Stochastic and Deterministic Networks for Texture Segmentation*

B.S.Manjunath, T. Simchony and R.Chellappa

University of Southern California

Department of EE-Systems

324, Powell Hall of Engineering

Los Angeles, California 90089

Telephone No.: (213)-743-8559.

September 14, 1988

*Partially supported by the AFSOR grant no. 86-0196.

Stochastic and Deterministic Networks for Texture Segmentation

B.S. Manjunath, T. Simchony and R. Chellappa

Abstract

This paper describes several texture segmentation algorithms based on deterministic and stochastic relaxation principles. We are mainly interested in developing algorithms which can be implemented on highly parallel networks. The segmentation process is posed as an optimization problem and two different optimality criteria are considered. The first criterion involves maximizing the posterior distribution of the intensity array given the label array (Maximum aposteriori (MAP) estimate). The posterior distribution of the texture labels is derived by modelling the textures as Gauss Markov Random Fields (GMRF) and characterizing the distribution of different texture labels by an Ising model. Fast approximate solutions for MAP are obtained using deterministic relaxation techniques implemented on a standard Hopfield type neural network. For comparison purposes simulated annealing is used to obtain the global optimum of the MAP estimate. A stochastic algorithm is then proposed which introduces learning into the iterations of the Hopfield network. This iterated hill climbing algorithm combines the fast convergence of deterministic relaxation with the sustained exploration of the stochastic algorithms. However unlike simulated annealing, this is guaranteed to find only a local minimum. The second optimality criterion requires minimizing the expected percentage of misclassification per pixel by maximizing the posterior marginal distribution. We use the Maximum Posterior Marginal (MPM) algorithm to obtain the corresponding solution. All these methods implemented on parallel networks can be easily extended for hierarchical segmentation and we present results of the various schemes in classifying some real textured images.

1 Introduction

This paper describes several algorithms, both deterministic and stochastic, for the segmentation of textured images. Segmentation of image data is an important problem in computer vision, remote sensing and image analysis. Most of the real world objects consist of textured surfaces. Segmentation based on texture information is possible even if there are no apparent intensity edges between the different regions. There are many existing methods for texture segmentation and classification, based on different kinds of statistics that can be obtained from the gray level images. The approach we use stems from the idea of using Markov random field models for textures in an image. We assign two random variables for the observed pixel, one characterizing the underlying intensity and the other for labelling the texture corresponding to the pixel location. We use the Gauss Markov Random Field (GMRF) model for the conditional density of the intensity array given the label array. Smoothness constraints are then imposed by assuming first or second order Ising distribution for the texture label configuration. The segmentation can then be formulated as an optimization problem involving minimization of a Gibbs energy function. Exhaustive search for the optimum solution is not possible because of the large dimensionality of the search space. For example, even for a very simple case of segmenting a 128×128 image into two classes, there are $2^{2^{14}}$ possible label configurations. Hence the main emphasis in this paper is in developing parallel algorithms which can be implemented on networks of simple processing elements (neurons).

There has been a growing interest in applying neural networks for solving computer vision problems. The inherent parallelism of neural networks provide an interesting architecture for implementing many computer vision algorithms. Few examples include image restoration [1], stereopsis [2] and computing optical flow [3]. Networks for solving combinatorially hard problems like the Travelling Salesman problem have received much attention in the neural network literature [4]. In almost all the cases these networks are designed to minimize certain energy function defined by the network architecture. The parameters of

the network are obtained in terms of the energy (cost) function and it can be shown that [4] for networks having symmetric interconnections, the equilibrium states correspond to the local minima of the energy function. For practical purposes, networks with few interconnections are preferred because of the large number of processing units required in any image processing application. In this context Markov Random Field (MRF) models for images play an useful role. They are typically characterized by local dependencies and symmetric interconnections which can be expressed in terms of energy functions using Gibbs-Markov equivalence [5].

We look into two different optimality criteria for segmenting the image. The first corresponds to the label configuration which maximizes the posterior probability of the label array given the intensity array. As noted before an exhaustive search for the optimal solution is practically impossible. An alternative is to use stochastic relaxation algorithms like simulated annealing [5], which asymptotically converge to the optimal solution. However the computational burden involved because of the theoretical requirements on the initial temperature and the impractical cooling schedules required overweighs their advantages in many cases. Fast approximate solutions can be obtained by deterministic relaxation algorithms like the Iterated Conditional Mode rule [6]. The energy function corresponding to this optimality criterion can be mapped into a Hopfield type network in a straightforward manner and it can be shown that the network coverges to an equilibrium state, which in general will be a local optimum. The solutions obtained using this method are sensitive to the initial configuration and in many cases starting with a Maximum Liklihood estimate is preferred. Stochastic learning can be easily introduced in to the above network and the overall system improves the performance by learning while searching. The learning algorithms used are derived from the theory of stochastic learning automata and we believe that this is the first time such a hybrid system has been used in an optimization problem. The stochastic nature of the system helps in preventing the algorithm from being trapped in a local minimum and we observe that this improves the quality of the solutions obtained.

The second optimality criterion minimizes the expected percentage of error per pixel.

This is equivalent to finding the label array that maximizes the marginal posterior probability given the intensity array [7]. Since calculating the marginal posterior probability is very difficult, Marroquin [8] suggested the MPM algorithm that asymptotically computes the posterior marginals. In [8] the algorithm is used for image restoration, stereo matching and surface interpolation. Here we use this method to find the texture label that maximizes the marginal posterior probability for each pixel.

The organization of the paper is as follows: Section 2 describes the image model. A neural network model for the relaxation algorithms is given in section 3 along with a deterministic updating rule. Section 4 gives the stochastic algorithms for segmentation and their parallel implementation on the network. A learning algorithm is proposed in section 6 and the experimental results are provided in section 7.

2 Image Model

We use a fourth order GMRF to represent the conditional probability density of the image intensity array given its texture labels. The texture labels are assumed to obey a first or second order Ising Model with a single parameter β, which measures the amount of cluster between adjacent pixels.

Let Ω denote the set of grid points in the $M \times M$ lattice, i. e. , $\Omega = \{(i,j)\ ,\ 1 \leq i,j \leq M\}$. Following Geman and Graffigne [9] we construct a composite model which accounts for texture labels and gray levels. Let $\{L_s\ ,\ s \in \Omega\}$ and $\{Y_s\ ,\ s \in \Omega\ \}$ denote the labels and zero mean gray level arrays respectively. The zero mean array is obtained by subtracting the local mean computed in a small window centered at each pixel. Let N_s denote the symmetric fourth order neighborhood of a site s. Then we can write the following expression for the conditional density of the intensity at the pixel site s:

$$P(Y_s = y_s \mid Y_r = y_r, r \in N_s, L_s = l) = \frac{e^{-U(Y_s=y_s \mid Y_r=y_r, r\in N_s, L_s=l)}}{Z(l)}$$

where $Z(l)$ is the partition function of the conditional Gibbs distribution and

$$U(Y_s = y_s \mid Y_r = y_r, r \in N_s, L_s = l) = \frac{1}{2\sigma_l^2}(y_s^2 - 2\sum_{r \in N_s} \Theta^l_{s-r} y_s y_r) \tag{1}$$

In (1), σ_l and Θ^l are the GMRF model parameters of the l-th texture class. The model parameters satisfy $\Theta^l_{r,s} = \Theta^l_{r-s} = \Theta^l_{s-r} = \Theta^l_r$.

The Gibbs energy function computed in (1) should be used in the classification process. However seldom do the texture features be so small as to be captured by a fourth order neighborhood. Increasing the order of the GMRF model requires the estimation of additional model parameters which are quite sensitive. An alternative approach is to calculate the joint distribution of the intensity conditioned on the texture label in a small window centered at the pixel site. The corresponding Gibbs energy can then be used in the relaxtion process for segmentation. We view the image intensity array as composed of a set of overlapping $k \times k$ windows W_s, centered at each pixel $s \in \Omega$. In each of these windows we assume that the texture label L_s is homogeneous (all the pixels in the window belong to the same texture). As before we model the intensity in the window by a fourth order stationary GMRF. The local mean is computed by taking the average of the intensities in the window W_s and is subtracted from the original image to get the zero mean image. All our references to the intensity array corresponds to the zero mean image. Let $\mathbf{Y}^*_s$ denote the 2-D vector representing the intensity array in the window W_s. Using the Gibbs formulation and assuming a free boundary model, the joint probability density in the window W_s can be written as,

$$P(\mathbf{Y}^*_s | L_s = l) = \frac{e^{-U_1(\mathbf{Y}^*_s | L_s = l)}}{Z_1(l)}$$

where $Z_1(l)$ is the partition function and

$$U_1(\mathbf{Y}^*_s | L_s = l) = \frac{1}{2k^2\,\sigma_l^2} \sum_{r \in W_s} \left\{ y_r^2 - 2 \sum_{\tau \in N^* | r+\tau \in W_s} \Theta^l_\tau y_r y_{r+\tau} \right\} \tag{2}$$

N^* is the set of shift vectors corresponding to a fourth order neighborhood system:

$$
\begin{aligned}
N^* &= \{\tau_1, \tau_2, \tau_3, \cdots, \tau_{10}\} \\
&= \{(0,1),(1,0),(1,1),(-1,1),(0,2),(2,0),(1,2),(2,1),(-1,2),(-2,1)\}
\end{aligned}
$$

The label array is modeled as a first or second order Ising distribution. If $\hat{N}_s$ denotes the appropriate neighborhood for the Ising model, then we can write the distribution function for the texture label at site s conditioned on the labels of the neighboring sites as:

$$P(L_s|L_r\ ,\ r \in \hat{N}_s) = \frac{e^{-U_2(L_s\ |\ L_r)}}{Z_2}$$

where Z_2 is a normalizing constant and

$$U_2(L_s \mid L_r,\ r \in \hat{N}_s) = -\beta \sum_{r \in \hat{N}_s} \delta(L_s - L_r)\ ,\ \beta > 0 \tag{3}$$

In (3), β determines the degree of clustering, and $\delta(i-j)$ is the Kronecker delta. Using the Bayes rule, we can write

$$P(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = \frac{P(\mathbf{Y}_s^* \mid L_s)\ P(L_s \mid L_r,\ r \in \hat{N}_s)}{P(\mathbf{Y}_s^*)} \tag{4}$$

Since $\mathbf{Y}_s^*$ is known, the denominator in (4) is just a normalizing factor. The numerator is a product of two exponential functions and can be expressed as,

$$P(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = \frac{1}{Z_p}\ e^{-U_p(L_s\ |\ \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s)} \tag{5}$$

where Z_p is a normalizing factor and $U_p(.)$ is the posterior energy corresponding to (5). From (1) and (2) we write

$$U_p(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = w(L_s) + U_1(\mathbf{Y}_s^* \mid L_s) + U_2(L_s \mid L_r,\ r \in \hat{N}_s) \tag{6}$$

Note that the second term in (6) relates the observed pixel intensities to the texture labels and the last term specifies the label distribution. The bias term $w(L_s) = \log Z_1(L_s)$ is dependent on the texture class and it can be explicitly evaluated for the GMRF model considered here using the toroidal assumption. However the computations become very

combersome if toroidal assumptions are not made. An alternate approach is to estimate the bias from the histogram of the data as suggested by Geman and Graffigne [9]. Finally, the posterior distribution of the texture labels for the entire image given the intensity array is

$$P(\mathbf{L} \mid \mathbf{Y}^*) = \frac{P(\mathbf{Y}^* \mid \mathbf{L})\, P(\mathbf{L})}{P(\mathbf{Y}^*)} \tag{7}$$

Maximizing (7) gives the optimal Bayesian estimate. Though it is possible in principle to compute the righthand side of (7) and find the global optimum, the computational burden involved is so enormous that it is practically impossible to do so. However we note that the stochastic relaxation algorithms discussed in the section 4 require only the computation of (5) to obtain the optimal solution. The deterministic relaxation algorithm given in the next section also uses these values, but is guaranteed to find only a local minimum.

