NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

University of Alberta

RELAXATION AND MATCHING RELATIONAL STRUCTURES IN OBJECT RECOGNITION

by

Allan F. Randall 

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall, 1990

ISBN 0-315-64968-2

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Allen F. Randall

TITLE OF THESIS: Relaxation And Matching Relational Structures in Object Recognition

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1990

(Signed) .....
Permanent Address:
[illegible] Allen Str.
Antigonish Co., Nova Scotia
Canada [illegible] 1A0

Dated 4 October 1990

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled Relaxation and Matching Relational Structures in Object Recognition submitted by Allan F. Randall in partial fulfillment of the requirements for the degree of Master of Science.

Wayne A. Davis

Supervisor

Date 4 October/1990

ABSTRACT

This thesis is concerned with the problem of object identification. A system is developed to match data images against a computer model of a simple object, such as a cup or a stick-figure of a human. The images are represented as graphs of relations between parts, and the model is built up from training examples. The parts in the data are matched to parts in the model with a highly parallel technique called relaxation, much like the type of processing that occurs in the human brain. Simulated annealing (controlled noise) is used to avoid incorrect minima, a serious problem for all local search methods. Because sub-optimal answers are not acceptable in matching, this technique cannot solve the problem by itself in acceptable time. The system works well, however, at eliminating a large number of possibilities in a very small number of time steps, before a more traditional search program selects the final answer. Noise and uncertainty in the inputs are handled well, and a final estimation is produced of the chance that the matched parts are actually an example of the modelled object.

Acknowledgements

I would like to thank first and foremost my supervisor, Terry Caelli, of the Department of Psychology, as well as my co-supervisor in the Department of Computing Science, Wayne Davis. I must also thank my parents, Gordon and Joan Randall, and the rest of my family for their continual support. I also thank my aunt, Carol Randall, who has helped me out in many ways while I've been in Edmonton.

There are others who have helped with their discussions and insights, or in some other way: Walter Bischof, Xiaobo Li, Hong Zhang, Gilles Dionne, Dale Schuurmans, Robert Choi, Norbert Klinghofer, as well as all the folks at the First Annual Meeting of the International Neural Network Society in Boston. I must also thank my relatives, the late Gordon Cochrane, Marion and Spencer Cochrane for helping to make me feel very much at home during my stay in Boston.

I would also like to thank for their support: the Department of Computing Science, the Faculty of Graduate Studies and Research and the Alma Mater Fund of the University of Alberta.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1. RELAXATION AND MATCHING

The cognitive sciences have seen a surge in new parallel models of thought and vision over the past few years. Most of these fall under the category of *relaxation* techniques [Hop82, KJV83, RMC86]. Several attempts have been made to use such techniques on the matching phase of object recognition, when a visual scene is matched against an internal model of the world [Col87, Cos88, KIK79]. These techniques rely on an orderly, highly constrained environment, and difficulties arise when dealing with noise and uncertainty in the inputs. This thesis will explore the use of relaxation methods for matching in a toy world with plenty of noise and uncertainty in the data. Serious limitations and problems arise, and several strategies will be presented to deal with them.

1.2. A NEW APPROACH TO COMPUTING

As the fields of Artificial Intelligence and Computational Vision mature, more and more researchers are looking at parallelism as the answer to our computational problems. Recent developments in massively parallel computer architectures have rekindled an interest in parallelism on a vast scale--the scale of the human brain. While the recent machines do not come close to rivaling the parallelism of the brain, they are close enough to inspire a remarkably different way of programming. During the past decade, these trends were shaped by a move away from the highly *symbolic* paradigm of the sixteen-seventies, a change dramatic enough to be labelled by some as the *connectionist revolution*. Connectionist systems use a highly parallel search method called *relaxation*. It involves simultaneously weighing vast numbers of small, local pieces of evidence against one another in large, parallel networks. The best architecture for this style of computation has many simple, highly interconnected elements, much like our current notions of neural

structures in the brain. Chapter 2 will give a survey of the theory of relaxation.

Many feel these systems were discredited in the nineteen-sixties as too limited, while others on the connectionist side of the fence argue that these limitations have been overcome, and do not apply to modern connectionist systems. Both views are largely myth. Forcing oneself to adhere to the unique advantages *and* disadvantages of massively parallel systems leads to a different approach to the problem of cognition. Chapter 2 concludes with a discussion of the computational complexity of parallel networks when applied to various problems, including the famous Minsky-Papert limitations so widely mythologized.

1.3. AN APPLICATION TO COMPUTATIONAL VISION

In Chapters 3 through 5, these issues will be explored through an application to an important aspect of computer vision--the matching problem. Matching is the general problem of finding the correspondences between two structures. It is fundamental, not just in vision and AI, but throughout computer science. During object recognition, an image is first segmented into regions, after which various features, such as its shape and topology, are extracted from it. The resulting parts must be matched against an internal model of objects in the world. Chapter 3 outlines the more general form of this problem, which is well known as a fundamental NP-complete problem, meaning that as the size of the problem increases, the amount of computer time needed grows exponentially.

Chapter 4 explores the possibility of overcoming these problems with parallelism. A system is developed for matching images, represented as relational graphs, against an internal model that is built up from training examples. The system employs a fixed relaxation method that does not involve much learning, although some preliminary work was done that begins to explore this subject. The system is currently not an end-to-end system; its images are not real-world data, but are already hand-segmented into regions. They contain much noise and uncertainty, which makes them an appropriate starting point.

Chapter 5 presents the results, and compares them to previous work in this area. We will find that the technique of relaxation behaves the same whether for noisy, clean, large or small images. As long as we

have a stochastic mechanism to avoid spurious results, the nature of the process is unchanged, but there is substantial loss of performance owing not to a breakdown in the search technique, but to a fundamental change in the problem being solved. To cope with this difficulty and still make use of relaxation, we must redefine its role in the problem. These issues, along with possibilities for future work, are discussed in the conclusion, Chapter 6.

The Alberta Relaxation Engine, or ARE for short, is a general-purpose connectionist simulator, in which the matcher is written. It is a package of routines that allows easy incorporation of connectionist networks into C programs. The Appendices give a short lesson on its use and a reference of ARE commands.

Chapter 2
Relaxation Theory

2.1. INTRODUCTION

Relaxation is a highly parallel method of constraint satisfaction, where vast numbers of small, local pieces of evidence are weighed against each other in a large, parallel network. The evidences are *local* because each element of the network is only allowed to consider the *compatibility* of its own state with that of its *neighbours*, or those other units connected to it. At no time is the overall compatibility of a large collection of elements in the network considered. This provides for a highly parallel implementation. To accomplish this, each element of the network has a *compatibility measure*, or *weight* with its neighbours. The process is one of *iterative improvement*, where the state of each element is updated at each iteration to improve its compatibility with its neighbours. Each local improvement of an element will have an effect on its neighbours, which will be felt in the next iteration. The process continues until no more improvements are possible, and the network is said to be in a stable, *equilibrium* state.

Take, as an example, the task of *image thresholding*, a common problem in image processing. An image consists of a large array of picture elements, or *pixels*, which take on *gray-level* values (various shades of gray), or possibly color values. Assume we are dealing with a large spectrum of gray-levels, and we wish to *threshold* them, dividing them into two classes, black and white (0 or 1). A simple *first-order* solution can be obtained by setting to zero each pixel that is less than 0.5, and all pixels at 0.5 or greater to one. This is not the best way, however, since a lightly shaded pixel that is surrounded by many dark pixels should probably be black, not white. So, we would like to consider the state of the pixel's neighbours (those pixels directly connected to it) before setting the final value. But this is still not enough, because a change in this pixel's value will affect the optimal value for its neighbours, and changing their values will affect their neighbours, and so on.

A highly parallel way to get a good setting for all the pixels is to consider each pixel a simple computer processor, which is connected by a simple link to the surrounding pixels in the image. This link contains a weight which measures the compatibility of both pixels being white. For thresholding, if the neighbouring pixel is lightly shaded, there is a greater chance that the pixel itself will be white, so the weight between the nodes will be high. For example, a weight of +1 could represent complete consistency (both pixels are already white), while -1 could mean complete inconsistency (one pixel is white, the other black). A weight of 0 would mean that there is no preference (both pixels are moderately shaded), while other values in between these three could represent varying degrees of consistency.

Once these weights are set, the relaxation process begins. At each step, the value of the pixel is updated to increase its compatibility with its neighbours, in accordance with the weights (we will see later exactly how this works). If most neighbouring pixels are lightly shaded, and the pixel is dark, its value will get pushed up. This change will further affect the pixel's neighbours, and their neighbours, and so on, until no further changes are possible and all pixels are set to either white or black. The system has *converged* on a stable equilibrium.

This simplified example illustrates the basic principles of relaxation, but many other variations are possible. Some have different ways of labelling the elements in the set, others have different ways of updating the nodes to increase consistency, and so forth. All the variations, however, retain the basic properties of parallel iterative improvement based on the local weighing of evidence.

2.2. TYPES OF RELAXATION

There are several ways to distinguish between different classes of relaxation systems:
1) *discrete relaxation* vs. *probabilistic relaxation*
2) *deterministic relaxation* vs. *stochastic relaxation*
3) *fixed relaxation* vs. *adaptive relaxation*
4) *uncontrolled relaxation* vs. *controlled relaxation*

Discrete relaxation involves labelling each element, or unit, with discrete labels selected from a finite domain list, such as {0,1} or {red,green,blue}. *Probabilistic relaxation* is similar, except that the labels are

not discrete, but probabilities. Discrete relaxation can usually be considered a type of probabilistic relaxation where the spectrum between 0 and 1 is split into a discrete number of probabilities, such as {0,1} or {0,0.5,1}. As such, most general theories of relaxation apply to discrete relaxation as a special case. However, it is often easier to construct proofs around discrete systems, since they are finite.

In *deterministic relaxation*, each label is brought closer and closer to its locally optimal value at each iteration. Since each unit only considers local information, however, the true global solution can be missed if the system gets stuck at a *local maximum*. *Stochastic relaxation* introduces controlled *randomness* to the system to kick it out of these local dead-ends. Some of the more popular stochastic relaxation systems are the *Boltzmann Machine* [HS86], the *Harmonium* [Smo86], the *Gibbs Sampler* [GeG84] and the *Hopfield network* [Hop82, HT85]. Most stochastic systems use a process called *simulated annealing* to control the amount of randomness in the system.

Compatibility, or consistency, is measured by pair-wise *compatibility functions*, or weights, between the units. In non-adaptive systems, like the pixel-thresholding example, these functions are fixed, designed for the problem at hand. *Adaptive relaxation* uses principles of *learning theory* to learn the compatibility functions between units, which usually have little, if any, built-in structure.

Controlled relaxation is a general term referring to a system where the units receive information from some external source other than the neighboring relaxation units [ChG87, RAG88]. This external source can be anything from a higher-level, sequential supervisor to another relaxation network.

2.3. SYMBOLISM AND CONNECTIONISM

The history of cognitive science is marked by a controversy between two schools of thought, *symbolism* and *connectionism*. These two paradigms are characterized in many ways, each with different connotations and uses, but representing the same dichotomy of thought, which is expressed in the following chart.

Two Paradigms of Cognitive Science

Symbolism	vs.	Sub-symbolism
Atomic Symbolism	vs.	Pure Connectionism
Knowledge-based Systems	vs.	Self-organizing Systems
Localized Representation	vs.	Distributed Representation
Artificial Intelligence	vs.	Cybernetics
Traditional AI	vs.	Connectionism
Serial Processing	vs.	Parallel Processing
Symbols	vs.	Numbers
Formal	vs.	Informal
Top-down	vs.	Bottom-up
High-level	vs.	Low-level
Logic	vs.	Statistics

These so-called opposing notions are not really mutually exclusive at all, and this chart is not meant to lend credibility to divisions of the world into such black and white categories. There is really no need to choose any one of these ideas over another. However, this chart does depict a very real tension between different research tendencies that has always existed within the field, and the question of their relative roles remains a valid one.

In the early years of modern cognitive science (1940's-50's), the idea of neural modelling was heavy in the air [McC54, McP43, Rose62, Wie600, Wie61]. The apparent on/off behaviour of neurons in the human brain seemed tantalizingly similar to the switching circuits in digital computers, so early attempts to simulate human thought concentrated on ideas gleaned from neurophysiology. Over the years, this paradigm was replaced by the idea of modelling cognition on a higher level--the level of *symbols*. Neurons in the brain were seen as no more relevant than the boolean switches of a digital computer. The brain's power, say the symbolists, derives from its ability to manipulate symbols, where they refer to things in the world.

This is the theory of *physical symbol systems* [New80].

The 1980's have seen a resurgence of interest in lower-level, *sub-symbolic processing*, the idea that symbols cannot be the lowest computational level in an intelligent system. These *connectionist* systems build symbols up from a non-symbolic, lower-level substrate—the *neural* level. Knowledge is not represented at the symbolic level, but lies in the strengths of the connections between the sub-symbolic units. Knowledge is not rule-based, knowledge is pure pattern.

The pure connectionist believes that no mention of the symbolic level is necessary to understand intelligence. The atomic symbolist believes that this level is the appropriate bottom layer of an intelligent system, and that symbols are properly treated as atoms. The field as a whole is moving towards a more pragmatic combination of these two approaches. Symbolic and sub-symbolic approaches, many feel, will both be needed to produce intelligent behaviour.

The debate between these two approaches to cognitive science is a complicated and intensely controversial one. The philosophical landscape has only been briefly sketched out here. See [Gar87, McC79] for a history of cognitive science and [Hof79, Hof85, Jak83] for some philosophical arguments in favor of connectionism. See [FoP88, New80] for the more traditional view. [McP88] (pp. 245-269) gives a view somewhere between the two extremes.

2.A. NEURAL NETWORKS

Relaxation systems are often called *ideal neural networks*, as relaxation is thought by many to be a major form of processing in the brain. Each unit works much like a neuron in the brain. It sums its inputs and thresholds the output. Such systems were formalized by [McP43]. The first computer simulation is due to [FeC54].

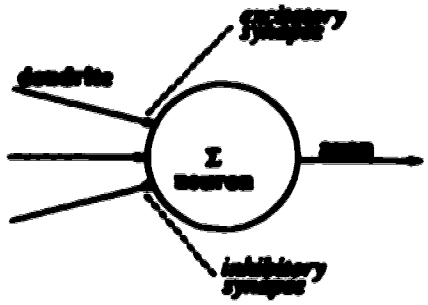


Figure 2.1 A simplified neuron.

The brain is a massive network of nerve cells, or *neurons*. Figure 2.1 shows an idealized neuron. The neuron is an adder, accepting many inputs by way of incoming *dendrites* and computing the sum on the single outgoing *axon*. This axon then branches out, providing input to perhaps many thousands of other neurons. There are about ten billion of these processors in the human cerebral cortex (the part of the brain responsible for higher thinking).

The input signals to a neuron must jump over a tiny gap between the incoming dendrite and the neuron, called the *synapse*. The signal is modified by the strength of the synapse, which can change over time as new experiences are learned (or so goes current thinking). There are two basic types of synapse: *inhibitory* and *excitatory*. Inhibitory synapses, distinguished in Figure 2.1 by an open circle, tend to *hyperpolarize* the cell body, making the voltage more negative. An excitatory synapse, represented by an arrowhead in the figure, will *depolarize* the cell body, making the voltage more positive. All the various synaptic inputs to a cell act together, affecting the voltage of the neuron. If that voltage rises above a certain threshold (normally about -50 mv), the neuron fires an *action potential*, a brief spiking of the voltage to around 50 mv. This action potential was mistaken in the past for a binary on/off encoding. In actuality, it is not the strength of the action potential that encodes the voltage of the cell body, but the *frequency* with which they are fired.

In terms of network theory, the brain is a huge network of nodes (neurons) connected by weighted arcs (synapses). Each node is a simple adder of the incoming weighted arcs, each input being multiplied by

the weight of its arc. A positive weight corresponds to an excitatory synapse and a negative weight corresponds to an inhibitory synapse. When the value of the node reaches a certain threshold, the node fires the value along all its outgoing arcs.

Each neuron in the brain, while computationally simple, can have thousands of connections with other neurons. There are about 10^{10} neurons in the cerebral cortex. This is parallelism on a massive scale. Massively parallel machines are only now starting to make head-way [HIL85], and we are a long ways off from matching the human brain. Jerome Feldman has estimated that, with a cycle time of about 10 *ms*, a simple visual task like recognizing your grandmother must be performed in about 100 sequential steps. This is known as the 100-step, or Feldman constraint.

The above is a greatly simplified view of the functioning of neurons, and its accuracy as a model of neurological functioning is still in question [CrA86]. It is not really possible yet to know what the appropriate level of abstraction is. The system developed in coming chapters is neurological only in the most general sense that it uses a parallel relaxation method.

