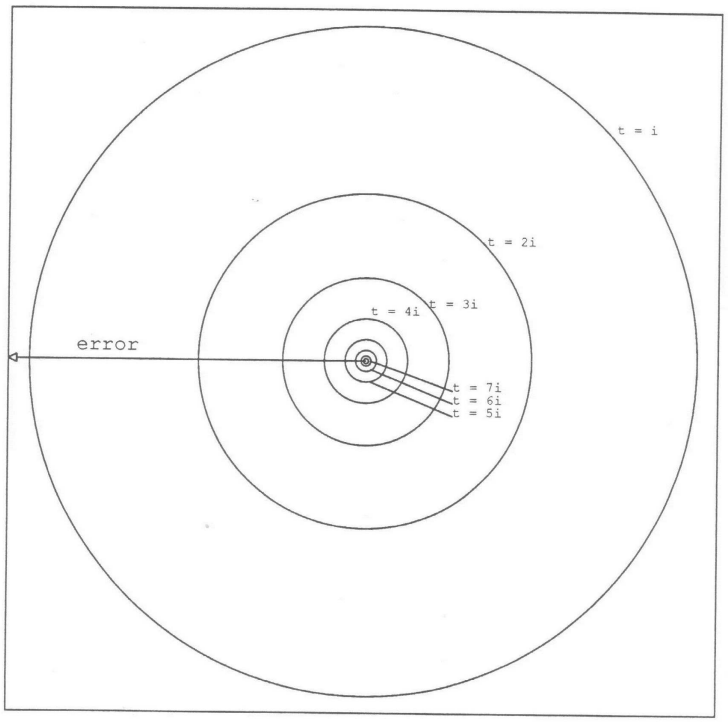


Figure 5.23 Convergence onto correct mapping, given a uniform data model

5.7 Conclusions from Experiments

Experimental results for both (TR < 1) and (TR > 1) show that for a given TR, the algorithm is of linear time as the size of the search space is increased. Central to Stochastic Searching for (TR < 1) and (TR > 1) is diffusion behaviour. Diffusion time is shown is shown to be of order O (ln (N)), where [N] is the size of the model. The expected value model of Diffusion behaviour is shown to be of the same order as experimental results, but does not accurately describe them. This is because at each iteration there is a distribution of 'correct' mapping cells, where the expected model only gives the most likely value. Due to the non linear nature of Diffusion, the error from the expected value, has a significant effect, with the result that the expected value model underestimates diffusion time.

Although to date, an accurate mathematical model of Diffusion behaviour has not been found, the experimental results show how the algorithm behaves as the probability of an erroneous test comparison rises. ADU noise, resolution, similarity and data uniformity all affect the probability of an erroneous test comparison and as this probability rises, the performance of the algorithm falls.

5.8 Stochastic Search Networks, (SSNs)

A Stochastic Search Net uses a multi-dimensional array of mapping cells, which are linked together via communication channels. Considering the two dimensional SSN illustrated in figure (5.24). Each cell is connected to each other by four channels.