3 A Neural Network for Texture Classification

In this section we describe the network architecture used for segmentation and the implementation of deterministic relaxation algorithms. The energy function which the network minimizes is obtained from the image model discussed in the previous section. For convenience of notation let $U_1(i,j,l) = U_1(\mathbf{Y}_s^*, L_s = l) + w(l)$ where $s = (i,j)$ denotes a pixel site and $U_1(\,.\,)$ and $w(l)$ are as defined in (6). The network consists of K layers, each layer arranged as an $M \times M$ array, where K is the number of texture classes in the image and M is the dimension of the image. The elements (neurons) in the network are assumed to be binary and are indexed by (i,j,l) where $(i,j) = s$ refers to their position in the image and l refers to the layer. The (i,j,l)-th neuron is said to be ON if its output V_{ijl} is 1, indicating that the corresponding site $s = (i,j)$ in the image has the texture label l. Let $T_{ijl;i'j'l'}$ be the connection strength between the neurons (i,j,l) and (i',j',l') and I_{ijl} be the input bias current. Then a general form for the energy of the network is [4]

$$E = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K}\sum_{i'=1}^{M}\sum_{j'=1}^{M}\sum_{l'=1}^{K} T_{ijl;i'j'l'}V_{ijl}V_{i'j'l'} - \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K} I_{ijl}V_{ijl} \tag{8}$$

From our discussion in section 2 we note that a solution for the MAP estimate can be obtained by minimizing (7). It is easy to see that this is equivalent to minimizing the following energy function for the network:

$$E = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K} U_1(i,j,l)V_{ijl} - \frac{\beta}{2}\sum_{l=1}^{K}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{(i',j')\in\hat{N}_{ij}} V_{i'j'l}V_{ijl} \tag{9}$$

where $\hat{N}_{ij}$ is the neighborhood of site (i,j) (same as the $\hat{N}_s$ in section 2). In (9), it is implicitly assumed that each pixel site has a unique label, i.e. only one neuron is active in each column of the network. This constraint can be implemented in different ways. For the deterministic relaxation algorithm described below, a simple method is to use a *winner-takes-all* circuit for each column so that the neuron receiving the maximum input is turned on and the others are turned off. Alternately a penalty term can be introduced in (9) to represent the constraint as in [4]. From (8) and (9) we can identify the parameters for the network,

$$T_{ijl;i'j'l'} = \begin{cases} \beta & \text{if } (i',j') \in \hat{N}_{ij}, \forall\, l \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

and the bias current

$$I_{ijl} = -\,U_1(i,j,l) \tag{11}$$

3.1 Deterministic Relaxation

The above equations (10) and (11) relate the parameters of the network in terms of the image model. The connection matrix for the above network is symmetric and there is no self feedback, i.e. $T_{ijl;ijl} = 0, \forall i,j,l$. Let u_{ijl} be the potential of neuron (i,j,l). (Note:l is

the layer number corresponding to texture class l) , then

$$u_{ijl} = \sum_{i'=1}^{M} \sum_{j'=1}^{M} \sum_{l'=1}^{K} T_{ijl;i'j'l'} \; V_{i'j'l'} + I_{ijl} \tag{12}$$

In order to minimize (9), we use the following updating rule:

$$V_{ijl} = \begin{cases} 1 & \text{if } u_{ijl} = \min_{l'}\{u_{ijl'}\} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

This updating scheme ensures that at each stage the energy decreases. Since the energy is bounded, the convergence of the above system is assured but the stable state will in general be a local optimum.

This network model is a version of the Iterated Conditional Mode algorithm (ICM) of Besag [6]. This algorithm maximizes the conditional proability $P(L_s = l|\mathbf{Y}_s^*, L_{s'}, s' \in \hat{N}_s)$ during each iteration . ICM is a local deterministic relaxation algorithm that is very easy to implement. We observe that in general any algorithm based on MRF models can be easily mapped on to neural networks with local interconnections. The main advantage of this detarministic relaxtion algorithms is its simplicity. Often the solutions are reasonably good and the algorithm usually converges within 20-30 iterations. In the next section we study two stochastic schemes which asymptotically converge to the global optimum of the respective criterion functions.

4 Stochastic Algorithms for Texture segmentation

In this section we look at two optimal solutions corresponding to different decision rules for determining the pixels labels. The first one uses the simulated annealing to obtain the optimum MAP estimate of the label configuration. The second algorithm minimizes the expected misclassification per pixel. The parallel implementation of these algorithms on the network is discussed in section 4.3.

4.1 Searching for MAP Solution

The MAP rule, suggested in [9] searches for the configuration L that maximizes the posterior probability distribution. This is equivalent to maximizing $P(\mathbf{Y}^* \mid \mathbf{L})\ P(\mathbf{L})$ as $P(\mathbf{Y}^*)$ is independent of the labels and $\mathbf{Y}^*$ is known. The right hand side of (7) is a Gibbs Distribution. To maximize (7) we use simulated annealing [5], a combinatorial optimization method which is based on sampling from varying Gibbs distribution functions:

$$\frac{e^{-\frac{1}{T_k}U_p(L_s \mid \mathbf{Y}^*_s,\ L_r, r \in \hat{N}_s)}}{Z_{T_k}}$$

In order to maximize:

$$\frac{e^{-U_p(\mathbf{L} \mid \mathbf{Y}^*)}}{Z}$$

T the varying parameter - will be referred to as the temperature. We used the following cooling schedule:

$$T_k = \frac{T_0}{1 + \log_2\ k} \tag{14}$$

where k is the iteration number. When the temperature is high, the bond between adjacent pixels is loose, and the distribution tends to behave like a uniform distribution over the possible texture labels. As T_k decreases, the distribution concentrates on the lower values of the energy function - equivalent to points with higher probability. The process is bound to converge to a uniform distribution over the label configuration that corresponds to the MAP solution. Since number of texture labels is finite, convergence of this algorithm follows from [5]. In our experiment, we realized that starting the iterations with $T_0 = 2$ did not guarantee convergence to the MAP solution. Since starting at a much higher temperature will slow the convergence of the algorithm significantly, we use an alternative approach, viz., cycling the temperature [7]. We follow the annealing schedule till T_k reaches a lower bound then we reheat the system and start a new cooling process. By using only a few cycles, we obtained results better than those with a single cooling cycle. Parallel implementation of simulated annealing on the hopfield network is discussed in section 4.3. The results we present in section 6 were obtained with two cycles.

4.2 Maximizing the Posterior Marginal Distribution

The choice of the objective function for optimal segmentation, can significantly affect it's result. The choice should be made depending on the purpose of the classification. In many implementations the most reasonable objective function is the one that minimizes the expected percentage misclassification per pixel. The solution to the above objective function is also the one that maximizes the marginal posterior distribution of L_s given the observation $\mathbf{Y}^*$, for each pixel s.

$$P\{L_s = l_s \mid \mathbf{Y}^* = \mathbf{y}^*\} \propto \sum_{\mathbf{l}|L_s=l_s} P(\mathbf{Y}^* = \mathbf{y}^* \mid \mathbf{L} = \mathbf{l})\ P(\mathbf{L} = \mathbf{l})$$

The summation above extends over all possible label configurations keeping the label at site s constant. This concept was thoroughly investigated in [8]. Marroquin [8] discusses this formulation in the context of image restoration, and illustrates the performance on images with few gray levels. He also mentions the possibility of using this objective function for texture segmentation. In [6] the same objective function is mentioned in the context of image estimation.

To find the optimal solution we use the stochastic algorithm suggested in [8]. The algorithm samples out of the posterior distribution of the texture labels given the intensity. Unlike the stochastic relaxation algorithm, samples are taken with a fixed temperature $T = 1$. The Markov chain associated with the sampling algorithm converges with probability one to the posterior distribution. We define new random variables $g_s^{l,t}$ for each pixel ($s \in \Omega$):

$$g_s^{l,t}\{L_s^t\} = \begin{cases} 1 & L_s^t = l \\ 0 & \text{otherwise} \end{cases}$$

Where L_s^t is the class of the s pixel, at time t, in the state vector of the Markov chain associated with the Gibbs sampler. We use the ergodic property of the Markov chain [10] to calculate the expectations for these random variables using time averaging:

$$E\{g_s^{l,t}\} = \lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} g_s^{l,t} = P_s\{L_s = l|Y^*\}$$

where N is the number of iterations performed. To obtain the optimal class for each pixel, we simply chose the class that occurred more often than the others.

The MPM algorithm was implemented using the Gibbs sampler [5]. A much wider set of sampling algorithms such as Metropolis can be used for this purpose. The algorithms can be implemented sequentially or in parallel, with a deterministic or stochastic decision rule for the order of visiting the pixels. In order to avoid the dependence on the initial state of the Markov chain, we can ignore the first few iterations. In the experiments conducted we obtained good results after five hundred iterations. The algorithm does not suffer from the drawbacks of simulated annealing. For instance we do not have to start the iterations with a high temperature to avoid local minima and the performance is not badly affected by enlarging the state space. One can create a decision rule that synthesizes the MAP and MPM decision rules, by sampling at a fixed temperature smaller than one (run the MPM algorithm with $0 < T < 1$). At $T < 1$ the distribution picks at configurations that correspond to a higher *aposteriori* probability. In the limit, if we could sample from the distribution at $T = 0$ the MPM rule would have been the same as MAP. But at T=0, the Markov chain produced by the Gibbs sampler may have more than one ergodic set, because different local minima of the Gibbs energy function correspond to different ergodic sets. Yet, for any $0 < T$ the Markov chain corresponding to the Gibbs sampler contains only one ergodic set, and we can use the ergodic property of the Markov chain to obtain the desired estimate that have properties intermediate between MPM and MAP.

4.3 Network Implementation of the Sampling Algorithms

All the stochastic algorithms described in the Gibbs formulation are based on sampling from a probability distribution. The probability distribution is constant in the MPM algorithm [8] and is time varying in the case of annealing. The need for parallel implementation is due to the heavy computational load associated with their use.

The issue of parallel implementation in stochastic algorithms was first addressed in [5].