2.5. LEARNING THEORY

There are different views on what exactly constitutes connectionism. Some emphasize the massively parallel aspect. Others stress learning and what is called *parallel distributed processing*, which is the notion that knowledge is not represented formally, but is distributed throughout a system as patterns of strengths on the connections between nodes in a complex network. A typical adaptive connectionist network (see Figure 2.2) consists of a layer of *input neurons* into which an experience, such as a visual scene, is placed. This pattern is propagated through the system of neurons to the final *output layer*. There may or may not be some *hidden layers* between the input and output. If the system is presented with a certain input pattern and a certain output pattern repeatedly, and the weights on the synapses are modified by some formula, the network should converge on a set of weights that will yield the output pattern whenever the input pattern is presented [Koh77].

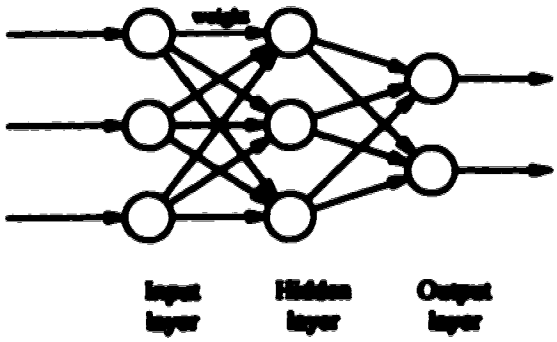


Figure 2.2 A perceptron convergence learning mechanism.

There are two basic types of learning a system like this can perform. In *associative learning*, an input pattern results in a certain output pattern, which may be some other arbitrary pattern (concept association), or a much simpler pattern that is activated by a whole class of inputs (concept generalization). In *discovery*, on the other hand, the system learns to respond to interesting features in the input patterns.

Early systems like the *perceptron* [Rose62, WIDH60] were, like the example in Figure 2.2, unidirectional. Information flowed in one direction from input cells to output cells. Another problem with the perceptron was that it was not multi-layered. Modern models are more complex, with both multi-layers and feedback, and perform better. Single-layer perceptrons are still useful, however, in limited situations and for constructing mathematical proofs. Minsky and Papert's classic book *Perceptrons* [MiPa88] remains the definitive work on the limitations of such systems. Minsky and Papert showed that some problems that are tractable in the symbolic (serial) paradigm become intractable in the purely parallel environment of a single-layer perceptron. The general applicability of Minsky and Papert's theorems is still a matter of controversy. The central core of the system described in later chapters will not employ any heavier guns for learning than a simple perceptron, as the adaptive aspect of the work is currently preliminary. However, it is important in motivation and to future extensions.

2.6. INFORMATION THEORY

Much of the stochastic relaxation mechanism is based on principles from information theory [Sha48]. The term *information* is often used with differing and conflicting meanings, but it usually has something to do with the degree of structure, or order, that a system contains. Thus, a book written in English has a higher information content than the same-sized book filled with random characters. Fewer bits are required to completely describe the English book than the random one. Note that *fewer* bits means *higher* information content, because there is more structure in the system.

Shannon [Sha48] first formalized information mathematically by introducing the logarithmic measure of information, call it I. For a system of n elements, each of which is set to one of m possible values, there is a total of $\Omega=m^n$ possible system configurations. If all these different configurations are equally probable, then the *source* of the information is said to be *homogeneous* and the information content is a straightforward measure of the number of bits[1] needed to specify any one of the configurations: $I = \log_m \Omega = n$.

Sometimes, however, the information source produces some configurations with greater probability than others. The information content of a particular configuration x, or its *self-information*, is (like I) a logarithmic measure: $H(x)=-\ln P_x$, where P_x is the probability of configuration x. When all probabilities are equal, this reduces to I. *Algorithmic information theory* [Cha77] treats $H(x)$ as the number of bits in the smallest Turing machine[2] program that prints out x. The *average self-information* of all the possible configurations is the *entropy* of the information source: $S = \bar{H}(x) = -\sum_x P_x \ln P_x$. Entropy measures the *complexity* (or *disorder*) in a system. High entropy means many bits are required to specify the average configuration of the system, and thus the system encodes little information.

From these concepts comes Shannon's *principle of maximal missing information*, or PMMI (Principle of Minimum Entropy). This principle states that, given a system we have little information about, the most

---

[1] Bits are determined by the base of the logarithm. Many bits correspond to such. Natural logarithms (used, which will be used here, correspond to *natural bits*.

[2] A Turing machine is a simplified, prototypical computer used in theoretical computer science.

likely way to "fill in the blanks" is to choose the probability distribution with the most "missing information," or highest entropy. If all we know is that the average probability is 0.8, then the distribution (0.0,0.2,0.3,0.5) would be more likely than (0.1,0.1,0.4,0.4) because it has higher entropy.

2.7. THE MECHANISM OF PROBABILISTIC RELAXATION

The topic of probabilistic relaxation will be introduced using, by and large, the formalisms of [RoKH2, RoHD86]. Assume we have a set of propositions, which can be true or false. A_i will be the ith such proposition, and p_i will be the probability that it is true.[3] An object is connected to each of its neighbours by a *compatibility measure*, or weight that tells us to what degree the current probabilities of the pair are *consistent* with each other. The consistency of both A_i and A_j being true will be labelled w_{ij}.

p_i = probability of A_i.
w_{ij} = the consistency of $(A_i \& A_j)$.

This formulation uses the logical AND ($\&$) function as the interpretation of the weights, so that $w_{ij}=w_{ji}$. Asymmetric relations, where $w_{ij} \neq w_{ji}$, are also possible, although the mathematical analysis becomes more complex. Thus, the symmetric weights are generally assumed for theoretical work.

At the start of the iteration process, each probability p_i is given some initial value. Each probability is then updated according to some compatibility measure to be more consistent with its neighbours, until equilibrium is reached and further iterations fail to increase consistency. This is called *hill-climbing*, because the system "climbs" the hill to the most consistent state. The process can also be equivalently expressed as minimization of *error*, *energy* or *entropy* [PERE86, Hop82, Smo86], called *gradient descent* because the system descends the "hill" in the direction of the steepest gradient (or first derivative).

---

[3] Note that p_i is the probability for one particular element of the system, which is not to be confused with P_n, from the previous section, which is the probability of one particular configuration of the entire system.

The weights measure compatibility, which usually means positive values for highly consistent pairs of nodes and negative values for highly inconsistent pairs. Since the nodes are linear summation units, this will drive the probability of the node down when it is inconsistent with its neighbours, and up when it is consistent. The simplest update rule for this just adds the linear summation to the previous value of the node: $p_i^{(r+1)}=p_i^{(r)}+\Delta p_i^{(r)}$, where the value of p_i at iteration r is $p_i^{(r)}$, and the value of the increment (the linear combination) at step r is $\Delta p_i^{(r)}$. However, Rosenfeld and Kak use a slightly more complicated version to ensure that the node values behave as proper probabilities:

$$p_i^{(r+1)} = \frac{1}{Z} p_i^{(r)}(1 + \Delta p_i^{(r)})$$

$$\Delta p_i = \eta \sum_n p_n w_{in}.$$

$$Z = \sum_n p_i^{(r)}(1 + \Delta p_i^{(r)}).$$

Z is a normalization factor and is necessary to ensure that the probabilities sum to unity: $\sum_n p_n=1$. Instead of adding the increment to the old value, it is first added to 1, and then multiplied by the old value. This ensures that the probabilities are never negative. The increment itself, Δp_i, is the linear combination, or weighted sum of the inputs. The size of the steps taken in the gradient-descent is determined by the proportionality constant, η. High values of η mean large steps are taken in the hill-climbing, which causes faster convergence, but runs the risk of overshooting the correct spot.

2.8. CONVERGENCE CRITERION

There are three ways a system can converge to equilibrium:
1) Fixed-point convergence: the system converges on a fixed state.
2) n-point convergence: the system oscillates between a states.
3) Chaos: the system converges on a seemingly random state with infinite complexity and no fixed point or cycle.

Notice that 1) is a special case of 2) with n=1. Which type of convergence is desired depends very much on the specific problem. Chaotic cycles are characterized by seemingly random, unpredictable behaviour, yet they are actually quite stable, representing a sort of local minimum, but in an infinitely com-

plex, *fractal* space. Such a stable, yet chaotic convergence is called a *strange attractor*. For an introduction to chaos, see [Gle87] and [Bel87]. For a general mathematical introduction to discrete iterative systems, see [SRT87].

As a general heuristic, it can be assumed that if a system is stable, but randomly oscillating, it is behaving chaotically.[4] So to test for convergence, wait until the system has iterated i iterations (where i is some integer) without the energy going below the lowest energy state so far encountered. If the system has not reached equilibrium, the total energy (inconsistency) will still be going down. If it is an n-point convergence or chaos, the system will be stable and the inconsistency will stay at a relatively constant level. The higher i is, the more certain we can be that the system has converged. This test for convergence satisfies all three of the above possible equilibrium situations, in the limit.

2.9. ADAPTIVE UPDATING

The basic learning rule for connectionist systems, known as the *Hebbian learning rule*, is simple: whenever the connection between two units is activated, its weight is strengthened. Any signal now travelling across this connection will be strengthened, so when a pattern is *clamped* on the input units (so that they can take on no other values), it leaves a memory trace in the synaptic weights of the connections. The update rule for the weights is: $\Delta w_{ij} = \eta'' p_j q_i$, where q_i is the *teaching input* for p_i. The *learning rate* is the proportionality constant, η'', which indicates the size of the steps to take in adjusting the weights. When the training phase is complete, the teaching units are unclamped, so they are once again free to take on their own values. Now, whenever the input units are clamped with a previously stored pattern, the associated pattern should appear on the output units. If we clamp the inputs with a vector that is similar to a stored memory trace, the system will hopefully approximate the associated pattern on the output bank, due to PMIE (Principle of Minimal Energy).[5] As in the human brain, the association need not be exact to work.

---

[4] There is a more rigorous test for chaos, desirable for discussions dealing heavily in chaos theory, called the *Lyapunov exponent*.

[5] Note that there is both a maximum entropy and a minimum entropy principle at work here. Minimizing entropy through relaxation provides a highly ordered (and therefore correct) system configuration. PMIE ensures that any existing information

Such pattern associators are best at matching similar patterns onto each other [MP69]. These are weak, highly underconstrained learning methods, and so are often applied when little is known about the problem domain. Unfortunately, without prior knowledge of a problem, the search space becomes too large for weak learning methods. Yet if a system is too problem-dependent, it cannot create the new representations necessary for learning. Striking a proper balance between these two extremes is the central difficulty of learning theory.

In practice, a more complicated rule than the Hebbian is usually used. The most common of these modifications is the *delta*, or *Widrow-Hoff learning rule* [Koh77, RMC86, WH60]. This is needed because of interference between different input patterns. Instead of learning the teaching input, this rule learns the *difference* between the teaching input, t_i, and the actual free-running value, p_i. For complete linear independence of input patterns, the simple Hebbian rule is enough, since there is no interference, and hence no difference between p_i and t_i.[6] The form of the rule is: $\Delta w_{ij} = \eta'' (t_i - p_i) p_j$.

2.10. LINEAR THRESHOLD UNITS

So far, the relaxation increment, Δp_i, has been completely linear. The problem with such linear models is that they are limited. It has been proved [MP69], in terms of what such systems can in principle learn, that nothing can be gained by adding more layers or feedback. A more common type of connectionist system is the *linear threshold* model, where the update of a node's value is only done if it exceeds some threshold value, θ_i. One popular way of implementing this is to output a 1 if the weighted sum of the inputs exceeds some threshold value and to output a 0 otherwise. The new update rule is:

$$\Delta p_i = \begin{cases} 1 & \text{if } (\sum_j p_j w_{ij} - \theta_i > 0) \\ 0 & \text{otherwise.} \end{cases}$$

---

will be as constrained as possible.

[6] Here, again, we see both maximum and minimum entropy principles at work. Adjustment of the weights according to t_i is, as before, minimizing the entropy. The error correction accomplished by using here current the free-running value p_i is a maximizing of entropy.

This makes the nodes into reasonable models of neurons. Hidden units can now be placed between the input and output layers, allowing a memory trace to propagate through several layers of units, and build up internal representations. For such a system to be adaptive, a new learning rule must also be devised to learn the weights on the hidden units, since t_i has no meaning for a hidden unit. The most common is *back-propagation*, which allows the error signal (the difference between teaching and free-running values) to propagate back through the internal layers of a system.

2.11. STOCHASTIC UPDATING

Relaxation systems have the advantage of highly parallel, local computations, but local maxima are a serious problem, and hill-climbing systems are notorious for them. Figure 2.3 illustrates the problem, expressed as a minimization of energy.[7] Energy is plotted on the y-axis, while the configuration of the system is represented by the x-axis. All the dimensions created by the many units are collapsed here into one dimension on the x-axis. The number of dimensions represented by this axis is equal to the number of relaxation units. The interaction between two units can be visualized as a three-dimensional landscape, by adding a z-axis for the second unit. For larger systems, this landscape, or *state-space*, can be incredibly complex, multi-dimensional, as well as fractal. In Figure 2.3, point B will converge on point B^*, a local minimum. The true global minimum is at A^*, which cannot be reached by purely local adjustments, unless we are lucky and start out close by (at point A, for instance). Notice that point C produces a very good solution, at C^*, but still not the absolute best. In many situations, this is good enough.

For adaptive systems, the convergence on the correct set of weights can also be viewed as hill-climbing. Here, the x-axis in Figure 2.3 would represent the configuration of weights, rather than the actual values of the units. The delta learning rule hill-climbs in this *weight-space* to minimize the sum-squared error between teaching samples, just as the stochastic update rule hill-climbs in *energy-space* to minimize the inconsistency of the system configuration. Both types of hill-climbing are similar, gradient-descent

[7] Figure 2.3 can be changed into a maximization problem simply by turning the diagram upside down.

techniques and suffer all the same local minima problems. Since the weights determine the consonancies of the configurations, a single point in weight-space corresponds to an entire energy-space landscape.

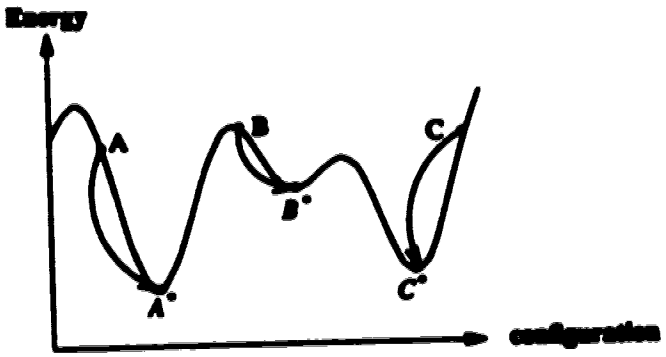


Figure 2.3 The energy landscape.

An intuitive way of understanding the problems of hill-climbing is to imagine yourself lost in a dark landscape of hills and valleys. You wish to climb to the highest peak in the territory, but you cannot gaze around to find the highest mountain, because it is pitch dark (or perhaps you are blind or there is a heavy fog). The best you can manage is to feel around to find which way is up, and climb in that direction. In other words, you have only *local* information with which to find a *global* solution. If the hill you are on is small compared to one further away, you will be trapped at its peak (a local maximum) and unable to progress further. This is a serious problem with landscapes of any appreciable complexity.

Another popular image that is often used is of a ball-bearing on a sheet of hills and valleys. Here, we are thinking in terms of energy minimization, so we want to find the deepest valley, instead of the highest mountain. If the ball-bearing is placed at some random spot and released, gravity will pull it down, and it will settle into the nearest valley (a local minimum). Stochastic relaxation attempts to solve this problem by introducing randomness to kick the system out of the minimum. Any particular update of a node has a certain probability of moving in the "wrong" direction and increasing global inconsistency. The mechanism usually used for this is *simulated annealing* [KJV83, MRR53], based on the concept of thermal equilibrium.

The system is started out with a high degree of randomness, called the *computational temperature*, which is lowered quickly at first and then gradually as equilibrium is approached.

This means that we start by vigorously shaking the sheet, so the ball-bearing bounces around mostly at random. As we gradually shake less and less violently, the ball will tend more and more to the deeper parts of the landscape, until finally, when the shaking has stopped, it will be at a local minimum. Except here, the minimum will be much deeper than the superficial one arrived at deterministically (without shaking), although it will not necessarily be the deepest valley of all. The more gradually the shaking is slowed down, the more confidence one can have in finding the true global minimum. The optimal annealing schedule [GeG84], takes too long for practical purposes, and a faster schedule must be chosen. For most applications, we must be content with a deep minimum rather than the deepest.