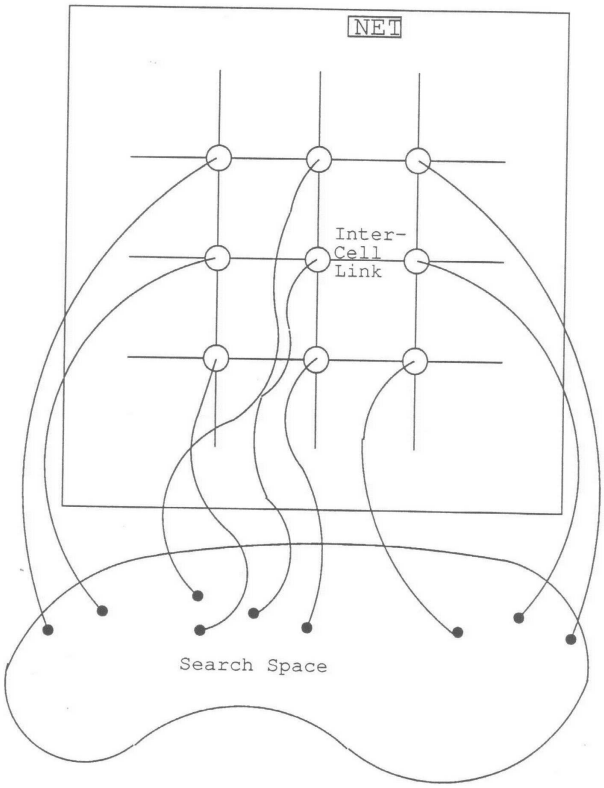


Figure 5.24 Two dimensional stochastic search network

The diffusion process now becomes:

```
PROCEDURE DIFFUSE(VAR MC : Mapping_Cells);
BEGIN
  FOR x := 1 TO nX DO
    FOR y := 1 TO nY DO
      WITH MC [x, y] DO
        IF (cFire = FALSE) THEN BEGIN
          p := RANDOM (number_of_channels);

          CASE p OF
            above : px := x; py := y + 1;
            below : px := x; py := y - 1;
            left  : px := x - 1; py := y;
            right : px := x + 1; py := y;
          END;

          IF MC [px, py].cFire = TRUE) THEN
            map := MC [px, py].map
          ELSE
            map := RANDOM (M);
        END
END (DIFFUSE};
```

The use of channels also allows the diffusion process to be carried out locally - each cell when it chooses a new mapping only needs access to its direct neighbours, rather than access to the whole array of mapping cells as in earlier stochastic diffusion. This would be a significant advantage in any possible

hardware implementation of the search technique, since each cell would only have to examine its neighbours mapping or select a new one (at random or by some other scheme).

5.8.1 Diffusion Gain

It has been described how the performance of a stochastic search is determined by fundamentally two factors; the probability of an erroneous test comparison and the Time Ratio. Another factor, inherent in the algorithm, which affects performance is **diffusion gain**. This is the rate at which potentially successful mappings, (those which yield positive test comparisons), are spread to other mapping cells. This gain is not constant throughout the diffusion process.

Examining the change in gain for the standard stochastic search. At time zero, one mapping cell from [N] has the correct mapping and diffusion gain is defined as:

$$Gain = \frac{cells\ TRUE\ at\ (t = 1)}{cells\ TRUE\ at\ (t = 0)}$$

$$Gain = \frac{initial\ number\ true + converts}{intial\ number\ true}$$

Using expected value:

$$converts = (number\ of\ cells\ FALSE) \times (prob.\ cell\ TRUE)$$

$$= (N - 1) \times \frac{1}{N}$$

Hence Diffusion Gain is:

$$Gain = 1 + (N - 1) \times \frac{\frac{1}{N}}{1}$$

$$= \frac{(2N - 1)}{N}$$

At time t' , when $\frac{N}{2}$) cells from N have the correct mapping, diffusion gain is:

$$converts = \frac{N}{2} \times \frac{1}{2} = \frac{N}{4}$$

$$Diffusion\ Gain = \frac{\frac{N}{2} + converts}{\frac{N}{2}}$$

$$= \frac{\frac{N}{2} + \frac{N}{4}}{\frac{N}{2}}$$

$$Diffusion\ Gain = \frac{3N}{2N} = 1.5$$

Then at time t' ' , when (N - 1) cells from [N] have the correct mapping, the Diffusion Gain is:

$$Diffusion\ Gain\ =\ \frac{(N\ -\ 1)\ +\ converts}{N\ -\ 1}$$

$$converts\ =\ 1\ \times\ \frac{N\ -\ 1}{N}$$

$$\therefore\ Diffusion\ Gain\ =\ \frac{\left((N\ -\ 1)\ +\ \frac{N-1}{N}\right)}{N\ -\ 1}$$

$$=\ \frac{N^2\ -\ N\ +\ N\ -\ 1}{N\ \times\ (N\ -\ 1)}$$

$$=\ \frac{N\ -\ 1}{N}$$

Stochastic Search Networks have a different diffusion gain, dependent on the number of inter cell communication channels (Q). At (t = 0), with one cell TRUE from [N];

converts = (*False cells connected to TRUE cell*) × (*Probability FALSE cell selects TRUE channel*)

$$converts\ =\ Q\ \times\ \frac{1}{Q}\ =\ 1$$

$$Diffusion\ Gain\ =\ \frac{1\ +\ converts}{1}\ =\ 2$$

Hence, at (t = t') with $\frac{N}{2}$) cells TRUE from [N]:

$$converts\ =\ \frac{Q}{2}\ \times\ \frac{1}{2}\ =\ \frac{Q}{4}$$

$$Diffusion\ Gain\ =\ \frac{\frac{N}{2}\ +\ \frac{Q}{4}}{\frac{N}{2}}\ =\ \frac{2N\ +\ Q}{2N}$$

A series of experiments were carried out to investigate the practical diffusion behaviour of a two dimensional Stochastic Search Net with (Q = 4). The number of mapping cells was varied between 4 and 81 (2^2 .. 9^2) and the results averaged over one hundred trials. One cell contained the correct mapping at (t = 0).