They show that the Gibbs sampler can be implemented in any deterministic or stochastic rule for choosing the order in which pixels are updated, as long as each pixel is visited infinitely often. They define an iteration as the time required to visit each pixel at least once (a full sweep). Note that the stochastic rules have a random period and allow us to visit a pixel more than once in a period. They consider the new Markov chain one obtains from the original by viewing it only after each full sweep is performed. Their proof is based on two essential elements. The first is the fact that the embedded Markov chain has a strictly positive transition probability p_{ij} for any possible states i, j, which proves that the chain will converge to a unique probability measure regardless of the initial state. The second is that the Gibbs measure is an invariant measure for the Gibbs sampler, so that the embedded chain converges to the Gibbs measure. The proof introduced in [5] can be applied to a much larger family of sampling algorithm satisfying the following properties:
1:The sampler produces a Markov chain with a positive transition probability p_{ij} for any choice of states i, j.
2:The Gibbs measure is invariant under the sampling algorithm.
The Metropolis and heat bath algorithms are two such sampling methods. To see that the Metropolis algorithm satisfy property 2, we look at the following equation for updating a single pixel:

$$P^{n+1}(i) = \frac{1}{m} \sum_{\pi(j)<\pi(i)} P^n(j) + \frac{1}{m} \sum_{\pi(j)<\pi(i)} P^n(i) \frac{\pi(i) - \pi(j)}{\pi(i)} + \frac{1}{m} \sum_{\pi(j)\geq\pi(i)} P^n(j) \frac{\pi(i)}{\pi(j)}$$

where m is the number of values each pixel can take. The first term corresponds to the cases when the system was in state j and the new state i has higher probability. The second term corresponds to a system in state i and a new state j that has lower probability. The given probability is for staying in state i. The third term corresponds to a system in state j and a new state i with lower probability. If we now replace $P^{n+1}(i)$ and $P^n(i)$ by $\pi(i)$ and $P^n(j)$ by $\pi(j)$ we see that the equality holds, implying that the Gibbs measure is invariant under the Metropolis algorithm. The first property is also satisfied. Note that the states now correspond to the global configuration. To implement the algorithm in parallel, one

can update pixels in parallel as long as neighboring pixels are not updated at the same time. A very clear discussion on this issue can be found in [8].

We now describe how these stochastic algorithms can be implemented on the network discussed in section 3. The only modification required for the simulated annealing rule is that the neurons in the network fire according to a time dependent probabilistic rule. Using the same notation as in section 3, the probability that neuron (i, j, l) will fire during iteration k is,

$$prob\ (V_{ijl} = 1) = \frac{e^{\frac{1}{T_k} u_{ijl}}}{Z_{T_k}} \tag{15}$$

where u_{ijl} is as defined in (12) and T_k follows the cooling schedule (14).

The MPM algorithm uses the above selection rule with $T_k = 1$. In addition, each neuron in the network has a counter which is incremented everytime the neuron fires. When the iterations are terminated the neuron in each column of the network having the maximum count is selected to represent the label for the corresponding pixel site in the image.

We have noted before that for parallel implementation of the sampling algorithms, neighbouring sites should not be updated simultaneously. Some additional observations are made in section 6.

5 Stochastic Learning and Neural networks

In the previous sections purely deterministic and stochastic relaxation algorithms were discussed. Each has its own advantages and disadvantages. Here we consider the possibility of combining the two methods using stochastic learning automata and compare the results obtained using this new scheme with the previous algorithms.

We begin with a brief introduction to the Stochastic Learning Automaton (SLA). A SLA is a decision maker operating in a random environment. A stochastic automaton can be defined by a quadruple (α, Q, T, R) where $\alpha = \{\alpha_1, \ldots, \alpha_N\}$ is the set of available actions to the automaton. The action selected at time t is denoted by $\alpha(t)$. $Q(t)$ is the

state of the automaton at time t and consists of the action probability vector $\mathbf{p}(t) = [p_1(t), \ldots, p_N(t)]$ where $p_i(t) = \text{prob}\ (\alpha(t) = \alpha_i)$ and $\sum_i p_i(t) = 1\ \forall t$. The environment responds to the action $\alpha(t)$ with a $\lambda(t) \in R$, R being the set of environment's responses. The state transitions of the automaton are governed by the learning algorithm T, $Q(t+1) = T(Q(t), \alpha(t), \lambda(t))$. Without loss of generality it can be assumed that $R = [0, 1]$, i.e., the responses are normalized to lie in the interval [0,1], '1' indicating a complete *success* and '0' total *failure*. The goal of the automaton is to converge to the optimal action, i.e. the action which results in the maximum expected reward. Again without loss of generality let α_1 be the optimal action and $d_1 = E[\lambda(t) \mid \alpha_1] = \max_i\{E[\lambda(t) \mid \alpha_i]\}$. At present no learning algorithms exist which is optimal in the above sense. However we can choose the parameters of certain learning algorithms so as to realize a response as close to the optimum as desired. This condition is called ϵ-optimality. If $M(t) \triangleq E[\lambda(t) \mid p(t)]$, then a learning algorithm is said to be ϵ-optimal if it results in a $M(t)$ such that

$$\lim_{t \to \infty} E[M(t)] > d_1 - \epsilon \tag{16}$$

for a suitable choice of parameters and for any $\epsilon > 0$. One of the simplest learning schemes is the Linear Reward-Inaction rule , L_{R-I} . Suppose at time t we have $\alpha(t) = \alpha_i$ and if $\lambda(t)$ is the response received then according to the L_{R-I} rule,

$$\begin{aligned} p_i(t+1) &= p_i(t) + a\ \lambda(t)\ [1 - p_i(t)] \\ p_j(t+1) &= p_j(t)[1 - a\ \lambda(t)\ p_j(t)],\ \forall j \neq i \end{aligned} \tag{17}$$

where a is a parameter of the algorithm controlling the learning rate. Typical values for a are in the range 0.01-0.1. It can be shown that this L_{R-I} rule is $\epsilon - optimal$ in all stationary environments i.e., there exists a value for the parameter a so that condition (16) is satisfied.

Collective behavior of a group of automata has also been studied. Consider a team of N automata $A_i (i = 1, .., N)$ each having r_i actions $\alpha^i = \{\alpha^i_1 \ldots \alpha^i_{r_i}\}$. At any instant

t each member of the team makes a decision $\alpha^i(t)$. The environment responds to this by sending reinforcement signal $\lambda(t)$ to all the automata in the group. This situation represents a co-operative game among a team of automata with an identical pay-off. All the automata update their action probability vectors according to (3) using the same learning rate and the process repeats. Local convergence results can be obtained in case of stationary random environments. Variations of this rule have been applied to complex problems like decentralized control of Markov Chains [11] and relaxation labelling [12]

The texture classification discussed in the previous sections can be treated as a relaxation labelling problem and stochastic automata can be used to learn the labels (texture class) for the pixels. A Learning Automaton is assigned to each of the pixel sites in the image. The actions of the automata correspond to selecting a label for the pixel site to which it is assigned. Thus each automaton has K actions and a probability distribution over this action set. Initially the labels are assigned randomly with equal probability. Since the number of automata involved is very large, it is not practicable to update the action probability vector at each iteration. Instead we combine the iterations of the neural network described in the previous section with the stochastic learning algorithm. This results in an iterative hill climbing type algorithm which combines the fast convergence of deterministic relaxation with the sustained exploration of the stochastic algorithm. The stochastic part prevents the algorithm from getting stuck in a local minima and at the same time "learns" from the search by updating the state probabilities. However unlike simulated annealing, we cannot guarantee convergence to the global optimum. Each cycle now has two phases: The first phase consists of the deterministic relaxation network converging to a solution. The second phase consists of the learning network updating its state, the new state being determined by the equilibrium state of the relaxation network. A new initial state is generated by the learning network depending on its current state and the cycle repeats. Thus relaxation and learning alternate with each other. After each iteration the probability of the more stable states increases and because of the stochastic nature of the algorithm the possibility of getting trapped in a bad local minima is reduced.

The algorithm is summarized below.

5.1 Learning Algorithm

Let the pixel site be denoted (as in section 2) by $s \in \Omega$ and the number of texture classes be L. Let A_s be the automaton assigned to site s and the action probability vector of A_s be $\mathbf{p}_s(t) = [p_{s,1}(t), \ldots, p_{s,L}(t)]$ and $\sum_i p_{s,i}(t) = 1 \forall s, t$, where $p_{s,l}(t) = \text{prob(label of site } s = l)$. The steps in the algorithm are:

1. Initialize the action probability vectors of all the automata:

$$p_{s,l}(0) = 1/K, \ \forall s, l$$

   Initialize the iteration counter to 0.

2. Choose an initial label configuration sampled from the distribution of these probability vectors.

3. Start the neural network of section 3 with this configuration.

4. Let l_s denote the label for site s at equilibrium. Let the current time (iteration number) be t. Then the action probabilities are updated as follows:

$$\begin{aligned} p_{s,l_s}(t+1) &= p_{s,l_s}(t) + a\ \lambda(t)\ [1 - p_{s,l_s}(t)] \\ p_{s,j}(t+1) &= p_{s,j}(t)[1 - a\ \lambda(t)\ p_j(t)], \ \forall\ s \quad \text{and } \forall j \neq l_s \end{aligned} \tag{18}$$

   The response $\lambda(t)$ is derived as follows: Suppose the present label configuration resulted in a lower energy state compared to the previous one then it results in a $\lambda(t) = \lambda_1$ and if the energy increases we have $\lambda(t) = \lambda_2$ with $\lambda_1 > \lambda_2$. In our simulations we have used $\lambda_1 = 1$ and $\lambda_2 = 0.25$.

5. Generate a new configuration from this updated label probabilities, Increment the iteration counter and goto step 3.

Thus the system consists of two layers, one for relaxation and the other for learning. The relaxation network is similar to the one considered in section 3, the only difference is that the initial state is decided by the learning network. The learning network consists of a team of automata and learning takes place at a much lower speed than the relaxation with fewer number of updatings. The probabilities of the labels corresponding to the final state of the relaxation network are increased according to (18). Using these new probabilities a new cofiguration is generated. Since the response does not depend on time, this corresponds to a stationary environment and as we have noted before this L_{R-I} algorithm can be shown to converge to a stationary point, not necessarily the global optimum.

6 Experimental results and conclusions

The segmentation results using the above algorithms are given on two examples. The parameters σ_l and Θ_l corresponding to the fourth order GMRF for each texture class were pre-computed from 64×64 images of the textures. The local mean (in a 11×11 window) was first subtracted to obtain the zero mean texture and the least square estimates [13] of the parameters were then computed from the interior of the image. The first step in the segmentation process involves computing the Gibbs energies $U_1(\mathbf{Y}_s|L_s)$ in (2). This is done for each texture class and the results are stored. To ignore the boundary effects, we set $U_1 = 0$ at the boundaries. We have experimented with different window sizes W_s. Larger windows result in more homogeneous texture patches but the boundaries between the textures are distorted. The results reported here are based on windows of size 11×11 pixels. The bias term $w(l_s)$ can be estimated using the histogram of the image data [9] but we obtained these values by trial and error.

In section 4 we observed that neighbouring pixel sites should not be updated simultaneously. This problem occurs only if digital implementation of the networks are considered as the probability of such a thing happening in an analog network is zero. When this simulataneous updating was tested for the deterministic case it always converged to limit

cycles of length 2 (In fact it can be shown that the system converges to limit cycles of length atmost two).

The choise of β plays an important role in the segmentation process and its value depends on the magnitude of the enegy function $U_1(.)$. Various values of β ranging from 0.2-3.0 were used in the experiments. In the deterministic algorithms it is preferable to start with a small β and increase it gradually. Large values of beta usually degrade their performance. We also observed that slowly increasing β during the iterations improves the results for the stochastic algorithms. It should be noted that using a larger value of β for the deterministic algorithms (compared to those of stochastic algorithms) does not improve the performance.

The nature of the segmentation results depends on the order of the Ising model. It is preferable to choose the first order model for the stochastic algorithms if we know apriori that the boundaries are either horizontal or vertical. However for the deterministic rule and the learning scheme, second order model results in more homogeneous classification.