To implement this in a connectionist system, a new update rule is adopted. In stochastic relaxation, there are only two possible output values for the units: 0 and 1. This is called the *activation* of the unit, and is labelled a_i for proposition A_i. This is the current most probable truth value of the proposition. There is still a probability, p_i, which is used to set the node on or off. With deterministic relaxation, p_i was the probability represented by the unit, as well as its value. Now, p_i is the *probability* of setting the node's actual value, a_i, to 1. The new update rule is: $p_i = \frac{1}{1+e^{-\Delta E_i / T}}$. This rule gives an exponential probability distribution. ΔE_i is the change in total energy that would occur if a_i were turned on (as opposed to E_s, which is the total energy of an entire system configuration, s). Figure 2.4 graphs the probability of updating a node against the change in energy. Note that at freezing temperature, $T=0$, the update rule is a single step and equivalent to the earlier deterministic update rule.

Figure 2.4 The stochastic update rule.

The mathematical relationship between the energy E_s and the entropic $H(x)$, from Section 2.5, is straightforward:

$$H(x) = -\ln P_s$$
$$P_s = e^{-H(x)} = \frac{1}{Z} e^{-E_s/T}, Z = \sum_s e^{-E_s/T}$$
$$H(x) = E_s/T$$

This result corresponds to the physical definition of entropy, and is comparable to the thermodynamic heat equation, where $H(x)$ is the change in entropy, E_s is the heat input and T is the thermal temperature. Physicists use this formula to study the gross, macroscopic properties of a system, which arise from the lower-level information-theoretic principles of Section 2.5. It describes the over-all statistical behaviour of many interacting components, and is known as the *Boltzmann* distribution (see Figure 2.5). This model is only a highly idealized abstraction of how randomness might be used in the brain. The effectiveness of noise in models more closely related to neurophysiology has also been studied [Bu887]. Other similar models that employ noise include Harmony theory [Smo86] and the Boltzmann Machine [HSS86], both of which employ versions of simulated annealing. The motivation behind Smolensky's theory is to connect the higher level descriptions of symbolic schema with the lower level sub-cognitive structures. The Boltzmann machine separates the data learning rule into a two-stage process similar to some theories of dream function [Cri883]. There are also models that employ randomness in a more heuristic fashion. Genetic algorithms [Hol87] use such techniques in a non-neural frame-work.

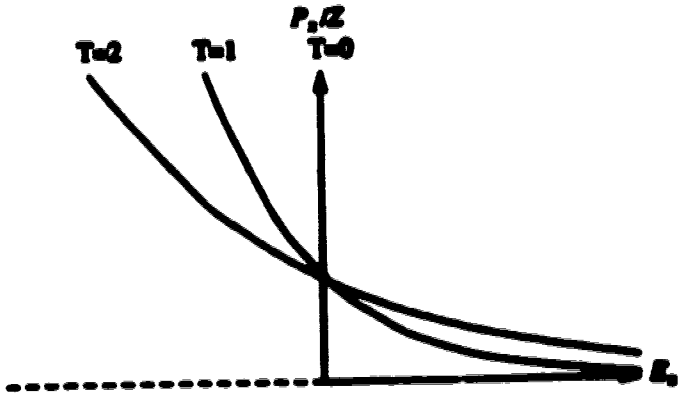


Figure 2.5 The Boltzmann distribution. As $T \to 0$, the system freezes into its minimal energy state.

Geman and Geman [GeG84] formalized limits on the *annealing schedule*, which is the procedure by which the temperature is lowered. Many different schedules are possible. For instance, what temperature do you start with? How many iterations do you let run before dropping the temperature, and then by how much? Sometimes an ad hoc approach is followed, where the schedule is tuned to whatever works for a particular problem (e.g. [HiSS85]). The schedule of Geman-Geman has an initial temperature proportional to the size of the system (number of units). Subsequent temperatures follow the schedule $T(k) \geq \frac{c}{\log(1+k)}$, where T(k) is the temperature at the kth iteration, and c is a constant. This schedule is said to be optimal, but only in the limit of many trials. Within one annealing, even this schedule can miss certain types of discontinuous and oddly shaped landscapes. Nonetheless, it is the closest one is likely to get to a guarantee of finding the true minimum. Unfortunately, this logarithmic schedule lowers the temperature too slowly, and a more realistic schedule must be used in practice. Most schedules retain the essential characteristic of Geman-Geman: lowering the temperature quickly at first, and then more slowly as equilibrium is approached. Perhaps most popular is the Kirkpatrick scheme [KiVe83], an exponential schedule which will be described in detail in Chapter 4.

The information-theoretic formulation of relaxation, outlined here, puts the subject on a firm theoretical footing, and could provide a powerful foundation on which to build models of vision and the mind. Much more work is required to fit this theory with real-world problems. The relaxation machine developed in Chapters 4 and 5 uses stochastic relaxation to match relational structures. This will involve mainly fixed, stochastic relaxation, with some preliminary work on adaptive relaxation, using relaxation control to allow one relaxation network to control another. No internal representations are learned, so the ideas of hidden units and back-propagation are not used, although these concepts may be important to future extensions.

2.12. THE COMPUTATIONAL COMPLEXITY OF RELAXATION SYSTEMS

2.12.1. Ideal Perceptrons

In their landmark 1969 book *Perceptrons* [MIPS69], Marvin Minsky and Seymour Papert showed that there are some problems, such as *connectivity* and *parity* that, while easily solved by serial algorithms, are inherently intractable in parallel. Some have interpreted this result as meaningless in the face of modern connectionist advances, others still feel it represents the inherent inability of such systems to do anything useful. This section will attempt to show why both views are myth.

Minsky and Papert call their idealized *perceptron* Ψ (see Figure 2.6). It is single-layered in that there is only one parallel linear combination of weights. However, this combination can be of functions of the input space, rather than just single input points. This gives the system two layers, not one, but there is still only a single linear combination. Minsky-Papert proved that the input functions, ϕ_i, can be restricted to *conjunctive masks*, which just AND various input points. A system in which each ϕ function relies on only a small, fixed number of input points is the most *local* type of perceptron. The most global kind would have each ϕ function depend on *all* the input points.

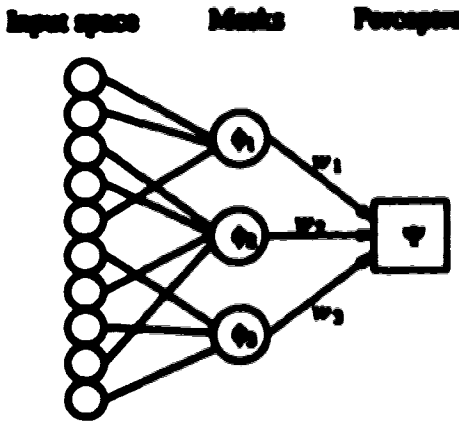


Figure 2.6 The ideal perceptron.

Minsky-Papert use a learning technique, similar to the learning rules already discussed, to adapt the weights in the perceptron. They proved that such a learning rule was guaranteed to find the correct solution, provided such a solution exists. In other words, for simple systems with one layer and no feedback, the landscape is a single, smooth hill, without any local maxima. Unfortunately, experience has shown that systems of even a little more complexity begin to exhibit local maxima (foothills), and so the Minsky-Papert proof does not apply. The importance of the ideal perceptron is that it allows an analysis of the computational complexity of *completely parallel* computations. Most parallel systems, in practice, have feedback and multiple layers, but these can be understood as a kind of seriality. Only the ideal perceptron gives a theoretical tool for studying pure parallelism, which is the simpler case that must be understood before moving up to hybrid systems. In a multi-layered or feedback system, each linear combination that is performed at each location can be understood, in some sense, as an ideal perceptron.

2.12.2. Parallel Complexity

Purely serial algorithms come under the category of classical computational complexity, and it will be assumed the reader is reasonably familiar with such concepts (a brief summary will be given in Chapter 3). Purely parallel systems are much less well understood, and the following will concentrate on aspects relevant to relaxation. The following notations will be used for computational complexity:

COMPUTATIONAL BOUNDS:
Upper bound on complexity: $O(\ )$.
Lower bound on complexity: $\Omega(\ )$.

COMPUTATIONAL COMPLEXITIES:
Serial computations:
$T_P(n)$: units of time to compute algorithm P for size n.
$S_P(n)$: units of storage to compute algorithm P for size n.
Parallel computations: T_P, S_P, in addition to:
$C_P^k(n)$: number of connections to processor k (algorithm P, size n).
$L_P^k(n)$: maximum number of input pulses accepted by inputs to processor k (algorithm P, size n).
$N_P(n)$: number of parallel processors (algorithm P, size n).

The most basic concept of parallel complexity is the time speed-up compared to a serial algorithm,

which is just the ratio of the time for the serial algorithm to the time for the parallel algorithm: $T_{serial}/T_{parallel}$. If the serial algorithm is just a sequential execution of the parallel algorithm, this speed-up is equal to the number of processors, $N_{parallel}$.

The upper and lower complexity bounds, $\Omega(\ )$ and $O(\ )$, are standard in complexity theory, and refer to rates of growth, not to exact complexities. $O(n^2)$ means the complexity grows, at worst, with the square of the problem size. The exact complexity could be much more complicated. Following are some examples of the basic types of worst-case growths:

$T_\Psi(n) = 2^n + n^6 + 1024 = O(2^n)$, *exponential complexity*
$T_\Psi(n) = 5n^2 + 12 = O(n^2)$, *polynomial complexity*
$T_\Psi(n) = 7n + 16 = O(n)$, *linear complexity*
$T_\Psi(n) = 112 = O(1)$, *constant complexity*

C_Ψ^j will be called the *local connectivity* of a processor, or unit. This is the number of input lines to that processor. For Ψ, in Figure 2.6, the connectivity is $C_\Psi=3$. For connectionist systems, the *total connectivity* of the system is generally on the same order as the *total storage*,[8] so S_Ψ will be used to mean either. The total connectivity is just the sum of local connectivities: $C_\Psi^{tot} = S_\Psi = \sum_j C_\Psi^j$. This is a measure of the amount of hardware required to solve a problem.

L_Ψ^j is the *worst-case input connectivity*, or *locality*, of processor k. This is the most number of points in the input-space that any one of the inputs to processor k depends on. In an ideal perceptron, this is just the unit with the most input lines (in Figure 2.6, $L^\Psi=4$). For multi-layer (or feedback) systems, however, the input space could be many layers (or time-steps) away from the processor in question. Although L could be defined in terms of the original input-space, it will be used here only in reference to ideal perceptrons. To apply this concept of locality to a multi-layered, feedback system, the processor in question must first be converted into an ideal perceptron, by considering only the connectivity of the processor, along with the connectivities of the units feeding into it (these units will always be convertible to masks, within this

[8] This is because, in a relaxation system, the memory storage for each processor is almost entirely devoted to the connectivity of the unit. In other types of parallel systems, one might count another term for connectivity, say C_Ψ^{mem}.

limited context). Locality, then, refers only to what can be computed at each parallel step.

Minsky and Papert call $L(n)$ the *order* of the perceptron, and it was their main theoretical tool. For instance, they showed that to solve the parity problem, at least one unit in an ideal perceptron had to rely on *all* the input points: $L_{parity}(n) = n$. Since the locality of this perceptron grows with the problem size, $O(n)$, parity is considered a global problem. If, on the other hand, $L(n) = c$, where c is a constant, then the problem is said to be *local of order c*, or $O(1)$. Such a perceptron is of *finite* or *constant order*, since $L(n)$ is not dependent on the problem size n.

It is important not to confuse $C(n)$ and $L(n)$. Locality depends on the latter, not the former. In Figure 2.6, even though $C^{\Psi}=3$ and $L^{\Psi}=4$, these are not true complexities because they are not expressed as functions of n, which in this case is 16. It is impossible to say, just from the diagram, how these complexities will change as the input space grows. Even if C^{Ψ} grows with n, which it probably will, Ψ is still a local perceptron, as long as $L^{\Psi}(n)=O(1)$, which means it will stay at 4 no matter how large the problem gets. So locality of a perceptron does *not* demand a constant hardware complexity, only a constant local dependence on the input space.

2.12.3. The Minsky-Papert Theorems

What Minsky and Papert actually showed for the problems of connectivity and parity was that they were not of constant order for an ideal perceptron. Since these problems are very easily solved by serial methods, the conclusion generally drawn is that they are not naturally local problems and should not be studied as such.

The Parity Theorem:
If a problem P can be reduced to *parity*, then an ideal perceptron Ψ_P that solves P cannot be of constant order (i.e. L_P is a function of the problem size).

The Topology Theorem:
All topological predicates, except for functions of the *Euler number*, can be reduced to *parity*.

These theorems show that any problem that amounts to determining whether a certain number of

items is odd or even (parity) is not of constant order, and therefore non-local. Furthermore, all but an express subset of *topological* problems fall into this unusually non-parallel category. To understand topology, the study of connectedness, imagine that all the world's objects are made of Play-Dough. You can shape them and deform them into whatever shapes you please, but you cannot break a piece apart or stick two pieces together. Thus, a sphere and a football are topologically identical, as one can be re-shaped into the other; both are *simply-connected*, with no holes. A coffee cup and a dough-nut are also identical, as they both have a single hole. Images can be distinguished topologically (within a particular dimensionality) by the number of holes, call it h, and the number of single, connected objects, call it c. Both c and h are topological properties. Now define two ideal perceptrons:

$\Psi_{c=1}$: determines if object is singly-connected.
$\Psi_{h=0}$: determines if object is simply-connected.

According to the Topology Theorem, $\Psi_{h=0}$ and $\Psi_{c=1}$ are not of constant order. In fact, the nonlinite order of $\Psi_{c=1}$, the connectivity problem, was one of the major results of Minsky and Papert. The interesting thing here is that if one of these two properties is given, computing the other *is* of constant order. For instance, $\Psi_{c=1 \mid h=0}$ determines connectivity, given that the object has no holes. This problem is a function of the *Euler number*, e, which is the number of connected components minus the number of holes: $e = c - h$. The Topology Theorem states that *only* such Euler-number topological predicates can be of constant order; all others are non-finite.

These theorems use the concept of *problem reduction*, common in complexity theory, which states that if one problem P_A can be reduced to another problem P_B ($P_A \propto P_B$), then P_A is at least as complex as P_B. Minsky-Papert show that ($\Psi_{\text{connectivity}} \propto \Psi_{\text{parity}}$) by re-expressing the connectivity problem as a switching network and reducing the task of determining connectivity to the problem of finding out whether an odd or even number of switches are in the on position.

The idea that connectivity is nonlinite is quite intuitive. For a simple object, it can be easy to just "see" that it is connected. But for more difficult images, such as an object with holes, it can be sometimes

next to impossible to tell, just by looking, that the object is continuous. Take a look at Figure 2.7, and try to decide which of the two objects is completely connected and which is not. You will probably need to resort to some kind of serial method, such as tracing through the figures with your finger, to decide which is which. This diagram makes it clear that even humans cannot, in general, determine connectivity through perceptual means. Thus, it is hardly discouraging if our connection machines also fail in this regard.

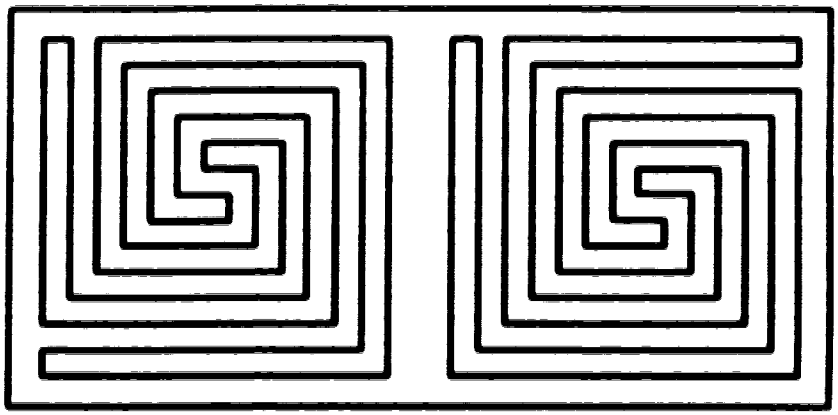


Figure 2.7 The connectivity problem.