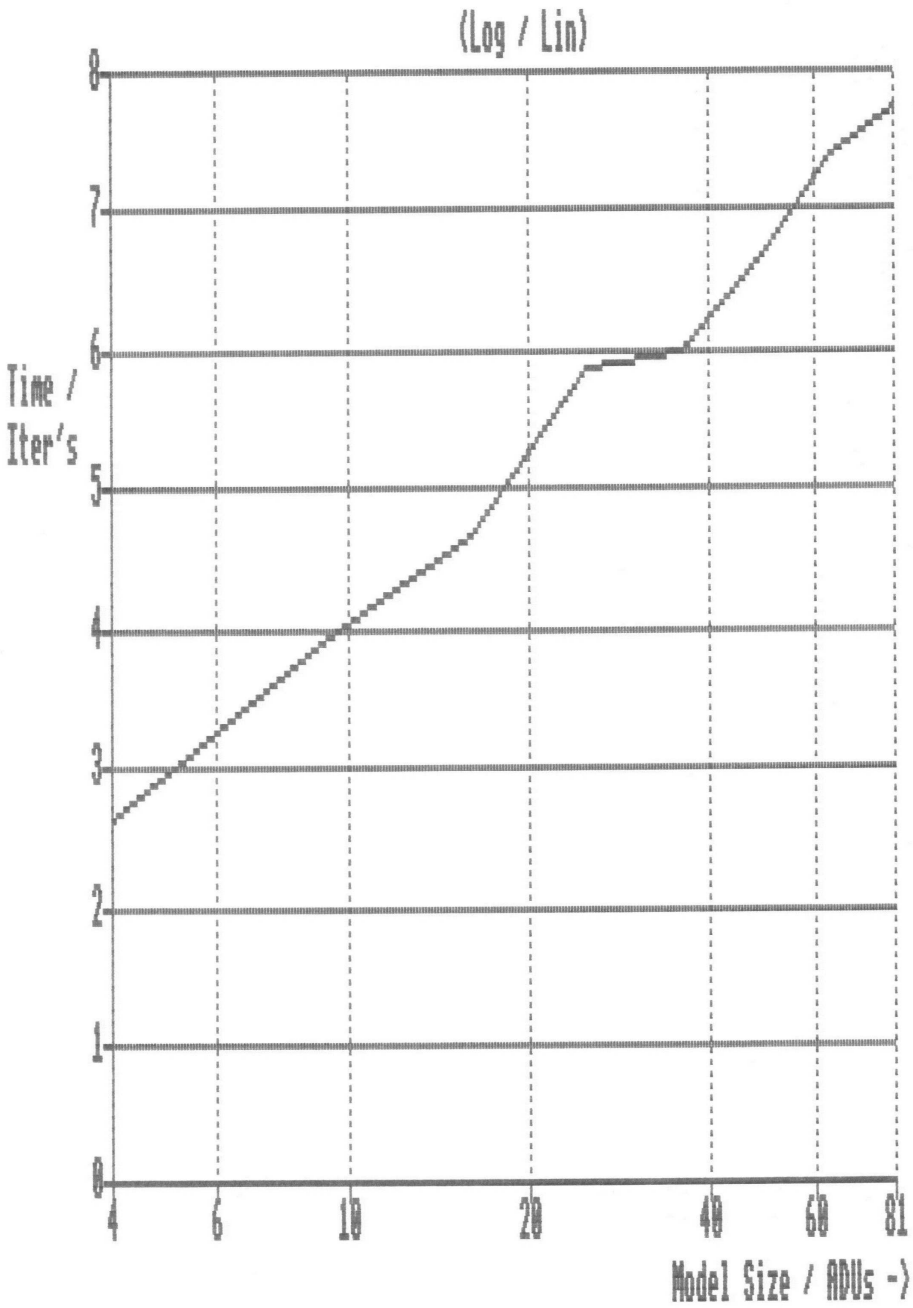


Figure 5.25 Stochastic Network diffusion time

Figure (5.25) shows the average number of iterations required to diffuse a mapping over [N] cells, where [N] was varied between [4..81] in a Stochastic Search Net with (Q=4). The graph is approximately logarithmically linear, and from it an approximation to diffusion can be obtained.

$$TD = 2.7 + 1.6 \times (ln(x) - ln(4)) \qquad (6)$$

This can be compared with the corresponding equation for diffusion using the standard search technique (5). The comparison shows that (c = 5.6, m = 1.85) for (5) and (c = 2.7, m = 1.6) for (6), hence the Stochastic Search Network, with (Q = 4) appears slightly more efficient.