The MPM algorithm requires the statistics obtained from the invariant measure of the markov chain corresponding to the sampling algorithm. Hence it is preferable to ignore the first few hundred trials before starting to gather the statistics. The performance of the deterministic relaxation rule of section 5 also depends on the initial state and we have looked into two different initial conditions. The first one starts with a label configuration L such that $L_s = l_s$ if $U_1(\mathbf{Y}_s^* | l_s) = \min_{l_k}\{U_1(\mathbf{Y}_s^* \mid l_k)\}$. This corresponds to the Maximizing the probability $P(\mathbf{Y}^* \mid \mathbf{L})$ [15] . The second choice for the initial configuration is a randomly generated label set. Results for both the cases are provided and we observe that the random choise often leads to better results.

In the examples below the following learning parameters were used: Learning rate $a = 0.05$, reward/penalty parameters $\lambda_1 = 0.25$ and $\lambda_2 = 1.0$.

Example 1: This is a two class problem consisting of the grass and calf textures. The image is of size 128 $\times$ 128 and is shown in figure 2. The segmentation results for the different algorithms are shown in fig.2.a - 2.f. In all the cases we used $\beta = 0.6$.

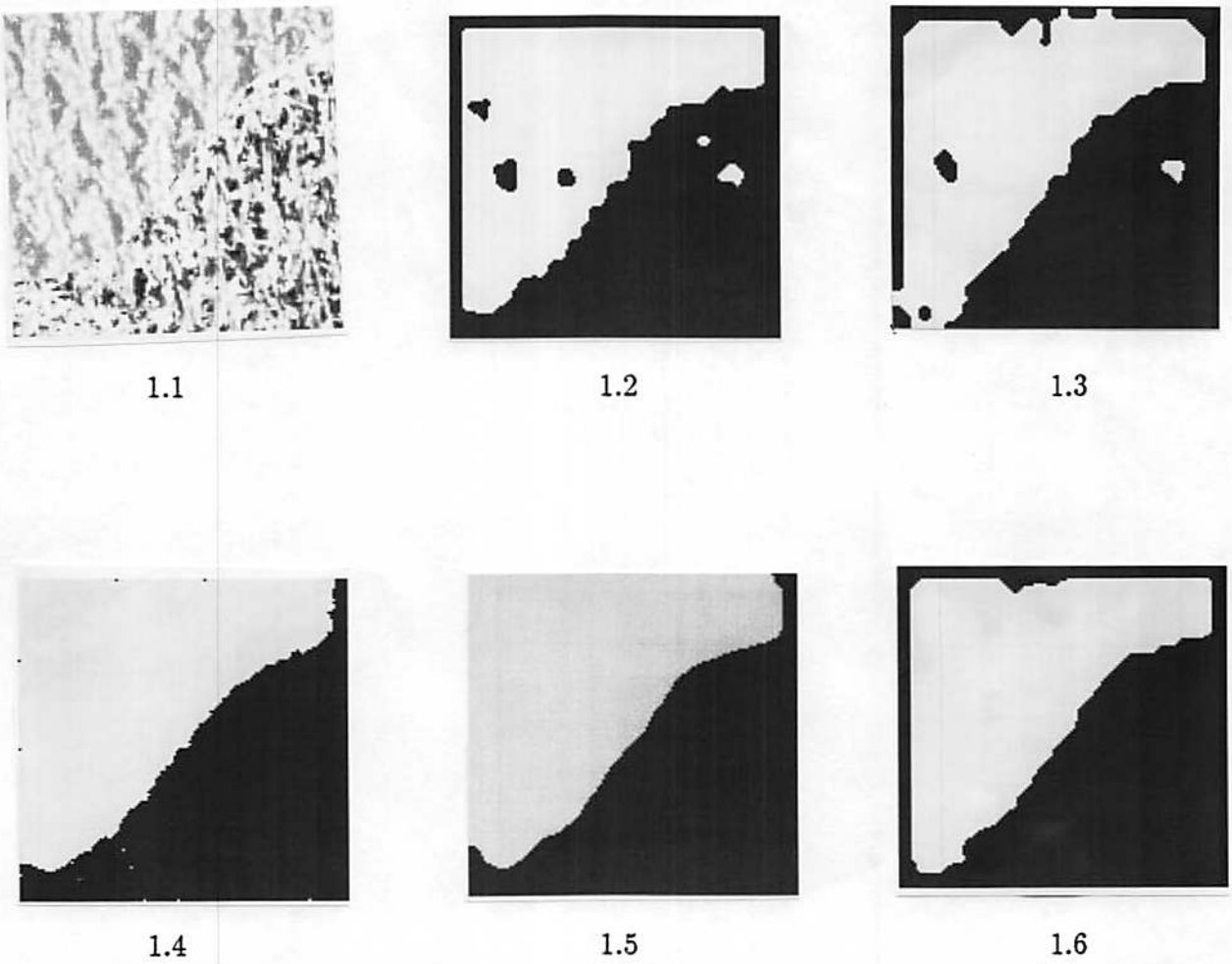
Example 2: This is a 256×256 imge having six textures: calf, grass, wool, wood, pig skin and sand. This is a difficult problem in the sense that three of the textures (wool, pig skin and sand) have almost the identical characteristics and even for a human eye it is hard to distinguish among them. As was mentioned in section 3 cycling of temperature improves the performance of the simulated annealing. The segmentation result was obtained by starting with an initial temperature $T_0 = 2.0$ and cooling according to the schedule (14) for 300 iterations. Then the system was reset to $T_0 = 1.5$ and the process was repeated for 300 more iterations. The final segmentation is shown in figure 2.5. In case of the MPM rule the first 500 iterations were ignored and figure 2.6 shows the result obtained using the last two hundred iterations. The simulation results for the various algorithms are shown in figure 2. The percentage error in classifying the different textures is summarized in table 1.

Algorithm	Percentage Error
Maximum Likelihood Estimate	22.17
Neural network (MLE as initial state)	16.25
Neural network (Random initial state)	14.74
Neural network with learning	8.7
Simulated annealing (MAP)	6.72
MPM algorithm	7.05
Hierarchical network	8.21

Table 1: Example 2 - Percentage misclassification

6.1 Hierarchical Segmentation

The various segmentation algorithms described in the previous sections can be easily extended to hierarchical structures wherein the segmentation is carried out at different levels - from coarse to fine. The energy functions are modified to take care of the coupling between

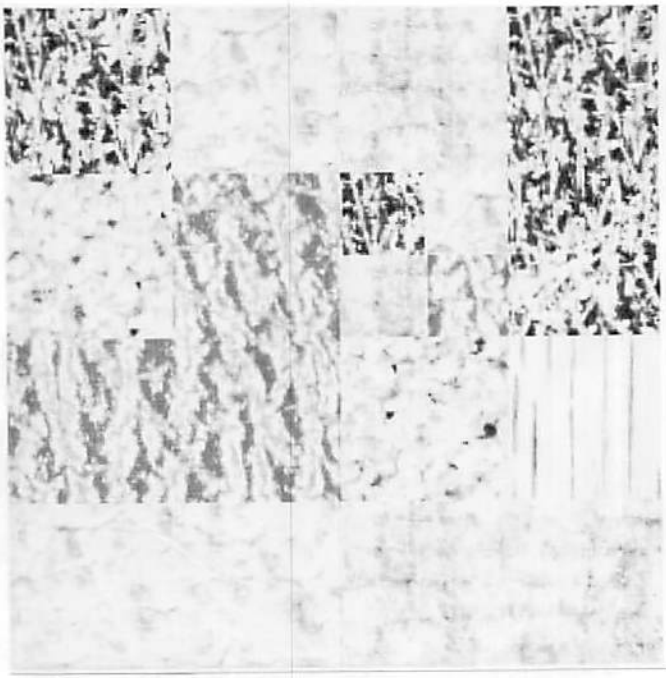


1.1 1.2 1.3

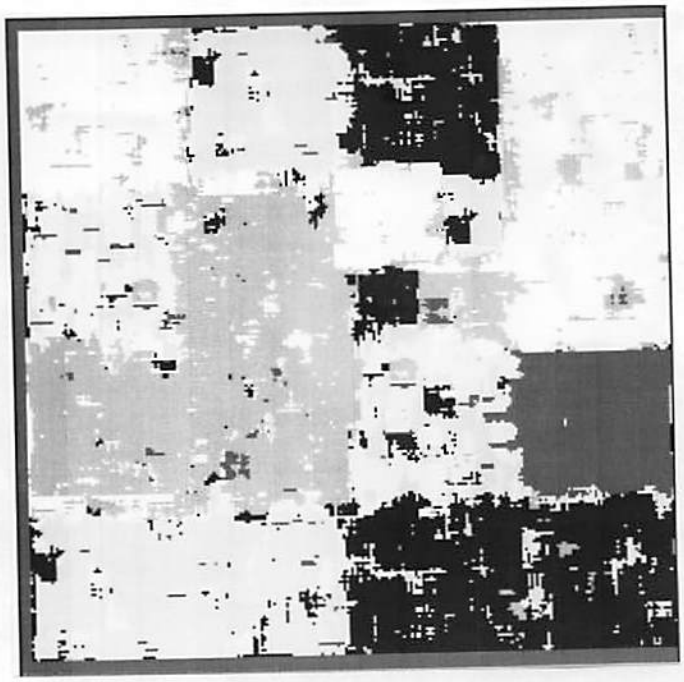
1.4 1.5 1.6

Figure 1: Two class segmentation problem - Example 1

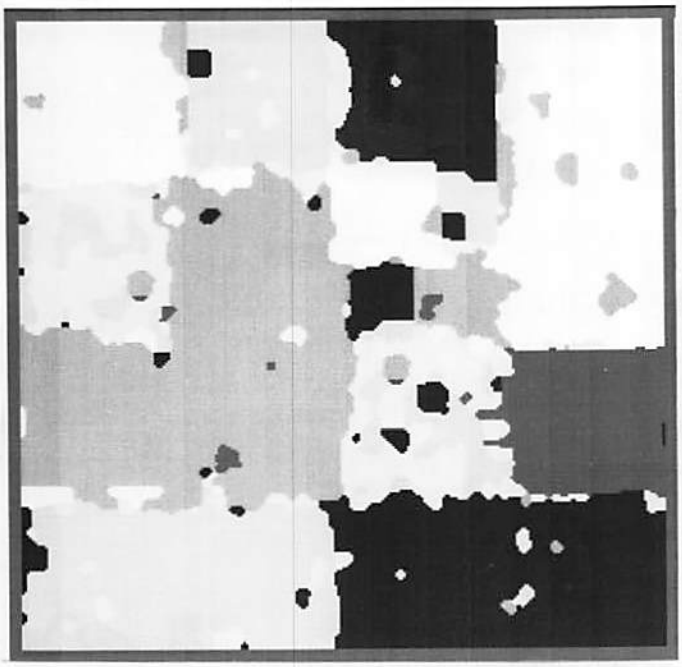
1.1 Original Image
1.2 Hopfield network solution with ML estimate as initial condition
1.3 Hopfield network solution with random initial condition
1.4 MAP estimate
1.5 MPM solution
1.6 Hopfield network with stochastic learning