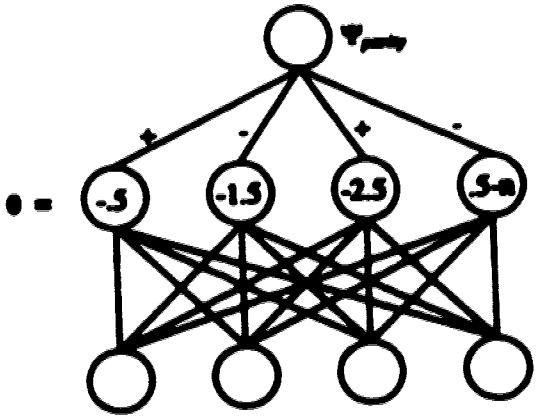
Minsky and Papert reduced connectivity to the parity problem, which is essentially a counting process, the very epitome of seriality. Counting is obviously non-finite for humans, as anyone knows who has tried to count more than about 5 or 6 objects just by looking at them. Perceptrons do not fare as badly in general geometry as they do in topology, at least not at first glance. There are many geometrical predicates that are reasonably testable. For instance, $L_{convex}(x)=O(1)$, so determining whether an object is convex is local of order 1, and a naturally parallel problem. Recognition invariant under *translations* is also finite, while recognition under *rotations* is not. However, Minsky and Papert also show that the constant order of $\Psi_{translation}$ comes with a catch, because it relies on a process called *normalization*. This oversteps the limitation on the perceptron's order by, roughly speaking, exposing a serial process in parallel. Another way of looking at it is that it packs a huge amount of information into each of the weights. The assump-

able growth comes from the information content of the weights (their size or precision), rather than the connectivity. Such a network works by having the correct result cancel out all the others, effectively performing an enumeration of all the possibilities. In coming chapters, this type of complexity will be included as one of connectivity, even though it is the storage requirement of the connections that grows, rather than the number of such connections.

Another interesting twist to translation invariance is that the constant color disappears when the object is to be recognized *in context* (with other objects or noise). Here, even stratification cannot work. That a possible solution, albeit an impractical one, should disappear when context is added illustrates the distinction between exact-match and best-match problems. When the exact match is required, various tricks (like stratification) can eliminate all the impossible choices. When we are looking instead for the best possible match, there is no way around searching a large area of the state-space. However, context and noise do not seem difficult for humans. This is partly because humans rarely need to find an exact *or* best match. They settle for a good match. Humans are not stopped by the parity counting limitation because we do not care about the exact count; what we do is more like a ball-park estimate, a type of "counting" completely unrelated to parity. Humans recognize general forms and shapes that distinguish classes of objects. This means that the stratification and finiteness limitations do not strictly apply, although they are bound to be of importance. It is sometimes suggested that many situation systems are mostly highly parallel exhaustive searches and stratification. Toy worlds are subject to this problem, and it should not be assumed they will scale up to the real world unless it can (at least) be shown that there is either more than just a stratified "look-up table" at work, or that the size of the problem will stay within the "toy" domain.

Such guarantees are often possible. For instance, the parity problem for two inputs is known as XOR (exclusive-or). A small XOR net, with only one or two hidden units, is often necessary as a sub-net of a larger system. Learning the weights for such a network, as well as computing the result of the XOR, is quite tractable, even though the system is not of constant order. [RHW86] demonstrates a parity network of size four, with weights learned via a back-propagation learning rule (see Figure 2.8). This system has four

input nodes, one output node, and four hidden units, so it is still small enough to be practical. The intractability sets in as the problem is scaled up to larger and larger sizes.



$C^{serial}_{parity}(n) = n^2+n = O(n^2)$
$N_{parity}(n) = 2n+1 = O(n)$
$L_{parity}(n) = n = O(n)$

Figure 2.8 A parity network for $n=4$.

This network has nonconstant connectivity and locality, which indicates that the problem involves a substantial global element. In this case, each layer is completely connected with the next layer. Completely connected networks are very common in connectionism, despite their nonlocal nature. This is a mistake only if it is assumed the system will scale up to larger sizes. Minsky and Papert were considering systems where each perceptron received input from every pixel in an image. Such problems are much too large to justify such nonlinear complexities. Problems which can justifiably be restricted to smaller sizes may be able to handle this kind of connectivity, especially if the complexity remains polynomial (or better). Keep in mind that the human brain can support many thousands of connections per processor. Exponential connectivity, however, would be beyond even the scope of the brain for all but the smallest problems. The parity problem mentioned above, while it has polynomial complexity, also suffers from the additional problem of combination. Notice that it solves parity by discounting the weights positive and negative, while the three

holds become more and more negative. Thus, for larger problems, the thresholds will require more and more precision. This is an example of stratification, and like exponential connectivity, it is beyond the capacity of the brain, since neural thresholds and synaptic strengths are currently thought to be quite imprecise. Stratification is essentially just a look-up table of all possibilities, and is a major limitation for multi-layered networks with hidden units.

Part of the reason human perceptual abilities are so good is also due to a certain degree of serially. A single parallel step is not enough; Feldman argues that we need more like 100. This is still extremely small, however, compared to most purely serial models, so ideal perceptrons have much to tell us. In the real-world, however, we will need to split a problem up into at least a number of individual perceptrons. Hidden layers and back-propagation may provide the means to capture higher-order structure in the environment, but it is very important to make the complexity explicit and to watch out for traps like stratification. The lesson to be learned from Minsky and Papert is not to abandon connectionism, but to understand a problem before attempting to solve it, so that limitations on the relaxation procedure will be recognized for what they are. Some limitations will be similar to human-style limits, others will be intractable, involving unacceptable growth.

Chapter 3
Matching In Object Recognition

3.1. THE OBJECT RECOGNITION PROBLEM

The object recognition problem can be conveniently divided into four major processing stages:

I. *Segment* the raw image into various regions.
II. *Extract* various features from the image (e.g. shape and topology).
III. *Match* the resulting representation against an internal model.
IV. *Recognize* the result as an instance of a class of world objects.

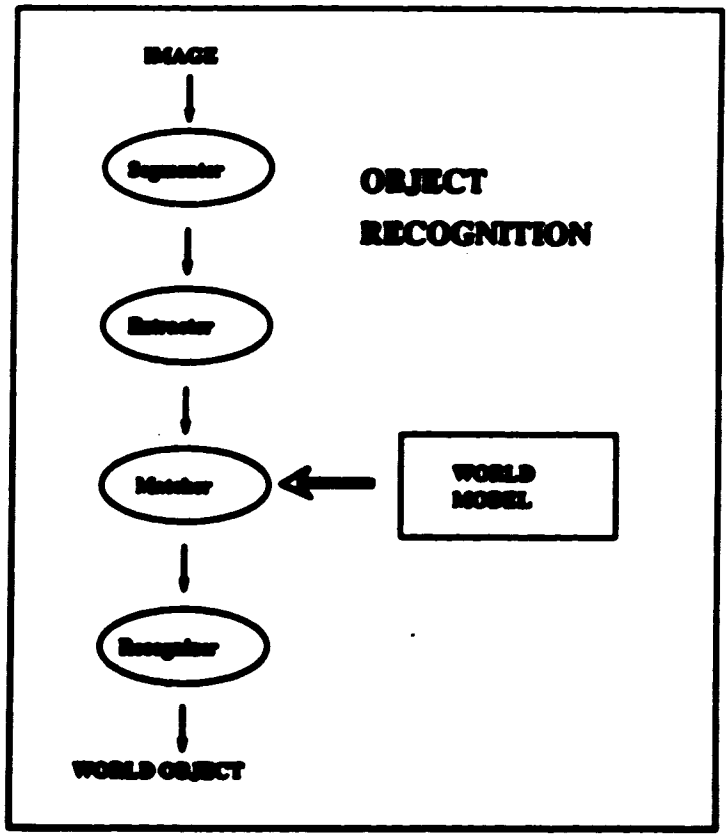


Figure 3.1 Object recognition.

Stage III, the correspondence problem, is the concentration of the current system, to be developed in detail in Chapter 4. A rudimentary stage IV has also been implemented, for recognition. It is assumed that Stage I has already been performed. Stage II is implemented as a set of relational graphs, extracted using traditional, serial techniques. It is also assumed that stages I and II can be done prior to, and independent of, stages III and IV. Since the segmentation stage is already done, the raw data consists of drawings with pixels already hand-classified into regions. Thus, this work could be viewed as dealing with the more abstract problem of matching relational structures rather than visual object recognition. As far as the relaxation part of the system is concerned, the raw data is semantic networks, not visual images. The system matches parts and their relationships, not pixels and gray-levels. However, the graphs are meant to represent images, and the relations are things like ABOVE and ADJACENT. However, unlike real-world images, these are two-dimensional. This choice was made to facilitate the construction of the hand-drawn data images. The system is intended to be part of a larger scale visual system, and no radical changes need be made to accommodate real-world, three-dimensional images.

The above five steps represent one phase of a complete system: the *recognition phase*. The other mode of operation is the *learning phase*, during which the system builds a world model from training examples. A rudimentary adaptive model has also been implemented. Currently, the system has only been used to match input against single-object models, like arches and cups. It is an important area of future work to combine various different objects in the world model. A major selling point of connectionist systems has been *distributed representations*, where the internal model contains many concepts distributed over the same relaxation units.

Figure 3.2 shows an example of an image of an arch. This is hand-drawn and each pixel is labelled as part of one of four regions (fig. a). The two features CONN (connected) and ABOVE are extracted as semantic networks, or relational graphs (fig. b), which are represented with adjacency matrices (fig. c). For instance, the first 1 value in the ABOVE matrix means that part A (the top of the arch) is always above part B (the left support). The 0.5 value in the same row, however, means that part A is above D as often as D is

above A, and so the relation is completely uncertain.[9] The 0.5 values represent completely uncertain arcs. For instance, the first 0.5 value in the ABOVE matrix means that part A will be above part B about half of the arches. Although only two features are illustrated in Figure 3.2, the current system uses fourteen, for a reasonably complete representation of an object.

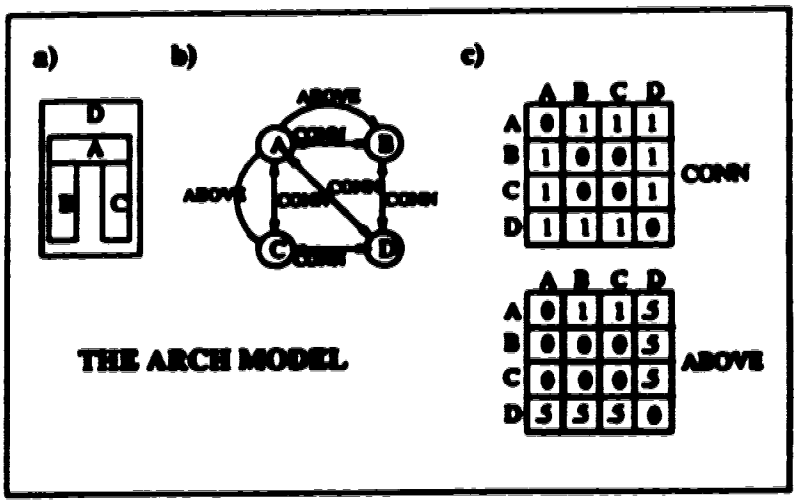


Figure 3.2 Representing an image as a semantic net.

The problem is to find a one-to-one mapping from the parts in such a network (the *model*) and a subset of the parts in another graph (the *data*). This is an instance of the *subgraph isomorphism* problem. If the model has *n* parts and the larger data image has *m* parts, there will be one *mxn* and one *nxm* matrix for each feature. These values represent the *intra-correspondence* between parts within the data image and the model. A third matrix with *mxn* elements represents the *inter-correspondence* between parts in the data image and parts in the model (see Figure 3.3).

[9] This particular example is meant only to illustrate the representation of objects with relational graphs, and was not actually produced by any of the programs described in Chapters 4 and 5.

3.2. ALGORITHMIC COMPLEXITY

It will be assumed the reader is generally familiar with classical complexity theory, but for those who are not, a brief introduction follows. The reader is referred to [Gal79, Hu87B] for a proper introduction to the subject. A problem whose time complexity grows polynomially with the problem size is said to be in P. Most problems that grow exponentially, like subgraph isomorphism, are in NP, which means they can be solved in polynomial time on a certain type of *non-deterministic* machine that is equipped with an *oracle*, a fictitious type of theoretical computer that can automatically decide the correct choice from a list of possibilities.[10]

A regular von Neumann, serial machine may or may not require exponential time for such a problem, but a non-deterministic machine with an oracle can always do it in polynomial time. A subclass of NP, the *NP-complete* problems, have no known polynomial time solutions for normal, deterministic machines. They are thought to be intractable, although if a polynomial solution could be found for any one of them, this would mean that P=NP, and the solution could be applied to all the others, as well. However, it is generally suspected that this is not the case (although, as yet, no proof exists).

All known solutions for NP-complete problems are exponential for deterministic machines. Although they can still be solved in polynomial time with an oracle, this is small comfort, since such machines do not exist. One way of viewing such non-deterministic computations assumes an infinite supply of processors, or *unlimited parallelism*. Thus, any NP-complete problem can be solved in polynomial time with unlimited parallelism. However, these problems are still intractable, because now it is the number of processors that grows exponentially, instead of the time.

---

[10] Most computers in practice are deterministic and equivalent to another kind of theoretical computer, the Turing machine. Several proposed non-deterministic computers, while beyond current technology, are possible in theory, and are equivalent to *limited-resource* Turing machines [Fre82]. Although they are capable of some non-Turing-computable computations, they are not powerful enough to act as oracles. An oracle is basically a Turing machine whose state can, in general, be completely known by the user, something forbidden in the real world by Heisenberg's Uncertainty Principle.

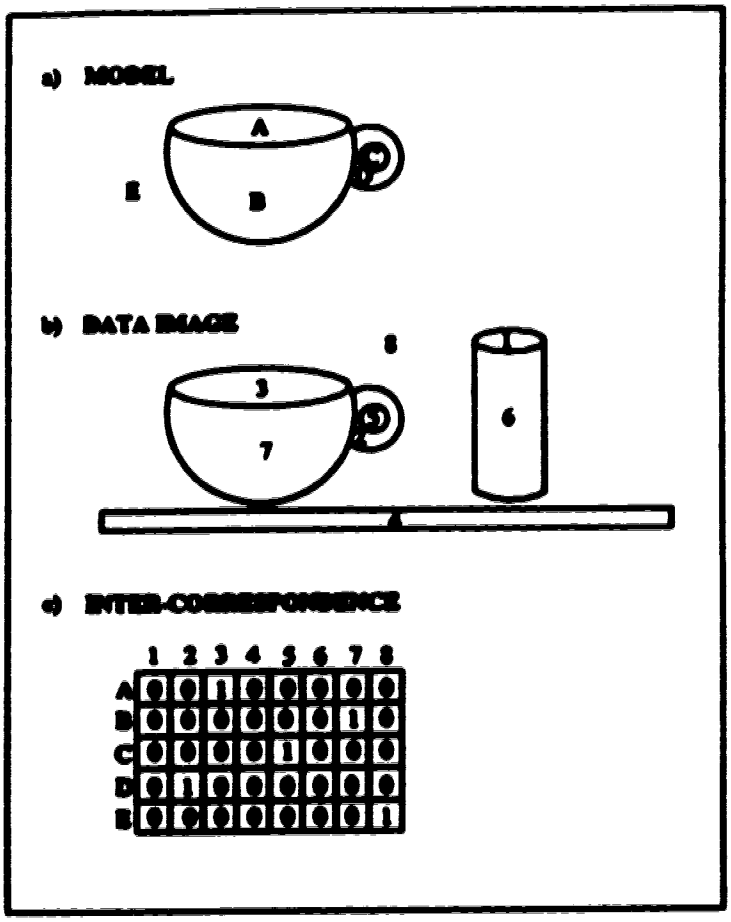


Figure 3.3 Inter-correspondence--matching a cup.

Fortunately, there are ways of dealing with many NP-complete problems, especially if one is willing to accept a merely "good" answer instead of the "best" one. *Heuristics* can be used to cut off certain branches of the search tree that look unpromising. Many problems can be divided into appropriate sub-problems. Using such tricks, of which relaxation is but one, some problems can be made tractable. NP-completeness means that the worst-case complexity grows exponentially. For the *average* case, or for certain classes of inputs, some NP-complete algorithms can run in polynomial time for most cases (often without relaxing the optimality criterion).

3.3. SUBGRAPH ISOMORPHISM

Subgraph isomorphism is a prototypical *NP*-complete problem [Gar79], and thus has wide applicability. It is the general problem of finding a one-to-one correspondence between the nodes of one graph and the nodes of a possibly larger graph. Matching in object recognition is one such problem, where each part of the *n*-part model must be given a correspondence with precisely one part of the *m*-part data image. The problem of finding the best matching, or labelling, of these variables is thought to grow exponentially. However, this NP-completeness may not extend to specific instances, if there is a way to constrain the problem.