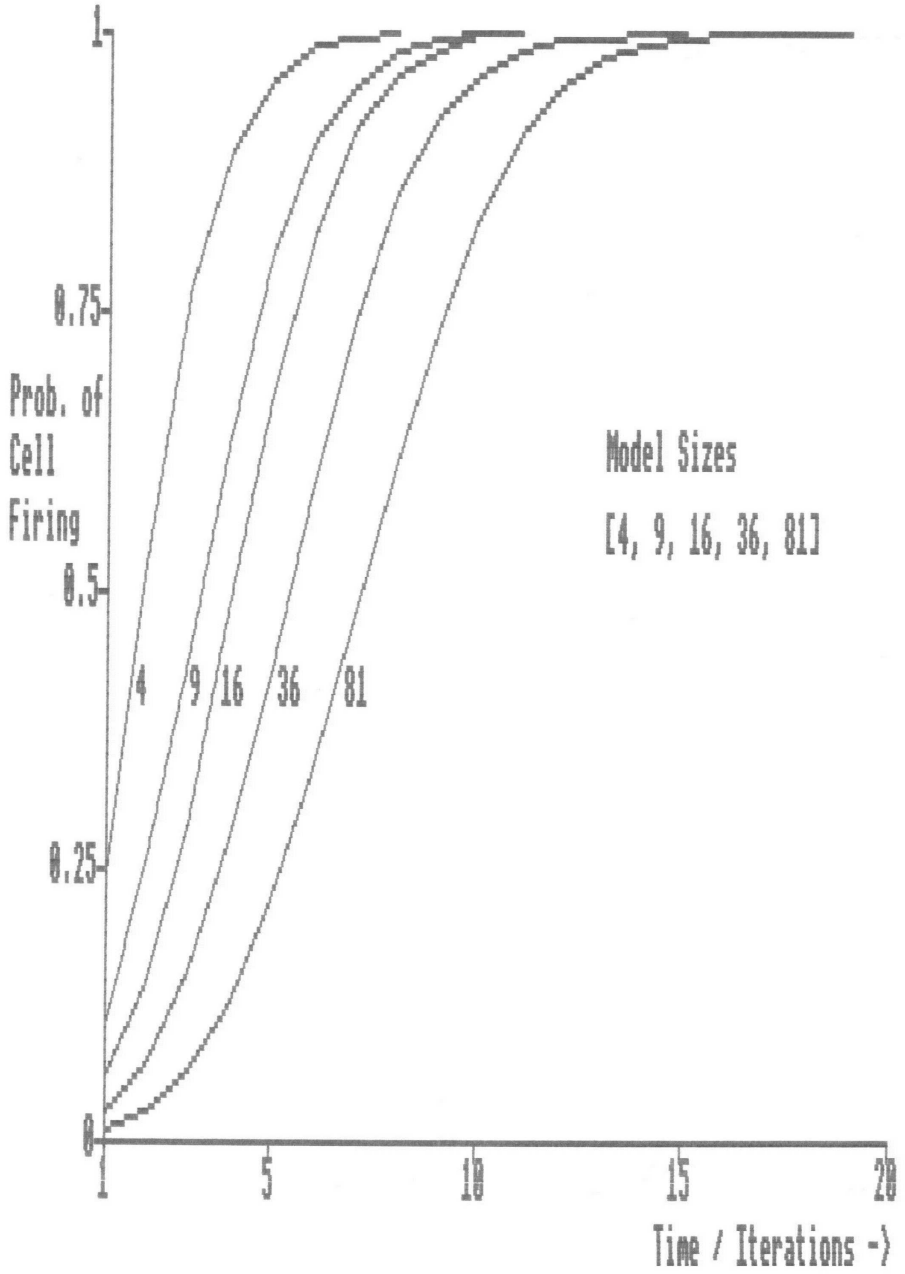


Figure 5.26 Performance of Stochastic Net

Figure (5.26) shows the comparative behaviour of a set of SSNs over time for a range of mapping cells [4, 9, 16, 36, 81].