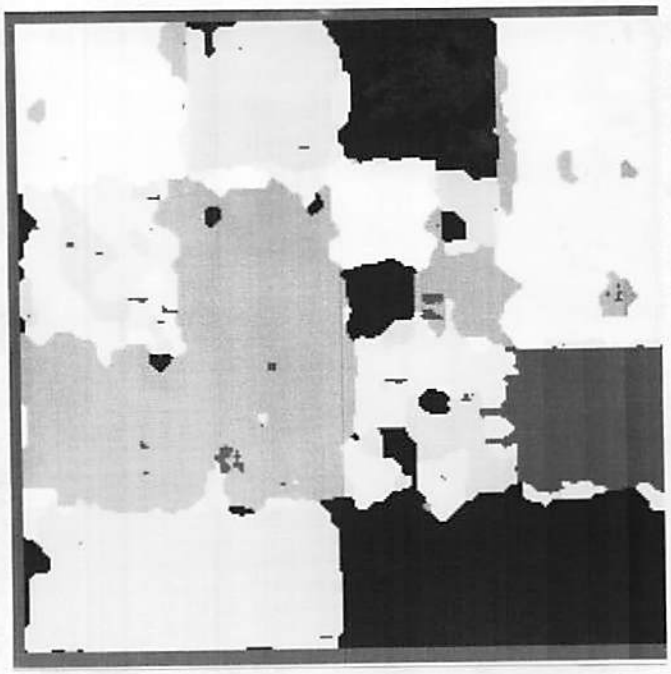
2.1



2.2



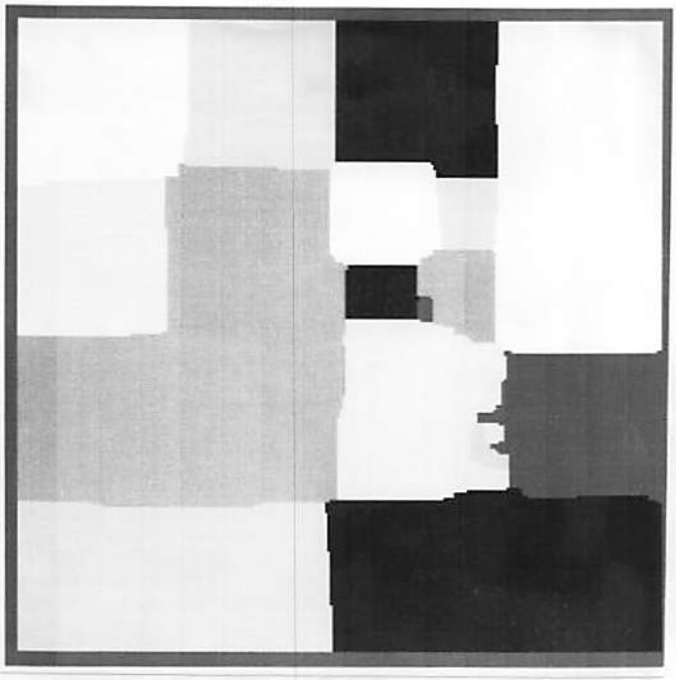
2.3



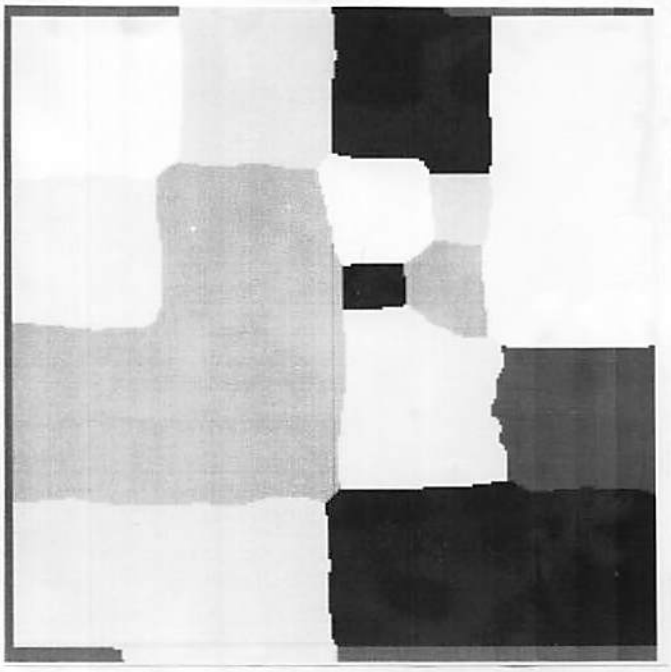
2.4

Figure 2: Six class segmentation problem - Example 2

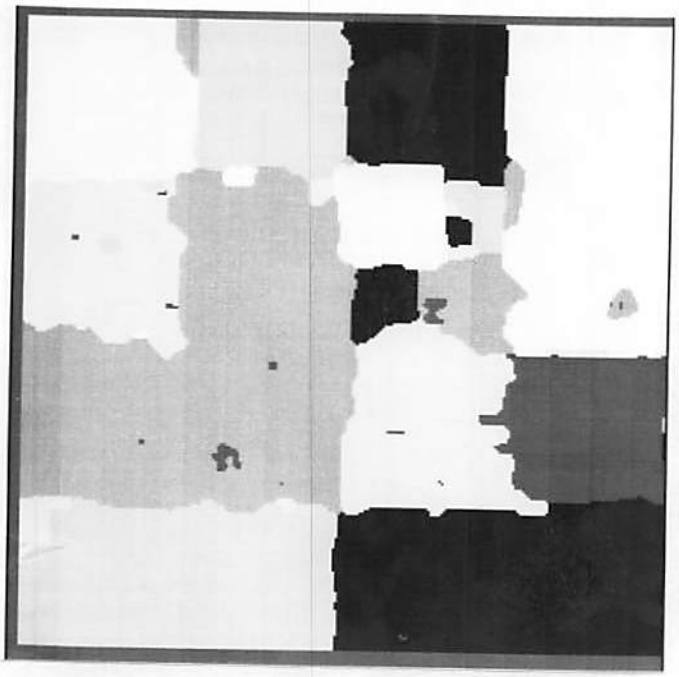
2.1 Original Image
2.2 ML Solution
2.3 Hopfield network solution with ML estimate as initial condition
2.4 Hopfield network solution with random initial condition



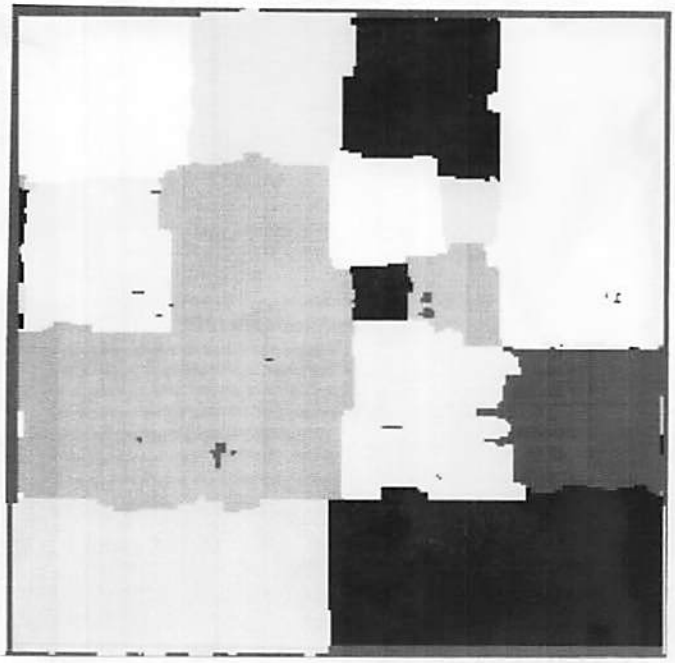
2.5



2.6



2.7



2.8

Figure 2: Six class segmentation problem - Example 2 (Contd.)

2.5 MAP estimate
2.6 MPM solution
2.7 Hopfield network with stochastic learning
2.8 Hierarchical network solution

the adjacent resolutions of the system. Consider a K-stage hierarchical system, with stage 0 representing the maximum resolution level and stage $K-1$ being the coarsest level. The energy corresponding to the k-th stage is denoted by $U_1^k(s,l)$ and $U_2^k(s)$ (equations 2 and 3). The size of the window used in computing the joint energy potential $U_1^k(.)$ increases with the index k. The potential U_2 is modified to take care of the coupling as below,

$$U_2^k(s) = -\beta \sum_{t' \in D_s^k} \delta\left(L^k(s) - L^k(t')\right) + \beta_k \left(\delta(L^k(s) - L^{k+1}(s)\right) + w\left(L(s)\right), 0 \leq k < K-1 \tag{19}$$

where $L^k(s)$ is the label for the site s in the k-th stage, β_k is the coupling coefficient

neighborhood set for the k-th

is shown in figure 2.8.

mentation algorithms based on

ed that a large class of natural

f several algorithms for texture

near optimal results as can be

l helps us to trivially map the

k. This deterministic relaxation

20-30 iterations for the 256×256

state of the system and often is

network a new algorithm which

twork was proposed. This helps

arning from the past experience.

nd stochastic relaxation schemes

n solving other computationally

References

[1] Y.T. Zhou, R. Chellappa,A. Vaid and B.K. Jenkins, "Image Restoration Using a Neural Network", *IEEE Trans. Accous.. Speech and Signal Processing*, vol. ASSP-36(7), pp. 1141–1151, July 1988 1988.

[2] Y.T. Zhou and R. Chellappa, "Stereo Matching Using a Neural Network ", In *Proc. IEEE International Conference on Accoustics, Speech and Signal Processing*, New York,NY, April 1988.

[3] Y.T. Zhou and R. Chellappa, "Computation of Optical Flow Using a Neural Network ", In *Proc. IEEE International Conference on Neural Networks*, SanDiego, California, July 1988.

[4] J.J. Hopfield and D.W. Tank, "Neural computation of decision in optimization problems", *Biological Cybernetics*, vol.52, pp. 114–152, 1985.

[5] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Inteligence*, vol.6, pp. 721–741, November 1984.

[6] J. Besag, "On the statistical analysis of dirty pictures", *Journal of Royal Statistic Society B*, vol. 48 No. 6, pp. 259–302, 1986.

[7] U. Grenander, *"Lectures in Pattern Theory"*, Volume I-III, Spinger-Verlag, New York, 1981.

[8] J.L. Marroquin, *"Probabilistic Solution of Inverse Problems"*, PhD thesis, M.I.T, Artificial Intelligence Laboratory, September 1985.

[9] S. Geman and C. Graffigne, "Markov Random Fields Image Models and their Application to Computer Vision", In *Proc. of the International Congress of Mathematicians 1986*, Ed. A.M. Gleason .American Mathematical Society ,Providence, 1987.

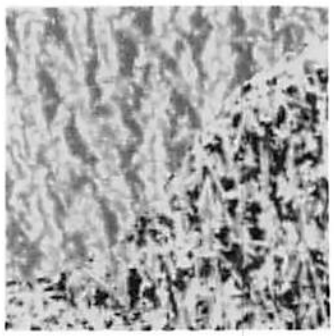
[10] B. Gidas, "Non-stationary Markov Chains and Convergence of the annealing algorithm ", *Journal of Statistical Physics*, vol. 39, pp. 73–131, 1985.

[11] Wheeler and K.S. Narendra, "Decentralized Learning in Finite Markov Chains", *IEEE Trans. Automatic Control*, vol. AC-31(6), pp. 519–526, June 1986.