While subgraph isomorphism is NP-complete, in general, the special case of graph isomorphism ($m=n$) has not been shown to be NP-complete, even though the search space grows exponentially (the question is still open). For random graphs, graph isomorphism has been solved in $O(n^2)$ [Cor78]. Ullmann [Ull76] solves it in $O(n^3)$, but has the advantage of parallelism and being extensible to subgraph isomorphism, although for this it quickly becomes intractable (Ullmann reports unacceptable growth at n=10, m=15). The problem with these empirical complexities is that they are for *random* graphs. Even a poor algorithm can do reasonably well on such graphs, as opposed to the highly structured graphs found in real world applications. The true complexity depends on the application. Even within one application, the search time can vary greatly from one instance to the next.

Assume all graphs are represented as adjacency matrices. A(i) is the ith node of graph A. A_{ij} is the arc from node A(i) to node A(j). If there are two graphs, M and D, the subgraph isomorphism problem asks for a one-to-one mapping from the smaller graph to a subset of the larger one. Figure 3.4 shows a simple problem of two graphs to be matched and the tree of all possible mappings that must be searched to find the answer. The number of mappings is equal to the number of leaf nodes on the tree. This is $O(\frac{m!}{(m-n)!})$, where $n < m$, which is exponentially large. When the data is the same size as the model ($n=m$), the problem is graph isomorphism and the search space is $O(n!)$, which is also exponential.

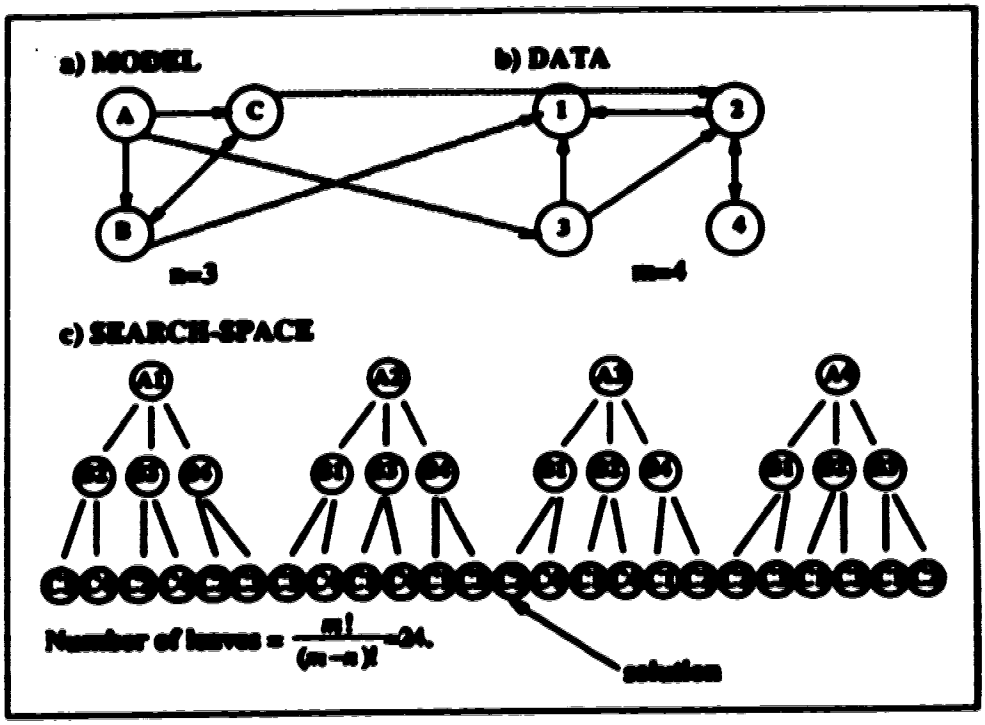


Figure 3.1 Subgraph isomorphism.

3.4. CONSISTENCY AND MATCHING

3.4.1. Consistent Labelling Problems

Matching can be described more generally as a *consistent labelling* or *constraint satisfaction problem* [Hal79, Mal80]. Assume there is a set of n variables, $\{x_1, \ldots, x_n\}$, each of which can be assigned a value from a domain of m labels, $\{l_1, \ldots, l_m\}$. There is also a set of r constraints, $\{R_1, \ldots, R_r\}$, which the labellings must satisfy. The problem is to find a consistent labelling that satisfies the set of constraints.

The constraints might have the form $R_k(x_i)$, which means that the kth constraint restricts only the labelling of the ith variable. This is a first-order *unary* constraint because it involves only a single label. A labelling of all variables that satisfies all such first-order constraints is said to be 1-consistent, or *node-*

consistent. The constraints, of course, can take into account any number of variables. The constraint $R_2(x_i, x_j)$ takes into account the simultaneous labelling of two variables, and labellings consistent with all such *binary* constraints are 2-consistent, or *arc-consistent*. Constraints can be of any order up to and including the number of variables, n. So a labelling consistent with all k-*ary* constraints of the form $R_k(x_1, ..., x_k, k \leq n)$ is said to be k-*consistent*.

This formulation can be applied to a large variety of different problems, such as map-colouring, magic squares, the n-queens puzzle, Boolean satisfiability, scene labelling (matching), theorem proving, database retrieval, edge detection, as well as all manner of graph matching problems [Hal80, Mac87]. There are three basic approaches to such problems: *search techniques*, *consistency techniques* and *relaxation techniques*. Rather than view each as a separate approach, I will consider each method to be a specialization of the previous one.[11] Thus, relaxation can be thought of as a type of consistency algorithm, which can be thought as a type of search. Search techniques that do not fall into the consistency camp include the sub-graph isomorphism searches discussed in the previous section. Most such techniques are some form of *hunt-and-peck*, where the system enumerates the search tree, cutting off any branches that give inconsistent partial solutions. These differ from the consistency methods because they rely on enumerating probable solutions, rather than eliminating improbable ones.

Mackworth [Mac77, Mac85] has developed the consistency notion as a search pre-processor, to eliminate many of the impossibilities before going into a traditional tree-search. His AC-3 arc consistency algorithm is an example of a non-relaxation consistency method (sometimes called a *serial relaxation* algorithm). It works much like discrete relaxation, but has time dependencies that cannot be parallelized. His AC-1 arc consistency algorithm, on the other hand, is a more general form of discrete relaxation. Thus, it is similar to the probabilistic relaxation methods in Chapter 2, except that, instead of updating labels (i.e. activation values) to be more consistent, inconsistent labels are removed from a finite domain list. The

[11] This is only one way to divide the field, as there are many techniques that combine these three ideas and do not fall easily into a single category.

removal of these inconsistencies, as in probabilistic relaxation, causes further inconsistencies to develop, which must be taken care of in the next iteration. This continues until there are no more inconsistencies, and the process stops. A simple version of the pseudo-code follows:

```
Mackworth's Arc Consistency (AC-1) Algorithm:
REPEAT
 FOR (all pairs of nodes with arc (i,j)) DO
  FOR (all possible i=x) DO
   IF (there is no possible j=y consistent with i=x) THEN
    delete x as a possible i
   ENDIF
  ENDFOR
 ENDFOR
UNTIL (no change in possible i's)
```

A similar algorithm is used in Waltz-filtering [Wal75], where the constraints are the compatibilities of various edge types on block-like objects in computer vision.

Another method of satisfying a system of constraints is the popular mathematical technique of *linear programming*. Relaxation and consistency methods can be formulated as a certain type of linear programming problem [Rel82, Hal83].

3.4.2. Matching as Consistent Labelling

If we view subgraph isomorphism as a labelling problem, the variables are the nodes of one graph and the labels are the nodes of a second graph. Any particular instantiation of the variables is a one-to-one mapping from the first graph to the second graph. The constraints to be satisfied are the *arc* consistencies between the two graphs.

Mackworth proved that AC-1 (i.e. discrete relaxation) runs in time linear to the number of binary constraints. This proof is only possible because there are only so many items in the discrete domain list, and once one is deleted, it does not come back. Without noise or uncertainty, every inconsistency can be completely eliminated, without any doubt as to its uniqueness. This is an example of exact-match. With best or good-match problems, discrete relaxation is just not sufficient. Probabilistic relaxation allows conflicting

pieces of evidence to be weighed against each other. This makes it harder to prove things as strongly as for discrete relaxation, although correctness is still guaranteed for systems with simple hills (no hidden units or loops to cause local foothills) [MP69]. The differences are even sharper when we look at stochastic systems, for which it is very hard to prove any of the theorems of deterministic and discrete relaxation. There is a guarantee of correctness only for the theoretically optimal annealing schedule, which is really just a random, global search. So an important question is: which concepts and proofs can be carried over, in spirit if not rigorously, from discrete to probabilistic to stochastic systems?

In the next chapter, an attempt is made to solve the matching problem with relaxation. Relaxation has been used before for matching relational structures [Col187, Coo68, KIR79]. These systems use discrete deterministic relaxation. The current system uses stochastic, probabilistic relaxation and allows noise in the inputs, which is a substantially more difficult problem. With discrete relaxation, possibilities eliminated in the search are known to be inconsistent. Once noise and uncertainty is introduced, inconsistent branches cannot automatically be pruned out of the search tree, as there is never any certainty about whether a given piece of evidence, by itself, is reliable. This harks back to the discussion in Chapter 2 of best-match versus exact-match.

These distinctions become important when examining previous attempts at using relaxation for matching relational graphs. [KIR79] created a very simple relaxation matcher, much like the AC-1 algorithm, that worked in linear time. Cooper's system [Col187, Coo68] is more ambitious, and he proves the correctness of the procedure. This proof works similarly to the AC-1 proofs. His algorithm is written in the style of connectionist systems, so the parallel nature is highly explicit, but the relaxation procedure is still discrete. There are only so many possibilities to eliminate, and each step is a step in the right direction. Although Cooper's system is end-to-end, with real-world data, his inputs are thirty-two figures, and so the numbers and types of parts are limited enough that inconsistencies can be eliminated with certainty. Cooper's system also runs in linear time.

3.6. CONCLUSION

For visual object recognition, the problem is here divided into four sub-stages. The correspondence problem is NP-complete, and it is possible that this inherent NP-completeness does not go away, even when the problem is appropriately constrained and parallelized. Humans, for all their flexibility, cannot recognize objects that contain large numbers of parts. While we may sometimes appear to do so, it is accomplished only by grouping the regions into a small number of larger regions, to be matched against the internal representations.[12] Humans will fail, just as machines, if asked to distinguish between two objects when the distinction relies on dividing the image into hundreds of parts. This phenomenon exists throughout cognitive science. In natural language understanding, for instance, humans quickly fail when given sentences to parse of much complexity. Any version of the connectivity or parity problems discussed in Section 2.10 will have this property. All this points to the idea that humans do not solve large problems, but rather collections of smaller toy problems. Marvin Minsky has called this philosophy the *Society of Mind* [Minsky]. Once a problem is properly constrained to make it tractable, we can go ahead and use weak methods like adaptive systems theory and gradient-descent. But these methods are only likely to work if the problem is small enough and represented in a way that gives a reasonable landscape in which to hill-climb.

---

[12] This problem is bypassed in the current system, since the segmentation phase is done by hand.

Chapter 4
A Relaxation Matcher

4.1. THE MATCHER: Informal Development

This system uses stochastic relaxation techniques to solve the matching problem. It has been assumed that the image has already been segmented into regions and, for now, that various features have been extracted to give a relational graph describing the image. It is also assumed, for now, that an *a priori* relational model of the object to be recognized has been provided. Later, a way of learning this model from training examples will be explored. The system is composed of a data image, an internal model, a matcher and a recognition unit (Figure 4.1). For convenience, in the examples to follow, letters are used to index the parts in the model and numerals to index the regions in the data image.

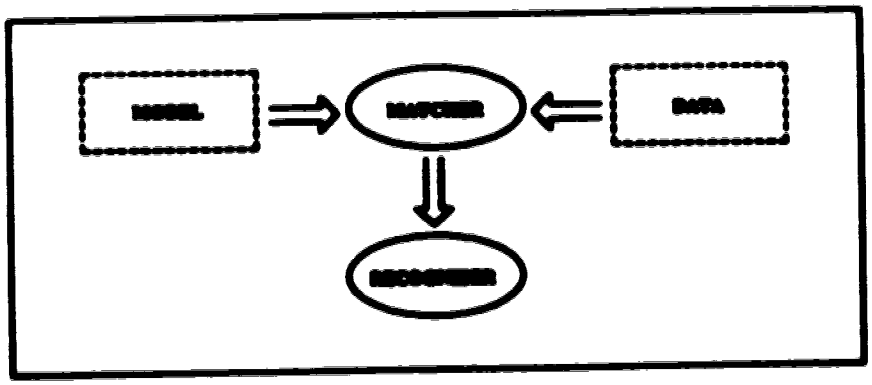


Figure 4.1 System overview.

4.1.1. The Matcher

Assume that M is the internal model of some object in the world, such as an arch, a tea cup, or a chair. D is the (possibly larger) image that must be matched against this model. Both the image and model are represented as relational graphs, or semantic nets. There are several (currently fourteen) *features* that are used to describe the scene. For each feature, f_k, there will be one model graph, M_k, and one data graph, D_k. If the M_k matrices are mm (i.e. there are m model parts), and the D_k matrices are nn (i.e. there are n data parts), then C will be an mn matrix representing the inter-correspondence between the model and data. W_k will be a set of $(mn)^2$ compatibility measures representing the consistencies of pairs of individual correspondences for feature f_k. No one feature is expressive enough by itself to come up with the correct match. The approach here is to combine the W_k matrices into a composite set of weights W^C. These are then used as the weights in relaxing C to get the final solution. The fourteen features are (they are described in more detail later on): *AREA, PERIMETER, RADIUS, COMPACTNESS, ECCENTRICITY, ADJACENT, CONNECTED, BIGGER, CLOSE, ABOVE, RIGHT_OF, INSIDE, CONCAVE* and *SIMILAR*.

If the image from D does indeed contain an example of the object modelled in M, then there should be a clear winner for the correspondence in each row of C (look again at Figure 3.3). If the data image does not contain an example of the modelled object, the system gives the best match it can. If we scan across each row of C, we should find one clear winner if the system succeeded in finding a solution. If the system fails to find a match, there will be no clear winner (later, we will see that this is usually preferable).

The matching is done using stochastic relaxation, so each element of C is a relaxation unit. This network is completely connected, so W^C contains a compatibility measure for each pair of units. For instance, $w_k(A1|B3)$ represents the compatibility, with respect to feature f_k, of M(A) matching with D(1) and M(B) matching D(3). This is scaled between -1 and +1, and is just a difference measure between the two arcs M_{AB} and D_{13}. The closer these two arcs are to matching, the higher the compatibility measure. Inconsistent matches will have negative weight, consistent ones will be positive. The weight used in C, the relaxation network, is $w^C(A1|B3)$, which is a linear combination of all fourteen $w_k(A1|B3)$ values.[8]

[8] For the results in Chapter 5, this amounts to a simple averaging over the fourteen features. Many experiments were also conducted with differing weights given to the various features, but none provided significantly improved results.

This example is illustrated in Figure 4.2, with the circles and solid lines representing the nodes and arcs in the model (M) and data (D). The rectangles represent elements of C, each of which is one possible match between a model part and a data part. The connection between the rectangles, represented as a dashed line, is the compatibility of the two matches from W^C. Only two elements of each matrix are shown in the diagram. The actual number of such elements for each feature is noted at the top. Although w(A1:B2) is determined by AB and 12, it does not change during the relaxation, because AB and 12 are fixed. The weights are determined ahead of time, and set to fixed values. Later, we will explore dynamic models, and will need to account for changing weights. After C is relaxed, it should contain the solution (although we will see later that this is still not quite enough). This solution can be used to align the input image D with the model, producing a new matrix, call it P. This can then be used as further training during the next learning phase.