5.9 Conclusions

The problem of pattern stimulus equivalence has traditionally proved difficult. In model based pattern classification systems it has often been tackled by some form of sequential search, which is bounded by $O(t^2)$, where t is the *Time Ratio,* ($\frac{N}{M}$), (*where N is the size of model that is being searched for and M is the size of the search space*). Fourier analysis is shown to be a useful tool in some problems of translational stimulus equivalence.

Where the problem of stimulus equivalence has been specifically addressed in Anarchic systems,

techniques such as Knowledge Replication and Hinton Mapping have been used. In a single layer n-tuple network, the model can simply be learnt in a variety of positions (a form of knowledge replication). These techniques are of varying efficiency and carry a storage/cost penalty in the case of knowledge replication. The Stochastic Search is important as it provides an efficient mechanism for the solution of 'best fit' problems on both conventional sequential computer architectures, where the algorithm is probabilistically linearly bounded by the Time Ratio of the search, and on more esoteric parallel computing devices, where the algorithm is probabilistically bounded by $O(ln(TR))$.

The performance of stochastic searching is shown to be directly influenced by two main factors, the Time Ratio of the search and the probability of a misleading elemental comparison being made in the search. The time ratio determines the number of iterations that are required before one mapping cell locates the model from the search space, (the Time to first hit - TH); the smaller TR, the longer this time will be. The probability of a misleading comparison determines how many iterations are required to diffuse the correct mapping across to all other mapping cells, (the time for diffusion - TD).

The algorithm is analysed analytically where possible, and a complete description of TH is derived. Stochastic Diffusion has proved more resistant to mathematical analysis and to date this has not been completely achieved. A mathematical approximation, using expected value analysis, is established, but although this results in the same order of function as stochastic diffusion, it is shown to be an inaccurate representation.

To obtain a more accurate description of stochastic diffusion, a recursive computational description is made. This is complete and accurate, but arithmetic limitation due to its use of the factorial function and computational limitations due to the recursion, prevent it from being widely used to define defusion behaviour beyond a model size of (N = 13).

A practical method to estimate diffusion performance, is made from experimental analysis. A description of (TD) is obtained from regression analysis of graphical data (5).

Stochastic searching is experimentally shown to be probabilistically of linear execution time when implemented on a sequential computer system.

Performance degradation factors are investigated, including the effects of noise, resolution and similarity. Noise is shown to degrade performance approximately exponentially, increased similarity causes a sharp increase in diffusion time, whereas increased data resolution is shown to reduce diffusion time rapidly. These factors all relate to fundamental algorithm behaviour as they increase or reduce the probability of making an erroneous test comparison.

Unlike many other techniques when searching for a given model, Stochastic Search performance is influenced by the structure of the data model. The worst case is uniform data which has zero internal structure. Analysis of average mapping cell error in this situation shows how the system rapidly converges to a value near to the correct mapping, then more slowly homes in onto the correct mapping. The speed of this convergence is shown to improve by comparing more than one ADU at a time in the test phase of the algorithm. The improvement is due to a reduction in the probability of making an incorrect comparison; the improvement is exponential with the number of points used, and this technique can be used to reduce the performance degrading effects mentioned above.

Finally, stochastic search networks are described, which experimental results suggest are slightly more

efficient than the original search algorithm and also promise to be a little easier to implement in hardware, as intercell communication is local. The concept of Diffusion Gain is introduced, which is another influence on algorithm performance. Diffusion gain is shown to vary over the duration of the search, and in Stochastic Search Networks is shown to be influenced by the intercell communication factor [Q].

Research into Stochastic Searching is ongoing, specifically with the aim of linking stochastic search nets to n-tuple networks. It is hoped that such a composite system will exhibit a degree of model generalisation, while preserving the speed of the basic stochastic algorithm when applied to problems of stimulus equivalence.

References

Fukushima, K., (1988), *Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition*, Neural Networks: **1**, pp.119-130. Pergamon Press USA.

Hinton, G., (1981), *A Parallel Computation that assigns Canonical Object-Based Frames of Reference*, Proceedings of the 7th International Joint Conference on Artificial Intelligence: **2**, pp. 683-685, Vancover, Canada.

Rhumelhart, D.E, McClelland, J.E., (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume one: Foundations*, pp. 113-118, The MIT Press, Cambridge MA.

Usher, M.J., (1984), *Information Theory for Information Technologists*, pp. 91-126, Macmillan, London.