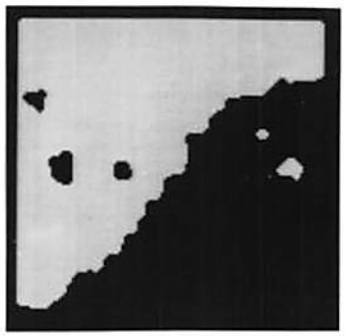
[12] M.A.L. Thathachar and P.S. Sastry, "Relaxation Labelling with Learning Automata ", *IEEE Trans. Pattern analysis and Machine Intelligence*, vol. PAMI-8(2), pp. 256–268, March 1986.

[13] R. Chellappa and S. Chatterjee, "Classification of Textures Using Gaussian-Markov Random Fields", *IEEE Trans. Acous., Speech, Signal Process.*, vol. ASSP-33, no. 4, pp. 959–963, August 1985.

[14] S. Chatterjee and R. Chellappa, "Maximum Likelihood Texture Segmentation Using Gaussian Markov Random Field Models", In *Proc. Computer Vision and Pattern Recognition Conf.*, San Francisco , California, June 1985.

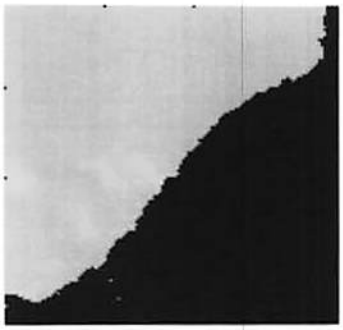


1.1

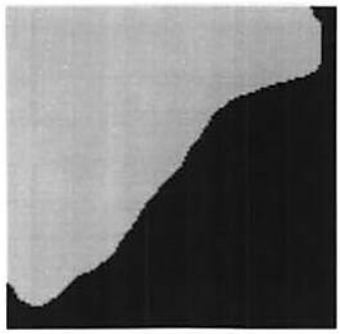


1.2

1.3



1.4

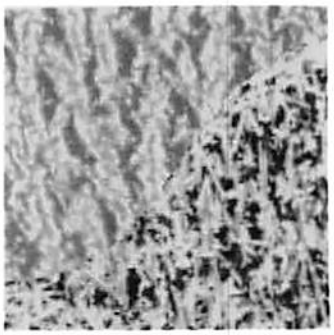


1.5

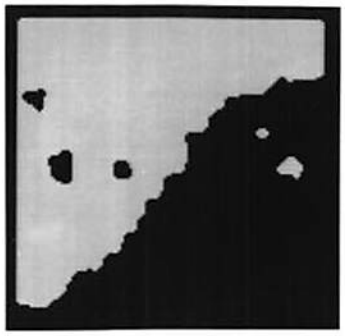
1.6

Figure 1: Two class segmentation problem - Example 1

1.1 Original Image
1.2 Hopfield network solution with ML estimate as initial condition
1.3 Hopfield network solution with random initial condition
1.4 MAP estimate
1.5 MPM solution
1.6 Hopfield network with stochastic learning

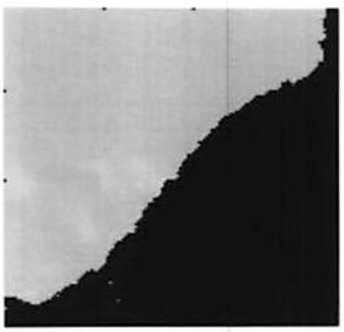


1.1

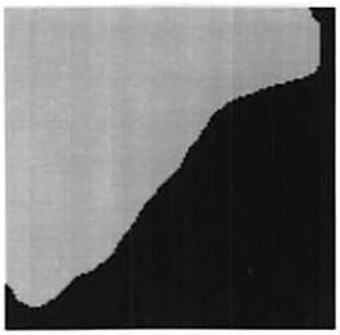


1.2

1.3



1.4

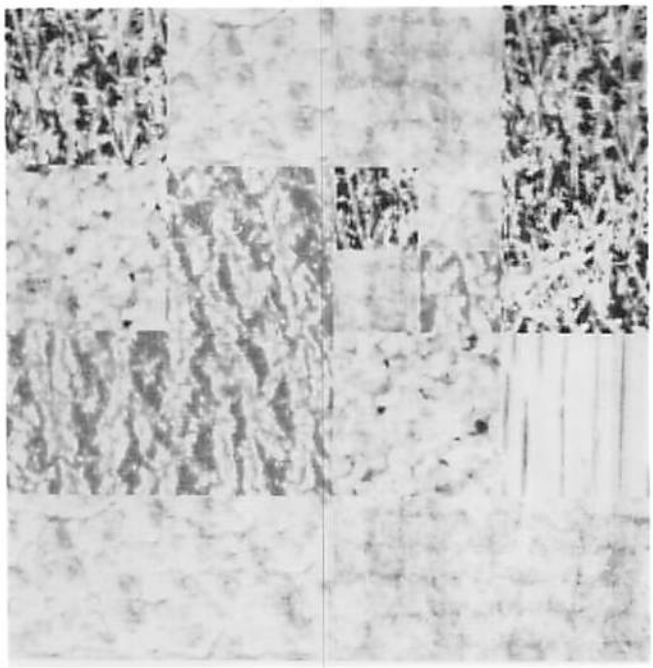


1.5

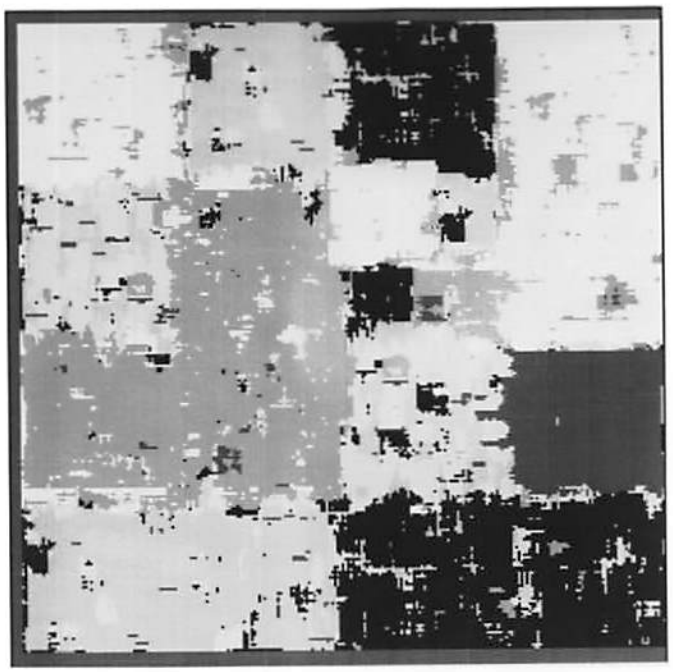
1.6

Figure 1: Two class segmentation problem - Example 1

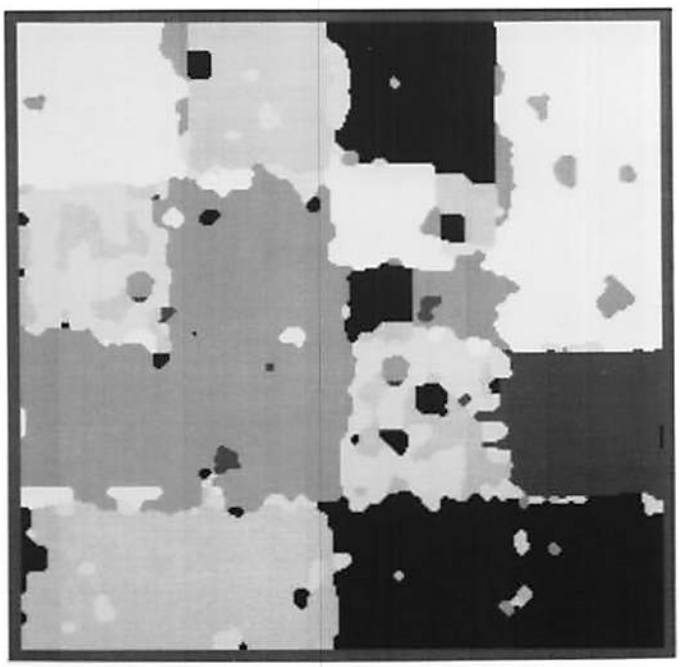
1.1 Original Image
1.2 Hopfield network solution with ML estimate as initial condition
1.3 Hopfield network solution with random initial condition
1.4 MAP estimate
1.5 MPM solution
1.6 Hopfield network with stochastic learning