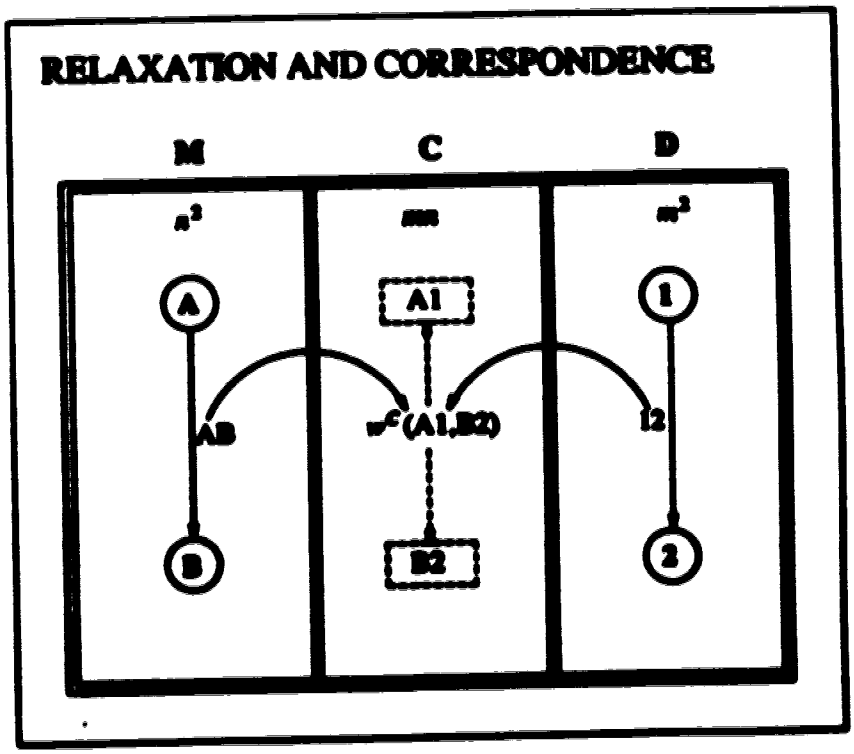


Figure 4.2 Arc consistency.

4.1.2. The Model

So far, a fixed, *a priori* model has been assumed. Although this can be the case if a fixed model is desired, in general M can be adaptive. The simplest way to do this is to create a *first-order* statistical model by averaging the values in the training examples to form the model M. This is simple and fast to compute but has limited application. Most of the results in the next chapter are for such a first-order model, but preliminary tests have been performed on an adaptive, *dynamic second-order* model that interacts with the matcher. Each M_k matrix is completely connected, so all elements are connected to all others by a weight from the $n^2 \times n^2$ weight-space W_k^M. Each weight represents the compatibility of that pair of relational arcs in the model. This network can be relaxed to find the best merging of the training examples, much as C was relaxed to do the matching.

When the model is adaptive, there are two main phases of processing: *learning* and *recognition*. During the learning phase, sample instances of the object being learned are *clamped* onto the units in the model. During learning, the weights in M are adapted according to the *Widrow-Hoff* delta rule, to minimize the sum-squared error between the training examples and the relaxed solution. Whichever type of model is chosen, the result is that various training examples are merged into the most consistent model. If we are modelling arches, we present many training examples of arches, which are combined into the most consistent composite arch. We are now ready for the *recognition* phase.

4.1.3. Annealing the Model

Once the adaptive model is trained, it has an appropriate set of weights that minimizes the sum-squared distance between the various training examples. The weights cannot be used by themselves as a model, of course, since they do not give actual relations between parts. One way to extract an actual model is to simply anneal M, unclamped with no training data, and let it reach an equilibrium. The relaxation mechanism remains the same here as in training, except that there are no training values and no changes at all are made to the weights. Thus, the system is simply performing a relaxation search, with weights acting

as compatibility measures, for the most consistent instantiation of the model. For instance, if the model has been trained with cups, an unclamped annealing will find the most consistent possible cup, according to the weights developed during training. What is produced is a representative cup, with no consideration of any input from the environment. It is analogous to a human recalling from memory his concept of a cup. Because of this, it could be called a "daydreaming" mode (although no rigorous correspondence is claimed with actual human daydreaming).

In experiments, the annealing does settle into a reasonable model without undue relaxation times. This is because relaxation is best at mapping similar patterns onto each other, so not too much is expected of the learning process. Only one class of object is modelled in the network, so there is no undue interference between completely orthogonal inputs. Such interference, however, must eventually be dealt with, although total distribution of all concepts over the same units seems unlikely, as it would quickly become unmanageable. There must be mechanisms for both distribution (merging concepts together) and insulation (keeping concepts apart).

4.1.4. Matcher-model Interaction

Even with only one class of object, though, there is still a need to deal with interference between different training examples. Say, for instance, that the system is learning to recognize cups. Assume for now that all cups have basically the same structure. Some are fatter, some are taller, but they are all basically the same (they all have a handle on the right side, say, and never on the left). All input patterns, then, are linear combinations of each other. So a first-order model should be sufficient. Now consider the cup handle. Suppose that the handle is on the left hand side about half of the time and on the right side of the cup the other half. Hmm, a first-order model is no longer enough. Take, for example, the feature RIGHT_OF. Fifty percent of the time, the handle is to the right of the body of the cup: RIGHT_OF(HANDLE,BODY). The rest of the time, it is to the left: RIGHT_OF(BODY,HANDLE). In a first order model, the system will match this properly perhaps only half the time: RIGHT_OF (HANDLE,BODY) = 0.5, RIGHT_OF(BODY,HANDLE) = 0.5. A cup with two handles or none at all

would be considered just as common as the proper cups. A second-order model, however, can represent this information. The weight between the above two relations, w(RIGHT_OF(HANDLE,BODY), RIGHT_OF(BODY,HANDLE)), will be negative (inhibitory). In addition, the first-order information is still there; w(RIGHT_OF(HANDLE,BODY), RIGHT_OF(HANDLE,BODY)) measures the first-order frequency of the handle on the right side.

Now assume that the model used for matching is the unclamped "daydreaming" model. The model says, since it is second-order, that there must be a handle on one side, but not the other. Both have equal probability. Since the model is stochastic, it will choose one possibility over the other, each with a 50% probability. This is accurate, in that it produces a correct cup, but it is not appropriate in matching a data image, for which the handle is on one specific side. Comparing this to human thought processes, it is like someone asking you to imagine a cup. Pulling out an image from your brain's associative memory banks, you imagine a typical cup. Since half the cups you have seen in your life have handles on the right side, your imaginary cup will have this feature roughly half the time.[14] True recognition requires more than this, however. The matching and modelling phases can only be separated for a first-order model. In the second-order model, there must be interaction between the two subsystems running concurrently. The associative memory function (annealing the model) must interact with the relaxation matching of the data, so that as your imaginary cup gets closer and closer, the matching process gets closer and closer to the correct correspondence. The final cup model should be one that corresponds closely to the input data. This means that, for the above example, the cup model will have its handle on the same side as the data cup.

This kind of interaction is possible in a relaxation matcher, because the matching is a gradual, incremental improvement process. Each step gets closer to the correct result. In a tree-search algorithm, each step is just another possibility, so the model cannot really be given this kind of continuous feedback.

To describe the actual mechanism for this, a new concept must be introduced: *controlled relaxation*.

---

[14] Keep in mind this is a simplified example. There are many more complicated possibilities, especially for third-order models with hidden units.

Rather than using the results of one computation to constrain the other, the two systems can be coupled together at the most basic level of operation, allowing one network's units to *gate or control* the units of the other [CaG87, RMG88]. Consider the model node AB. In the free-running, "daydreaming" mode, AB is connected only to the other nodes in the model. But now we want it also to be affected by the relevant nodes in the matcher and the data image. For instance, if the data node 12 is high, as well as the matcher nodes A1 and B2, then this will contribute to the model node AB. Information must also flow from the model to the matcher, so that the arc consistencies are continually updated to reflect the ever-changing model.

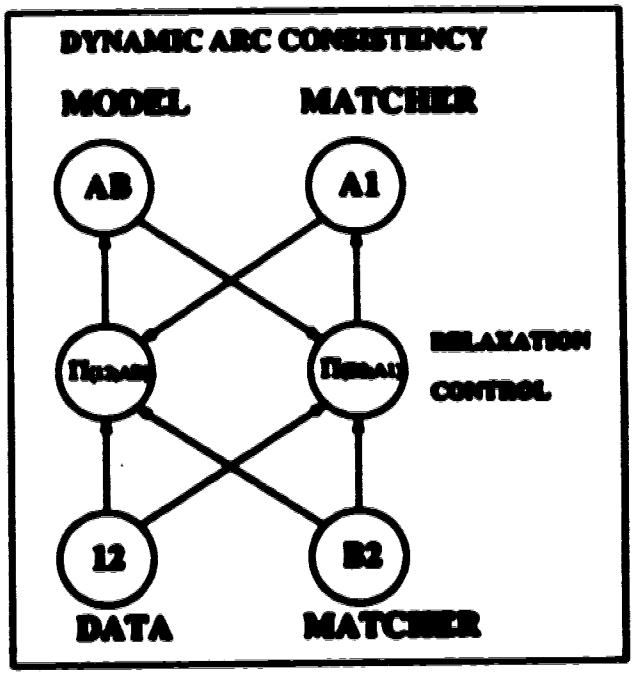


Figure 4.3 Matcher-model interaction.

As shown in Figure 4.3, this can be described as the addition of a second kind of node to the relaxation system. Normal relaxation units are *sigma* units (Σ), which sum their inputs. The *pi* unit (Π) multiplies its inputs, allowing one network to gate or control another. This allows the matcher to gate, or control

the model, and vice-versa. This means that the arc consistency weights in the matcher are now influenced by a constantly changing model, and can no longer be set up beforehand. They must be updated at each step, not via a learning rule, but to reflect the new information in the model. Unit Π(12:AB) in Figure 4.3 can be viewed as a *dynamic weight* between the model and the data, controlled by the matcher (or it could be called a weight between the model and the matcher, controlled by the data). Likewise, another Π unit is placed between A1 and B2.

4.1.5. The Recognizer

The final part of the system is the *recognition unit*, Ψ. With the matching, the system has already done much of the object recognition task. C is used to re-align the relevant regions in D into a new object P. This is a simple re-shuffling of D's elements to reflect its correspondence with M. For example, if C shows that M(1)=D(3), then D(3) becomes the value for P(1). This new *n*-part image is used as input to the recognition unit, whose energy is a measure of the chance that the detected subgraph is an instance of the modelled object. Ψ, is a single-layer perceptron (see Figure 4.4), which is trained via Widrow-Hoff at the same time the internal model is being learned. The recognition unit is completely connected to the units of M. The input value clamped onto Ψ is 1.0, if D is an example of the object modelled in M, and 0.0 if it is a counter-example. Training times are currently very low, as counter-examples are not yet used. Thus, the recognition unit is always clamped on 1.0 during training. If the system is to handle near-misses, counter-examples will have to be added.

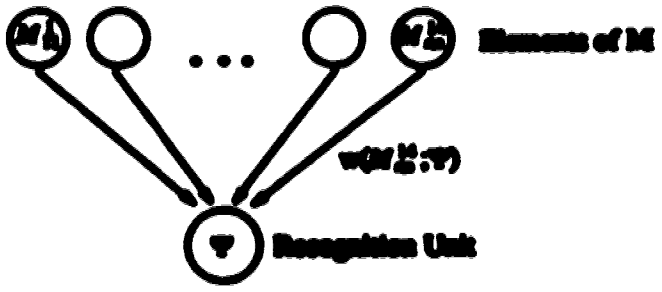


Figure 4.4 The recognition unit.

4.1.6. System Flowchart

Figure 4.5 illustrates the flow of information. To summarize, D is matched with M to produce compatibility measures in W, one for each feature. These are combined into W^C, which interconnects the relaxation network, C. This is relaxed to produce a matching, which can be used with the data, D, to produce a new, properly aligned object P, which can be used as a further training example,[15] and/or presented to Ψ. This value of Ψ is the real end-result. The system will produce some kind of match no matter how poor the data, so Ψ is needed to give a judgement on whether the matched item really is a valid instance of the modelled object, or not.

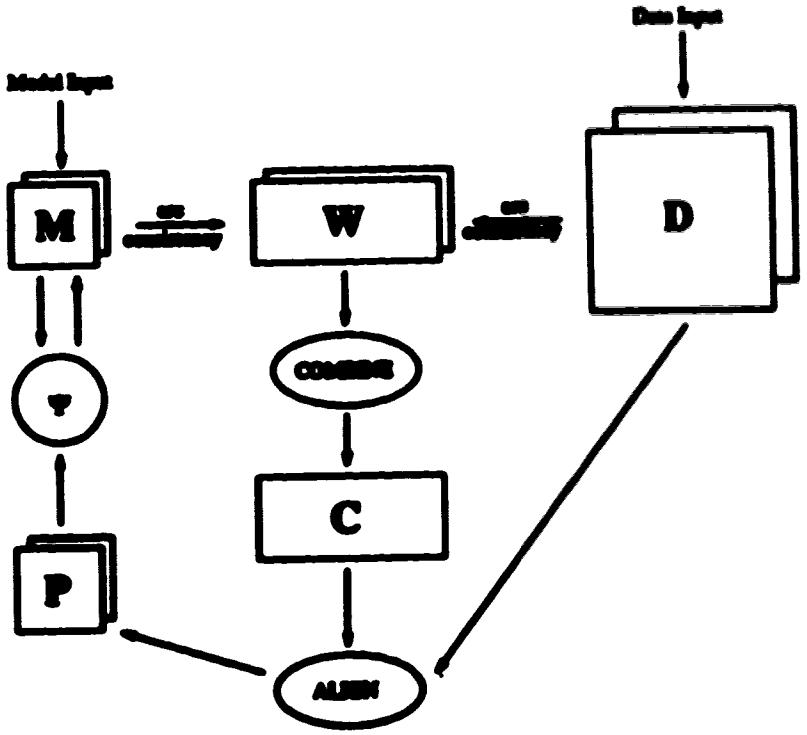


Figure 4.5 System flowchart.

[15] Feedback from the user may be desirable here (i.e. do we clamp P back onto the model unless the system was successful in the recognition phase).

4.2. THE MATCHER: Formal Development

4.2.1. Definitions

The relaxation matcher consists of the following parts:

THE FEATURE SET (F): a set of Q features f_k:
$F = \{f_k, k \le Q\}$.

THE MODEL (M): an internal model of a class of objects, consisting of:
A set of n parts in the model: $\{M(i), i \le n\}$.
Q relational graphs, represented as $n \times n$ adjacency matrices:
$M_k = \{m_{ij}, (i,j \le n), k \le Q\}$.

THE TRAINING EXAMPLES (T): a set of R training examples with which M is trained:
$T = \{T_k, k \le Q\}$.
$T_k = \{d_{ij}(r), (i,j \le n), r \le R\}$.

THE DATA IMAGE (D): an image that must be matched against the model, consisting of:
A set of m regions in the data image: $\{D(i), i \le m\}$.
Q relational graphs, represented as $m \times m$ adjacency matrices:
$D_k = \{d_{ij}, (i,j \le m), k \le Q\}$.

FEATURE WEIGHTS (W): the set consistencies between M and D for each feature, Q $nm \times nm$ matrices:
$W_k = \{w_k(ip,jq), i,p \le n, j,q \le m; k \le Q\}$.

COMPOSITE WEIGHTS (W^C): a linear combination of the feature weights, one $nm \times nm$ weight matrix:
$W^C = \{w^C(ip,jq), i,p \le n; j,q \le m\}$.
$$w^C(ip,jq) = \frac{\sum_k w_k(ip,jq)}{Q}.$$

THE SOLUTION MATRIX (C): the relaxation network in which the matching is performed:
One nm adjacency matrix, completely connected by the weights in W^C:
$C = \{c_{ij}, i \le n, j \le m\}$.
c_{ij} = probability that $M(i) = D(j)$.

4.2.2. Arc Consistency

W^C measures arc consistency between the model and the data. If arcs m^k_{ij} and d^k_{pq} are about the same, then the two matches (M(i)=D(j) and (M(p)=D(q)) are compatible. If the two arcs are not equal, then the matches are incompatible. The measure of compatibility is the difference between the two arcs, scaled between -1 and +1: $w_a(i,j;p,q) = 1-2|m^k_{ij}-d^k_{pq}|$. For example, if $m^k_{ij} = 1$ and $d^k_{pq} = 0$, then the two arcs are totally incompatible and the weight is -1. If, on the other hand, $m^k_{ij} = 0.4$ and $d^k_{pq} = 0.4$, then the two arcs are completely compatible and the weight is +1. If $m^k_{ij} = 0.2$ and $d^k_{pq} = 0.7$, the system remains neutral on their compatibility and the weight is 0.

4.2.3. Incomplete Matchings

There is another source of constraint for the match, owing to the nature of the matching process. A node in one graph cannot match with more than one node in the other graph (at least, we want to discourage this). Such matches must be made incompatible, by assigning negative weights to links connecting two C nodes that are in the same row or column. This says that any two correspondences that match the same model part to two or more data parts (or vice-versa) are incompatible, and do not contribute towards a complete matching of the two graphs. These pairs of correspondences will be called *same-row/column*, or *incomplete match* pairs. For example, c_{A1} and c_{A2} are *same-row* matches, while c_{A1} and c_{B1} are *same-column* matches. These incomplete matches are more than just a technical detail; they are crucial to our understanding of the degradation of relaxation systems in the presence of noise. There are two approaches to using the same-row/column weights: strong inhibition and weak inhibition. A highly negative value can be used to force a single complete match as the result. This forces the system to do a more global search, and is called a WTA, or winner-take-all system, because the strong inhibition between various possibilities means that only one complete matching can win out. A weaker inhibition is commonly more with eliminating inconsistencies, and does not result in one unique match. This follows Mackworth's kind of eliminating the impossible as a preprocessing stage to a traditional search algorithm [Mac77]. The problem of eliminating impossibilities is, like the more general subgraph isomorphism, also NP-complete. Note that

for model-number interaction, even though the fixed weights were replaced with dynamic ones, this WTA aspect of the matcher remains the same.

The current system was found to work well with the following strategy: two inhibition values are used, ξ_{row} for same-row matches and ξ_{col} for same-column matches. Highly negative values produce very few matches per model part and low values produce many matches. This factor can be fine-tuned by keeping ξ_{row} constant at 1.0, and adjusting ξ_{col} to produce an acceptable number of 1's in each row of the solution matrix. The final formula for compatibility functions is:

$$w_2(i,j;p,q) = \begin{cases} 1.0 & \text{if} \quad (i=p) \& (j=q) \\ -\xi_{col} & \text{if} \quad (i=p) \& (j \neq q) \\ 1-2|m_{ij}^k - d_{pq}^k| & \text{otherwise} \end{cases} .$$

To avoid constant tweaking of ξ_{col}, most of the adjustment is automated. The parameter has been hand-tuned for the case of noise, and it was found that the relationship shown in Figure 4.6, between m (number of data parts) and the number of matches per model part, provides a good balance between speed and performance. Too many 1's in each row and the later exhaustive search phase is far too expensive, but too few 1's and the problem of local maxima appears. One possible way around this would be to run the relaxation to a complete solution (pure WTA) many times quickly, taking the result with the highest consistency (lowest energy). This has the added benefit that each relaxation could be done in parallel in a separate C matrix, adding a whole new aspect to the parallel nature of the solution, and avoiding an exhaustive search phase entirely. Unfortunately, it was found that the problem of local maxima is much too acute for this unless the data is quite free of noise and the object typical of its class.[16] We must be content with a reduction of possible matches left in the solution matrix, on which we can perform an exhaustive search, or any kind of heuristic search of our choice.

---

[16] Again, we see that the problem of requiring a hare-search instead of an exhaustive search prevents us from removing all the possibilities.

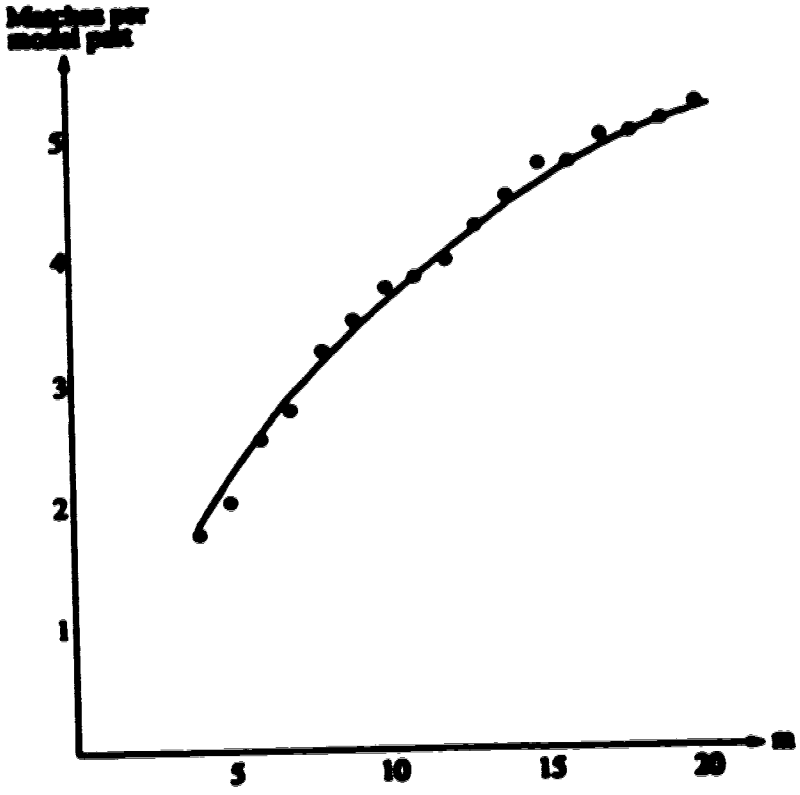


Figure 4.6 A balanced relationship between number of matches remaining and performance.

In the trial runs performed, this behaviour was dramatic. Insisting on complete matches generally required worse than exhaustive search time for all but the cleanest images. A slight loosening of this requirement, as in Figure 4.6, and the mild time complexity drops dramatically. The trade-off in Figure 4.6 was hand-tuned for the case of melon, which provided a stepping stone from which the other models could build their own trade-offs. The system keeps a table, for each model it builds, with a ξ_{min} value for every value of m: $\xi(m)_{min}$. The relationship for melon in Figure 4.6 was used as the initial guess, and $\xi(m)_{min}$ was incremented by some amount $\Delta\xi$ every time a match contained more incomplete matches than desired. The value was decremented by the same amount when there were too few matches. After a suitable period of matching attempts, during which ξ_{min} is adjusted, a curve is fitted through the resulting table of values.[17]

[17] The curve-fitting is not really necessary, but discloses any possibility of unintentionally helpful matches due to trade-space tabling.

4.2.4. Constraint Propagation

Information is propagated according to the standard equations from Chapter 2 for stochastic relaxation, with an additional Π term for the case of dynamic arc consistency:

$$p_i^c(1) = \frac{1}{1+e^{-\Delta E_i^c / T(t)}}.$$
$$\Delta E_i^c = \sum_{j \neq i} c_{ij} w^{cc}(i,j)\, p_j^c(1) - \theta_i^c + \Pi_i^c.$$
$$\Pi_i^c = \sum_{k \neq c} a_k m_{ik}^k c_{ik} d_{ik}^k, \text{ if there is model-matcher interaction (0 otherwise).}$$

$$p_i^m(1) = \frac{1}{1+e^{-\Delta E_i^m / T(t)}}.$$
$$\Delta E_i^m = \sum_{j \neq i} m_{ij}\, w^{mm}(i,j)\, p_j^m(1) - \theta_i^m + \Pi_i^m.$$
$$\Pi_i^m = \sum_{k \neq m} a_k c_{ik} d_{ik}^k c_{ik}, \text{ if there is model-matcher interaction (0 otherwise).}$$

The threshold has been included in these equations, although actual experiments work best with $\theta=0$ for the first-order model. Further experiments will be needed to determine the usefulness of non-zero thresholds for higher-order systems.

4.2.5. The Annealing Schedule

This system uses the exponential Kirkpatrick-Gelatt-Vecchi annealing schedule [KJV83], abbreviated KGV:

$$T_i = T_0 \sigma^i$$
$$r = \frac{T_1}{T_0}.$$

This schedule mimics the cohesion of freezing quickly at first and then slowly near the convergence point (see Figure 4.7). There are three possible values from which KGV can be computed, any two of which will completely determine the annealing schedule:

T_{min}: the equilibrium temperature.
k: number of iterations for equilibrium to occur.
r: the annealing factor, defined above.

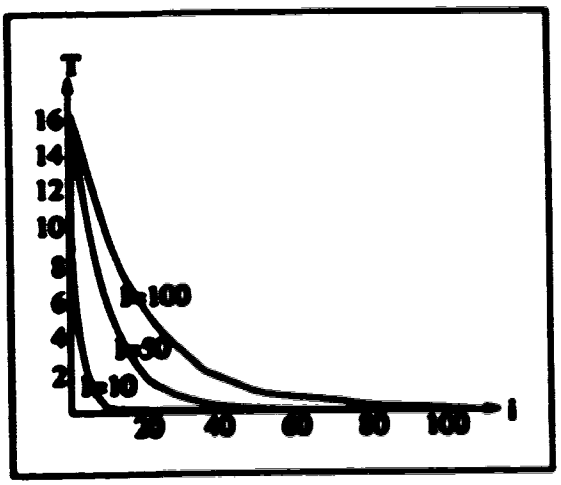


Figure 4.7 The Kirkpatrick-Gelatt-Vecchi annealing schedule.

In actual experiments, the matcher generally locks into an equilibrium state by T_{min}=0.05. For various practical reasons, to be discussed in more detail later, it was found that k=100 is about right for the number of iterations to convergence. As an example, assume we are dealing with 5 model parts and 10 data parts and we wish to know the annealing schedule for the matcher. The Geman schedule [Ge084], will be followed here by using initial temperature T_0=max=50, which is the size of the matcher, and this gives:

$$r = \left(\frac{T_{min}}{T_0}\right)^{\frac{1}{k}} = \left(\frac{0.05}{50}\right)^{\frac{1}{100}} = 0.001^{0.01} = 0.933.$$

4.2.6. The Adaptive Model

For second-order models, the M_2 matrix is a completely connected adaptive relaxation network, like C, but with adaptive weights W_2^{ij} ($w_2^{ij}(i,j,p,q)$, $i,j,p,q \leq n$). Various examples from the training set T of the class to be learned are clamped onto M, meaning that this training input is steadily applied to the nodes throughout the training phase. Each example creates a *memory trace* in the model, which is recorded by adjusting the weights according to the standard Widrow-Hoff delta rule. So for each T_p: $\Delta w_2^{ij}(i,j,p,q) = \eta^{ij}(\phi_p(p)-\alpha[w_2]_{pq}$, where η^{ij} is the learning rate. η^{ij}=0.25 was found to be satisfactory, although any

reasonable value produces good results.

In short, M is trained by being clamped by each of the training examples in T. The system is then annealed many times, with different training examples. Each node really has two values attached to it: its current activation value and its training value. The activation value is free-running and changes during each annealing as information is propagated through the system. After enough presentations of the training data, the system converges on a set of weights that minimizes the euclidean distance between the training values and the current node activation. The system is now ready to be annealed again, but without adaptation and interacting with the matcher and data image. Without such interaction, the system is "daydreaming" and only produces a random example of the input class.

4.2.7. Determining Complexity

One annoying aspect of stochastic relaxation is that you can never know if you have given it too little or too much time to solve the problem, because the number of iterations to equilibrium must be specified ahead of time in the annealing schedule. One approach would be to try out a few problems of varying sizes and iteration times, and choose some complexity relationship that is an acceptable trade-off. When we try this for arches and cups,[10] the (empirical) complexity interpolates nicely into the following polynomial: COMPLEXITY = $6n^3 - 45n + 105n$. This complexity, which is $O(n^3)$, will be the standard complexity of the relaxation matcher, as it is the total number of possible solutions considered by the system during the search. According to the description of the system so far, this is equal to the number of iterations.

Fortunately, the actual time complexity, allowing for parallelism, can be brought down even more if we simply split the total required iterations into a whole array of parallel relaxation matchers. This way, we have many fast annealings performed in parallel. Instead of one long annealing giving a single answer, there are many trial solutions, each produced by an independent relaxation matcher. To choose the best trial solution, the result with the lowest energy of relaxation is taken. This works surprisingly well, and

[10] At least two different values for n seem to exist to give a meaningful empirical complexity, where the complexity is independent of n, which is true as n tends to infinity in this case.

brings the parallel time complexity down to a constant 100 steps. In fact, most images that could not be solved in under 100 time steps (in parallel) were simply too hard for the system and required times on the order of an exhaustive search. The phrase "constant time" here must be taken in a strictly empirical sense, within the context of the types and sizes of images that have been tried. The theoretical performance may well eventually worsen (indeed, it probably does). This is not necessarily bad, because, as discussed in Chapter 2, human ability also seems to break down after a certain threshold point.

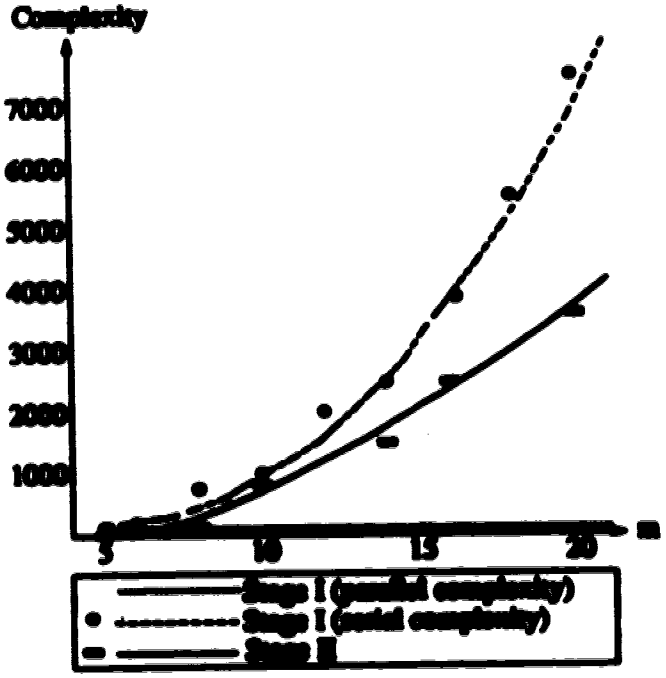


Figure 4.8 Complexity of the relaxation matcher.

Figure 4.8 shows a graph of the resulting complexities for the cup. This is a two-stage process, since a tree-search procedure must be used to get rid of the remaining ambiguities in the relation matrix.

Stage I:
Relax for 100 time steps, in $n = \frac{\text{COMPLEXITY}}{100}$ parallel networks.

Stage II:
Do a tree-search on the remaining possibilities.

Stage I is represented in the diagram as a broken line, for the number of relaxation iterations (i.e. the number of possible solutions examined). Stage I is also represented by the dotted line for the total parallel time complexity. Note that Stage II is below the Stage I graph. Although this is true for small values of n, larger values have Stage II rapidly growing and catching up to Stage I. Thus, there is no way to avoid the serial search phase eventually becoming the dominant consideration as n and m grow. As we saw in Chapter 2, problems like this require distinctions akin to counting a large number of items at a glance. It is much more likely that a human uses things like focus of attention, texture segmentation and segment-matcher interaction to reduce the number of parts to be dealt with. Humans group things into larger groups, which are grouped into still larger groups, so that details in an image are ignored for the purposes of recognition.

4.3. FEATURE EXTRACTION

The *features* consist of two types: *relations* and *properties*. Properties are modelled as relations, but with arcs only between a part and itself. A set of fourteen features has been chosen, with five properties and nine relations, which seem to capture much of the essence of an image. This particular feature set is not written in stone, and the system can easily be adjusted to use any number of features, perhaps even in a domain other than computational vision. The current features are largely adapted from [Dud66] (pp. 254-256, 376-378) and [Duf71]. All features are translationally invariant and are normalized as Gaussian (normal) probability distributions. Mean and standard deviation are computed for each image according to the regions in that image only. Relations between pairs of regions that are enormous or meaningless were left out of the mean and standard deviation calculation. This includes relations between a region and itself, unless the relation is a property. Arcs with value 0 or 0.5 were left out in certain features for which they are meaningless. The background was left out, except for inside/outside and convex/nonconvex relations.[10] The

[10] This is because the background values are so often anomalous compared to the rest of the regions, and this makes a difference when there are only a few regions. This is the only instance in which knowledge of which region is the background was used.

working rule for the images is that only one region may touch the edge, which is the background region, so there must be at least a line line of background pixels surrounding the picture. Images are 4-connected (diagonal pixels are not considered to be connected), and every region must be completely connected.

Feature Definitions

PROPERTIES:

AREA(A): the area of region A.

PERIMETER(A): the sum distance traced out by the boundary of A.

RADIUS(A): $\frac{2(AREA(A))}{PERIMETER(A)}$.

COMPACT(A): $\frac{4\pi AREA(A)}{PERIMETER(A)^2}$.

ECCENTRICITY(A): $\frac{b}{a}$.

a: maximum length line segment covering A.
b: maximum length line segment perpendicular to a.

RELATIONS:

ADJACENT(A,B): the fraction of A's boundary that borders on B.

CONNECTED(A,B):
1, if there is at least one pixel of A's boundary connected to B, 0 otherwise.

INSIDE(A,B): 1, if A is completely enclosed within B, 0 otherwise.

BIGGER(A,B): $1-\frac{AREA(B)}{AREA(A)}$, if $AREA(A)>AREA(B)$, 0 otherwise.

CLOSE(A,B): $1-g(\frac{DISTANCE(A,B)}{MEAN\_RADIUS(A,B)})$.

g(x): the gaussian normalization equation.
DISTANCE(A,B): distance from CENTER(A) and CENTER(B).
CENTER(A): center of minimum rectangle covering A.
$MEAN\_RADIUS(A,B)=\frac{RADIUS(A)+RADIUS(B)}{2}$.