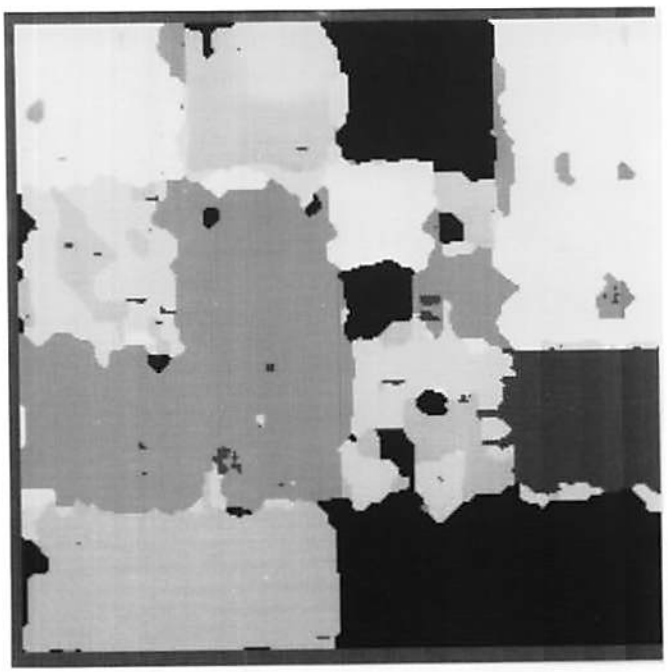
2.1



2.2



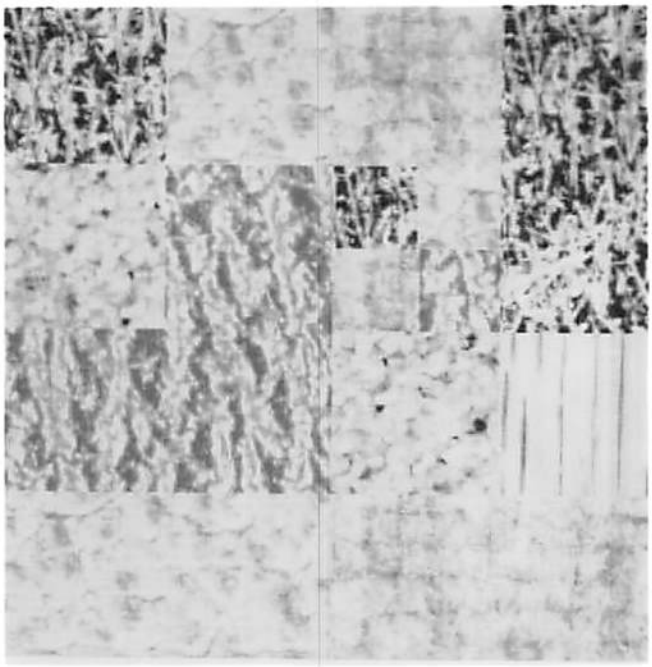
2.3



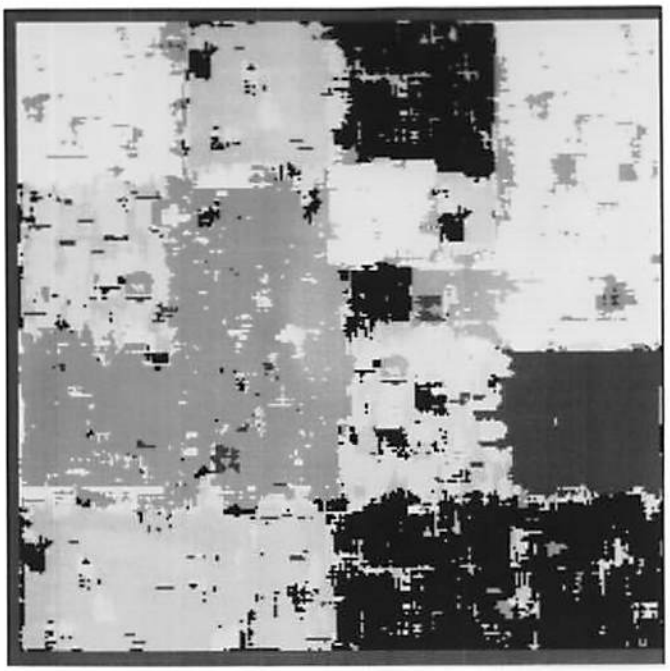
2.4

Figure 2: Six class segmentation problem - Example 2

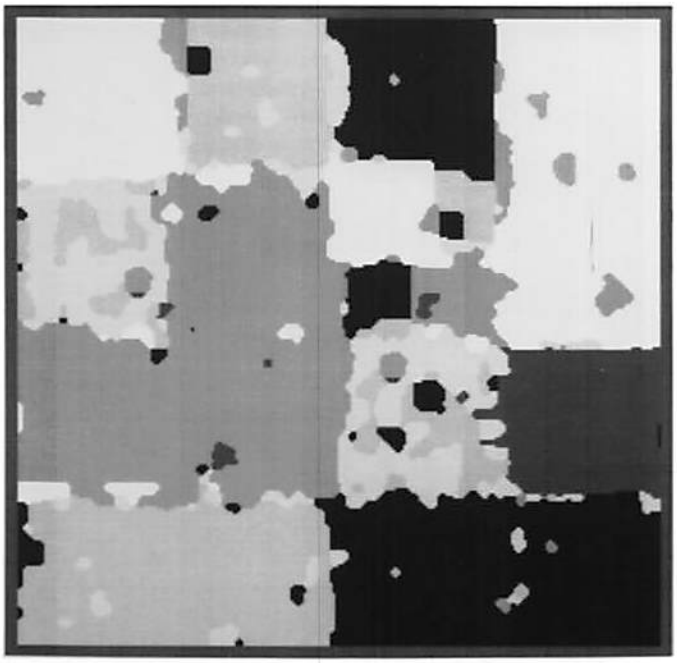
2.1 Original Image
2.2 ML Solution
2.3 Hopfield network solution with ML estimate as initial condition
2.4 Hopfield network solution with random initial condition



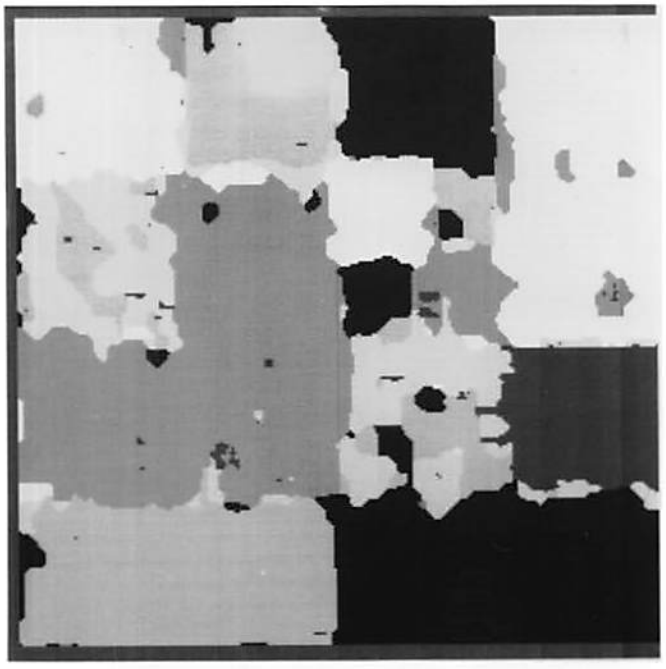
2.1



2.2



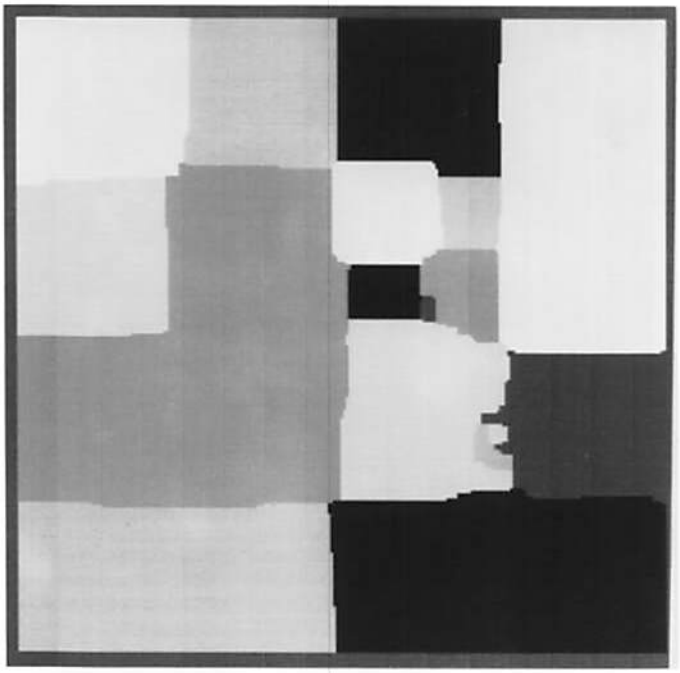
2.3



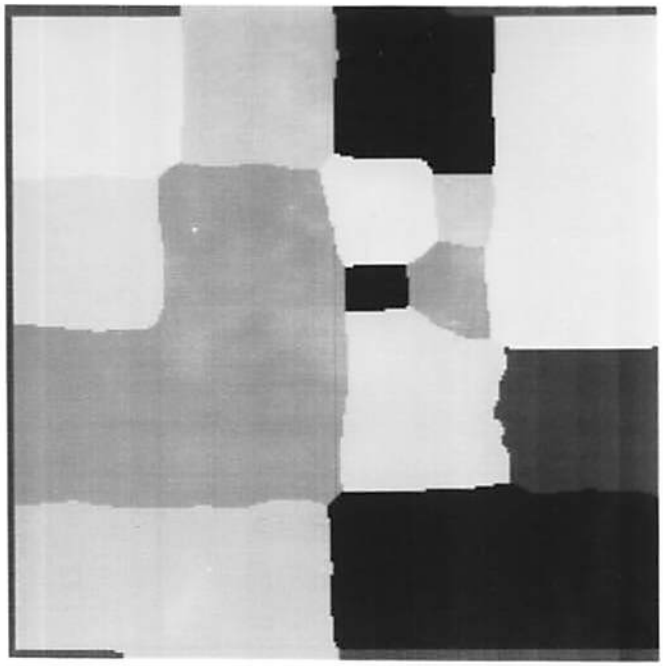
2.4

Figure 2: Six class segmentation problem - Example 2

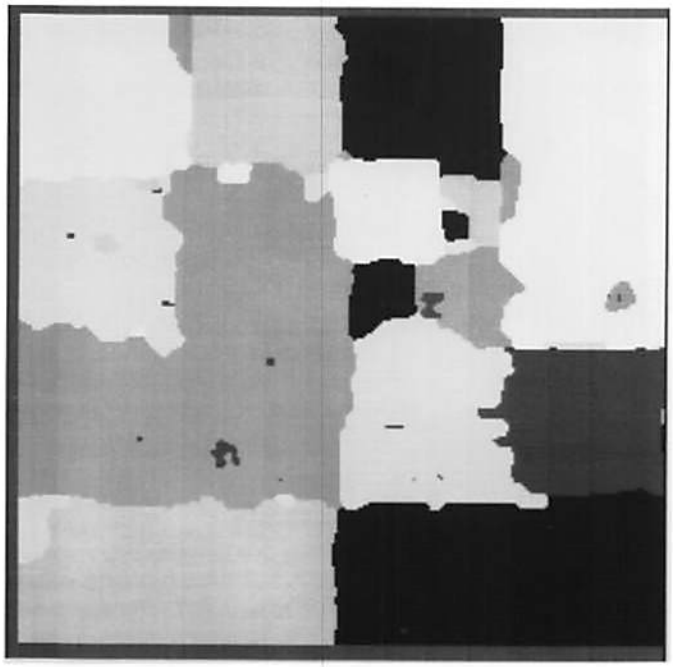
2.1 Original Image
2.2 ML Solution
2.3 Hopfield network solution with ML estimate as initial condition
2.4 Hopfield network solution with random initial condition



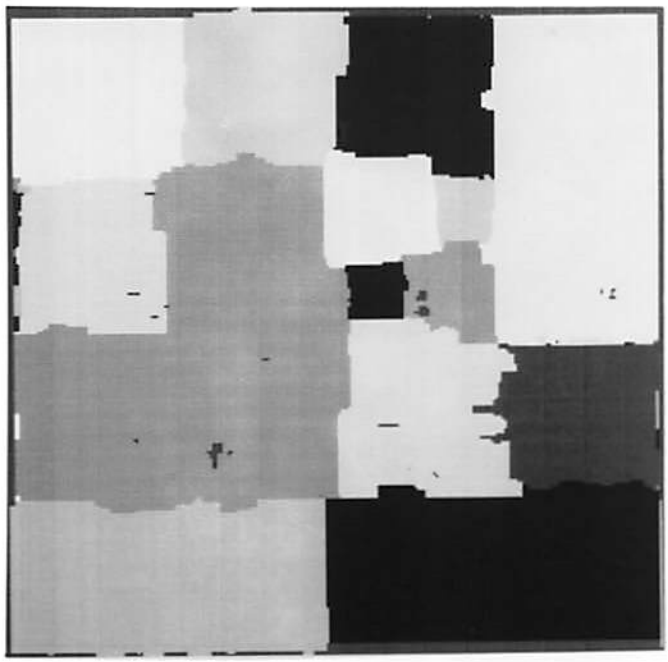
2.5



2.6



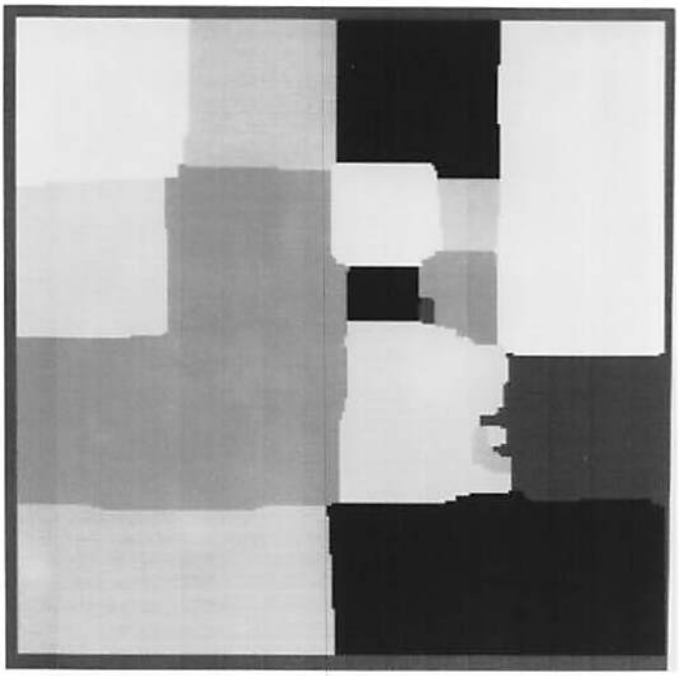
2.7



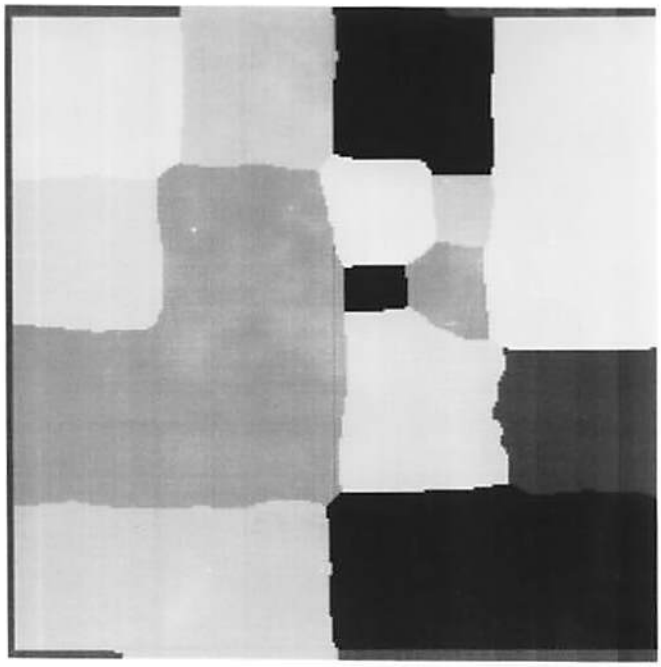
2.8

Figure 2: Six class segmentation problem - Example 2 (Contd.)

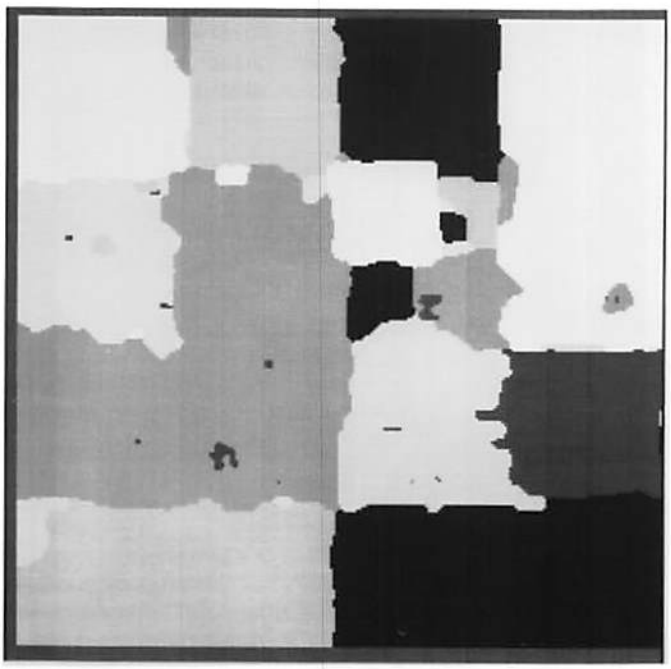
2.5 MAP estimate
2.6 MPM solution
2.7 Hopfield network with stochastic learning
2.8 Hierarchical network solution



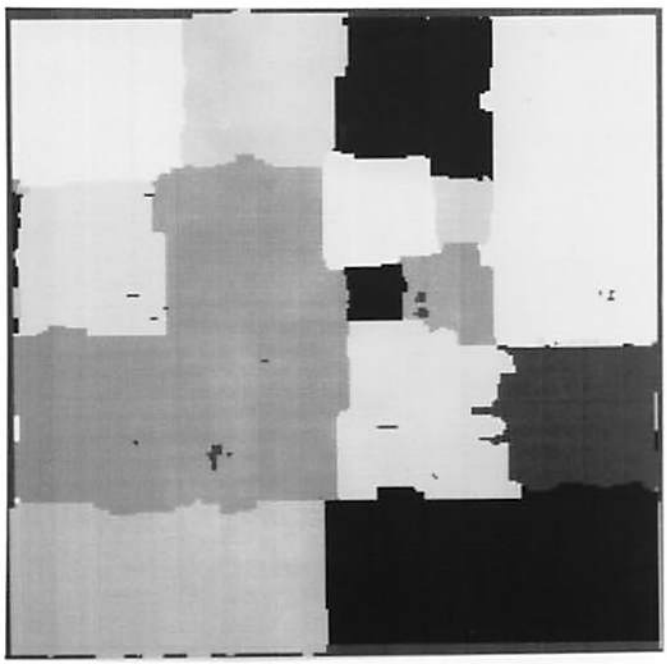
2.5



2.6



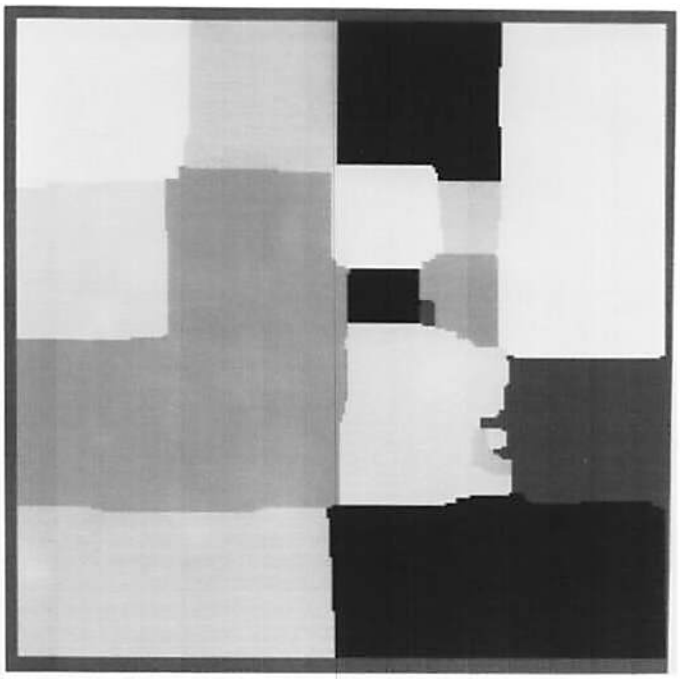
2.7



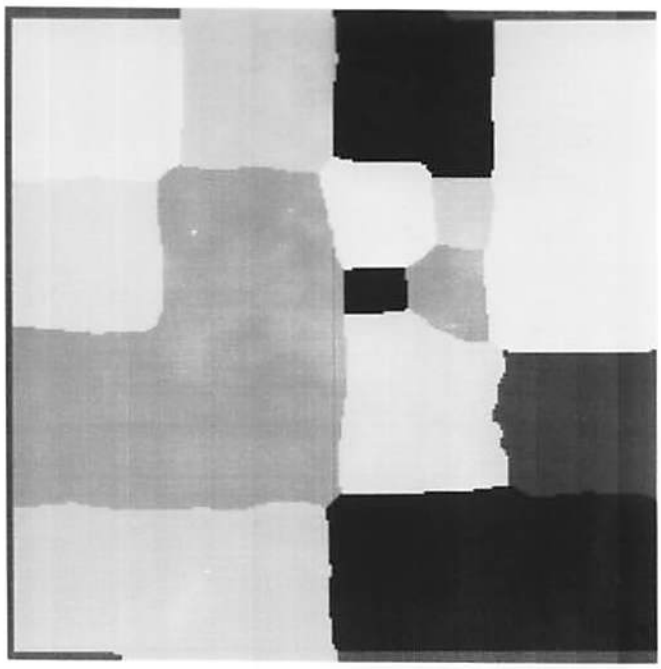
2.8

Figure 2: Six class segmentation problem - Example 2 (Contd.)

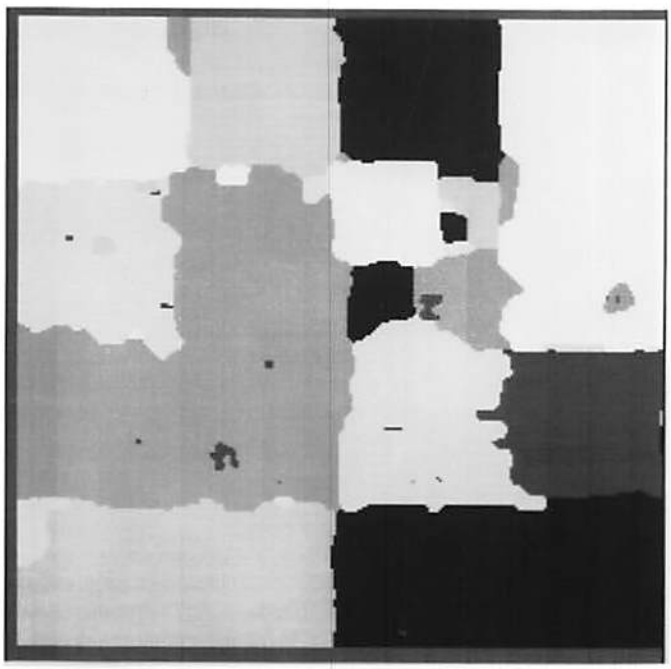
2.5 MAP estimate
2.6 MPM solution
2.7 Hopfield network with stochastic learning
2.8 Hierarchical network solution



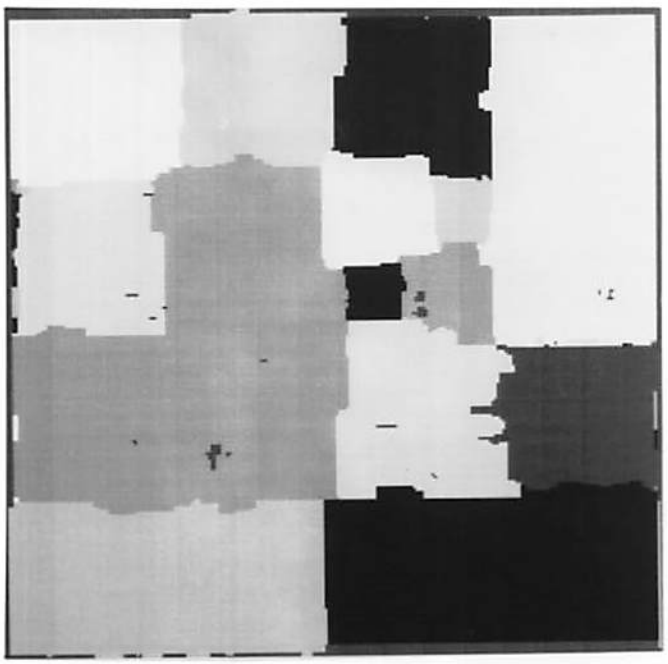
2.5



2.6



2.7



2.8

Figure 2: Six class segmentation problem - Example 2 (Contd.)

2.5 MAP estimate
2.6 MPM solution
2.7 Hopfield network with stochastic learning
2.8 Hierarchical network solution