ABOVE(A,B):
1 - CLOSE$_y$(A,B), if (CENTER$_y$(A) > CENTER$_y$(B)), 0 otherwise.
CENTER$_y$(A): y-component of CENTER(A).
CLOSE$_y$(A,B): $1-g(\frac{DISTANCE_y(A,B)}{MEAN\_RADIUS(A,B)})$.
DISTANCE$_y$(A,B): distance from CENTER$_y$(A) and CENTER$_y$(B).

RIGHT_OF(A,B):
1 - CLOSE$_x$(A,B), if CENTRE$_x$(A) > CENTRE$_x$(B), 0 otherwise.
CENTRE$_x$(A) : x-component of CENTRE(A).
CLOSE$_x$(A,B) : $1 - g(\frac{DISTANCE_x(A,B)}{MEAN\ RADIUS(A,B)})$.
DISTANCE$_x$(A,B) : distance from CENTRE$_x$(A) and CENTRE$_x$(B).

SIMILAR(A,B): $\frac{2\sum_{x,y} |A(x,y)-B(x,y)|}{AREA(A)+AREA(B)}$, the cross-correlation between A and B.
The centre of A is offset to the centre of B, and the result normalised by area.
A(x,y) = 1, if pixel (x,y) is in region A, 0 otherwise (likewise for B(x,y)).

CONCAVE(A,B): MEAN_CURVATURE(A,B), if A is concave, 0 otherwise.

A curve of the form $y=ax^2+bx+c$ is fitted to the boundary of A where it borders on B. The curve is sampled at all points except where there is more than one y value for one value of x, in which case the higher y value is taken. Curvature is defined as the absolute value of the radius of curvature. MEAN_CURVATURE is the average curvature taken at each sample point along the curve. If the radius of curvature is negative, the curve is concave down and the region above is concave while the region below is convex. A negative radius means the reverse. A zero or infinite radius means a straight line and curvature is defined as 0 for both these cases. Abovemess (and belowness), as used here, is not the same as *ABOVE*, but is defined in terms of the boundary between the two regions.

AREA and *PERIMETER* measure the size of a region (one gives the total area and the other the length of the boundary). *RADIUS* is not a straight-forward geometrical radius, which works only for circles, but instead uses a standard formula that gives an overall radius-like measurement for any shape of region. *ECCENTRICITY* and *COMPACT* both measure the circularity of a region. *ECCENTRICITY* approaches 0 as the shape becomes more elliptical, and is one for a perfect circle (as such, it actually measures the inverse of circularity). *COMPACT* varies in a similar manner, except that it is sensitive to holes. A region with a perfectly circular outline that has holes will have a low value for *COMPACT* and a high value for *ECCENTRICITY*.

The two relations *ADJACENT* and *CONNECTED* are similar, but *ADJACENT* tells not just whether the two regions are connected, but to what degree. *CONNECTED* gives the topological connectedness of the image. *ADJACENT* is not, as may appear, reflexive: the proportion of A's boundary that borders on B is not necessarily the same as the proportion of B that borders A. *CONNECTED*, being the logical version, is reflexive: CONNECTED(A,B) = CONNECTED(B,A). *INSIDE* is also a logical relation (a part is either

inside another part, or not). The relations *BIGGER*, *CLOSE*, *ABOVE*, and *RIGHT_OF* all deal with distance and size comparisons between regions. These are not logical relations, as they measure the magnitude of the measurement (e.g. *BIGGER*(A,B) measures how much bigger A is than B). *CONCAVE* and *SIMILAR* are also quantitative. *CONCAVE* measures how much one region curves into the other, while *SIMILAR* measures how alike the two regions are.

Chapter 5
Results

5.1. INTRODUCTION

5.1.1. The Implementation

The current system was tested on four types of objects (not all exactly like their real-world counterparts): cushions, cups, tables and stick-figures of people. The images are all two-dimensional and each object is restricted to certain views; the system only learns cups from the front with the handle off to the right, for instance. Besides being hand-drawn and hand-segmented, all images are arranged on a 40 x 40 grid and are 4-connected. Unavoidably, the variety of images used was highly sensitive to the idiosyncrasies involved in constructing the images by hand.

Part of the system is an independent package of C routines, ARE (Alberta Relaxation Engine), for writing connectionist programs. This package contains all the standard connectionist functions, as well as many convenient features for implementing them within a C program (see the Appendices). The computation was shared by three machines: a SUN 3/60, a DEC VAX 11-780 and a MIPS M/120 RISCompute, all running under the UNIX† operating system. These are standard serial computers and are restricted to simulating the parallelism. They provide interactive run-times for the smaller to mid-sized problems, but computation becomes increasingly prohibitive for the larger images. A highly parallel machine would be required to get the full speed-up of constant time complexity.

† Registered trademark of AT&T in the USA and other countries.

5.1.2. Complexity

The gain in speed is paid for in hardware, with a total connectivity of m^2n^2, the number of weights in the matcher. Recall from Section 4.2.7 that t is the number of trials performed in parallel: $t=.06mn^2-.6mn+1.8n = O(n^2)$. The largest problems tried were of size n=7, m=20, with a connectivity of 2×10^6, shared between $mn = 13300$ processors, which comes to 140 connections per processor. Following are the important complexity measures for the matcher (the parallel C network). Note that the serial time complexity is just the product of the parallel time and hardware complexities.

$T_{parallel} = 100 = O(1)$; *parallel time complexity*
$N(n,m) = mn = O(n^2)$; $t=O(n^2)$; *number of processors*
$C(n,m) = mn = O(mn)$; *processor connectivity*
$L(n,m) = mn = O(mn)$; *processor locality*
$S=C^{total}(n,m) = N(n,m)C(n,m) = tn^2m^2 = O(n^4)$; *total connectivity*
$T_{serial} = T_{parallel}C^{total}(n,m) = 100tn^2m^2 = O(n^4)$; *serial time complexity.*

Note that the connectivity and locality of each processor are the same, since each relaxation network is completely connected. Locality must be determined as described in Section 2.12.2, by considering the connectivity of an equivalent ideal processor. Figure 5.1 shows one element of C from this perspective. Only the inputs to the processor and their connectivities are considered. Since the network is completely connected, each processor is obviously not of finite order, so we should not expect to be able to scale up to very large sizes.

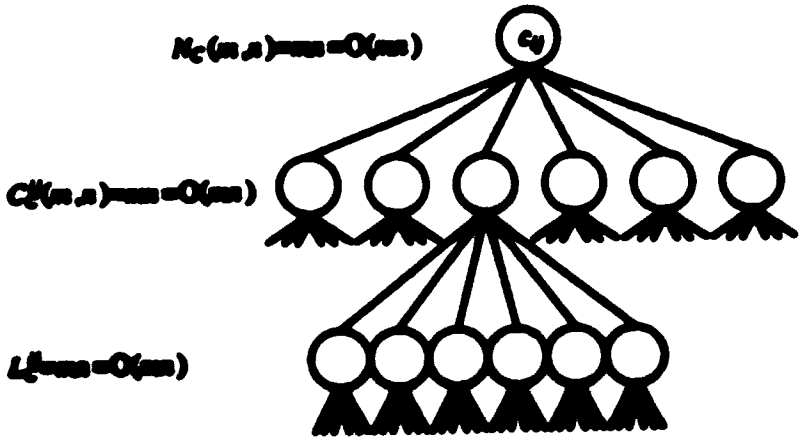


Figure 5.1 Local connectivity for one matcher element, c_{ij}, for m=6,n=4.

While these polynomial complexities are a lot to simulate even on a fast serial machine, they are well within the scope of the visual centers of the brain (of course, it must be kept in mind that this is only one small part of the total visual task). If the brain does, by chance, use a similar algorithm, it would need 13300 neurons with 140 synapses per neuron to do the computation. In keeping with the Feldman constraint, all annealings performed in this chapter converged within 100 iterations. This was also the most practical choice, even when simulating the parallelism on a serial machine. No advantage in serial complexity was gained in any of the experiments by annealing for longer than about 100 iterations, unless it was brought up to the level of an exhaustive search, in which case the results are not of great interest.

5.2. TRAINING THE MODEL

Figure 5.2 shows a selection of typical data images for matching. The individual objects in these images are also typical of the training samples for the model. Before any new images were matched, the system was trained with at least a few samples to give it an initial model to work with. After that, the results of correct matches were used for training.

5.2.1. First-Order Model

Figure 5.3 shows the result for the first-order model of a cup (the background was left out in the diagram, as well as zero values). The axes in the graph are simple averages of all the inputs. An examination of the values will reveal that, while a first-order model may not be very sophisticated, it is quite serviceable. Features that would obviously benefit from a second-order model, like the position of the cup handle, were not varied much in the cup images, although, as we shall see, there was some experimentation with this sort of thing for the robot. The training of the first-order model is a little more than just averaging the data images. There is also the recognition unit, T, which is trained the same way as high-order models. However, since higher-order features (like counter-examples) are not being modelled, the recognition unit needs little training at this time.

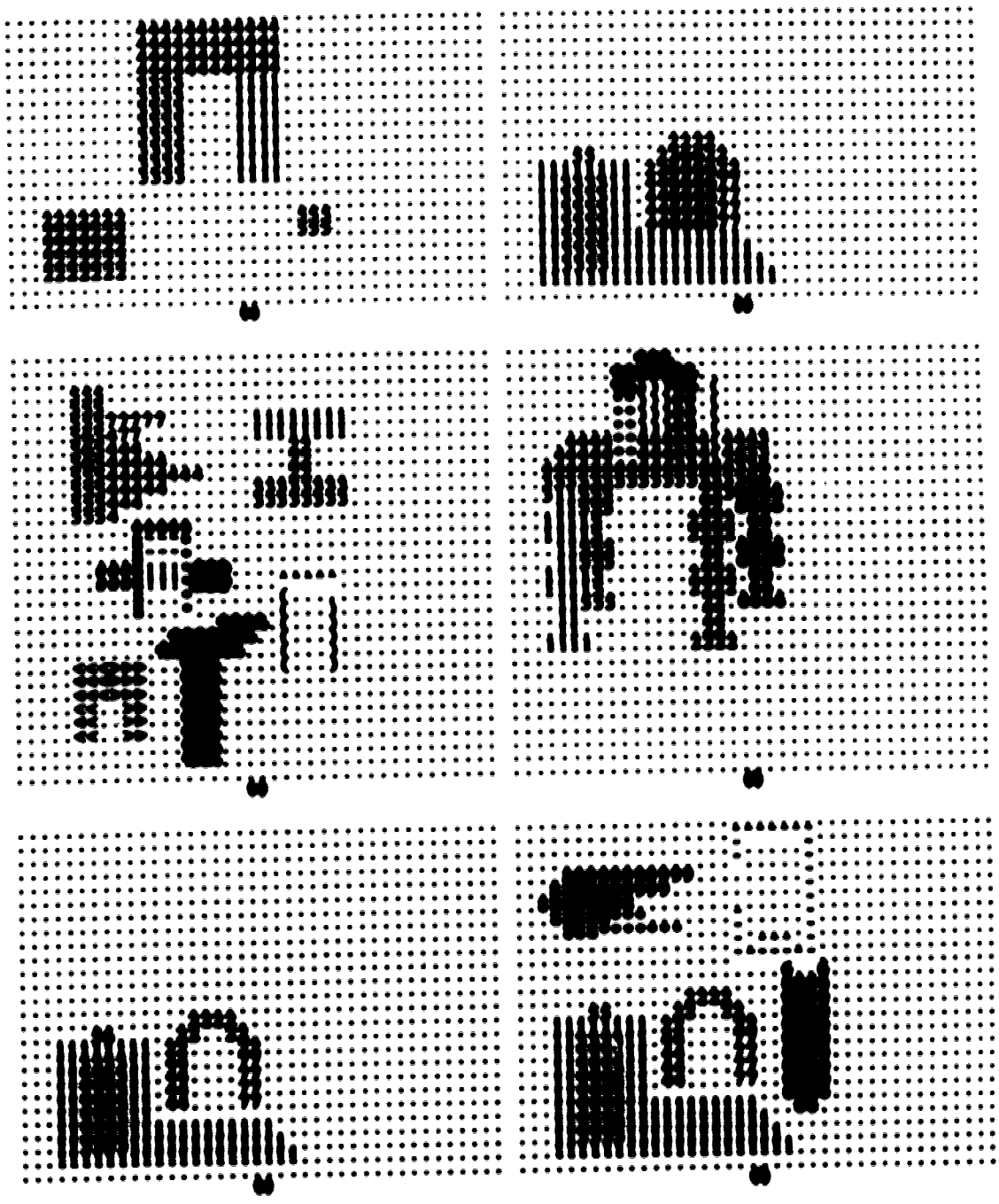


Figure 5.2 Data Images

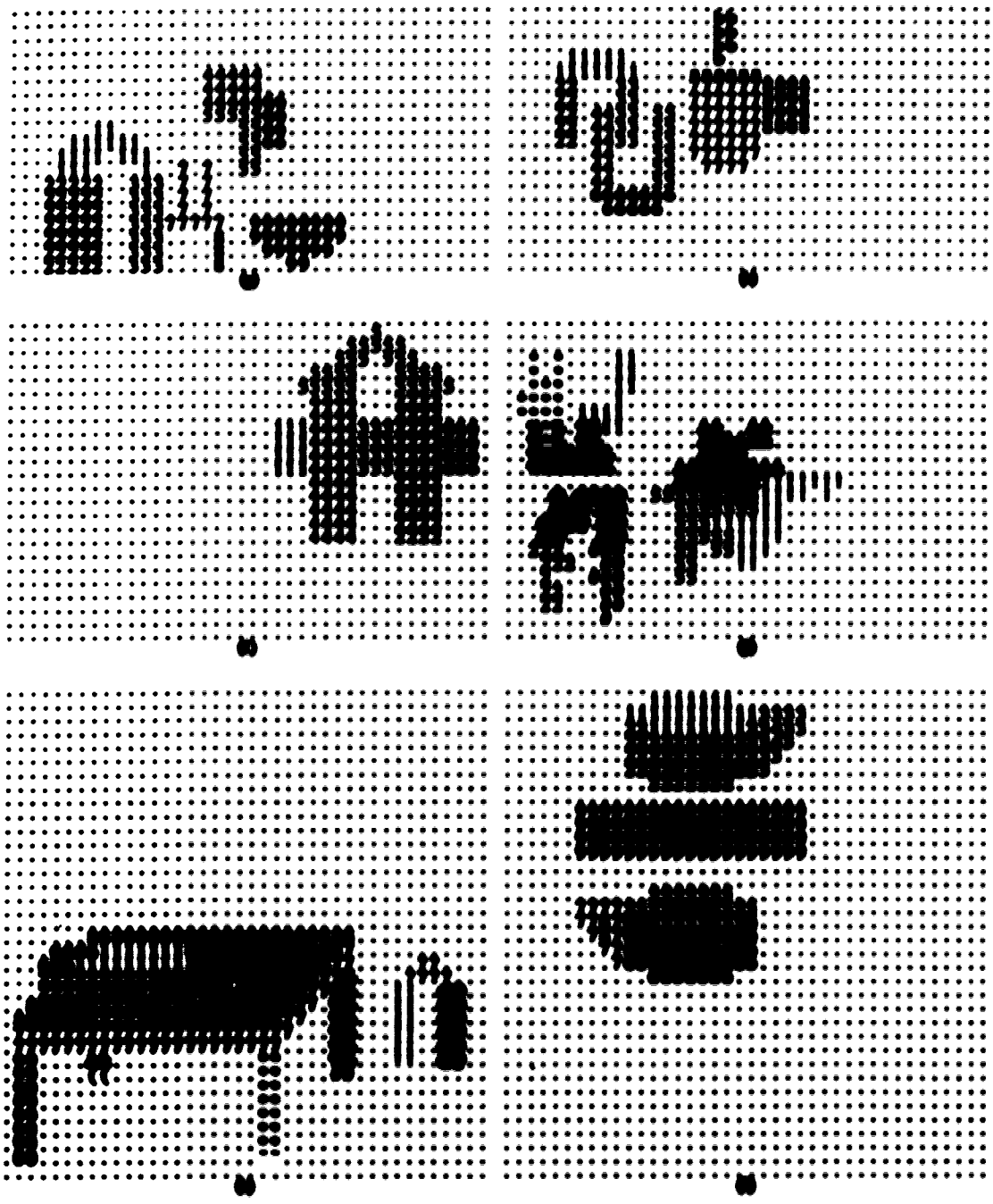


Figure 5.2 Data Images

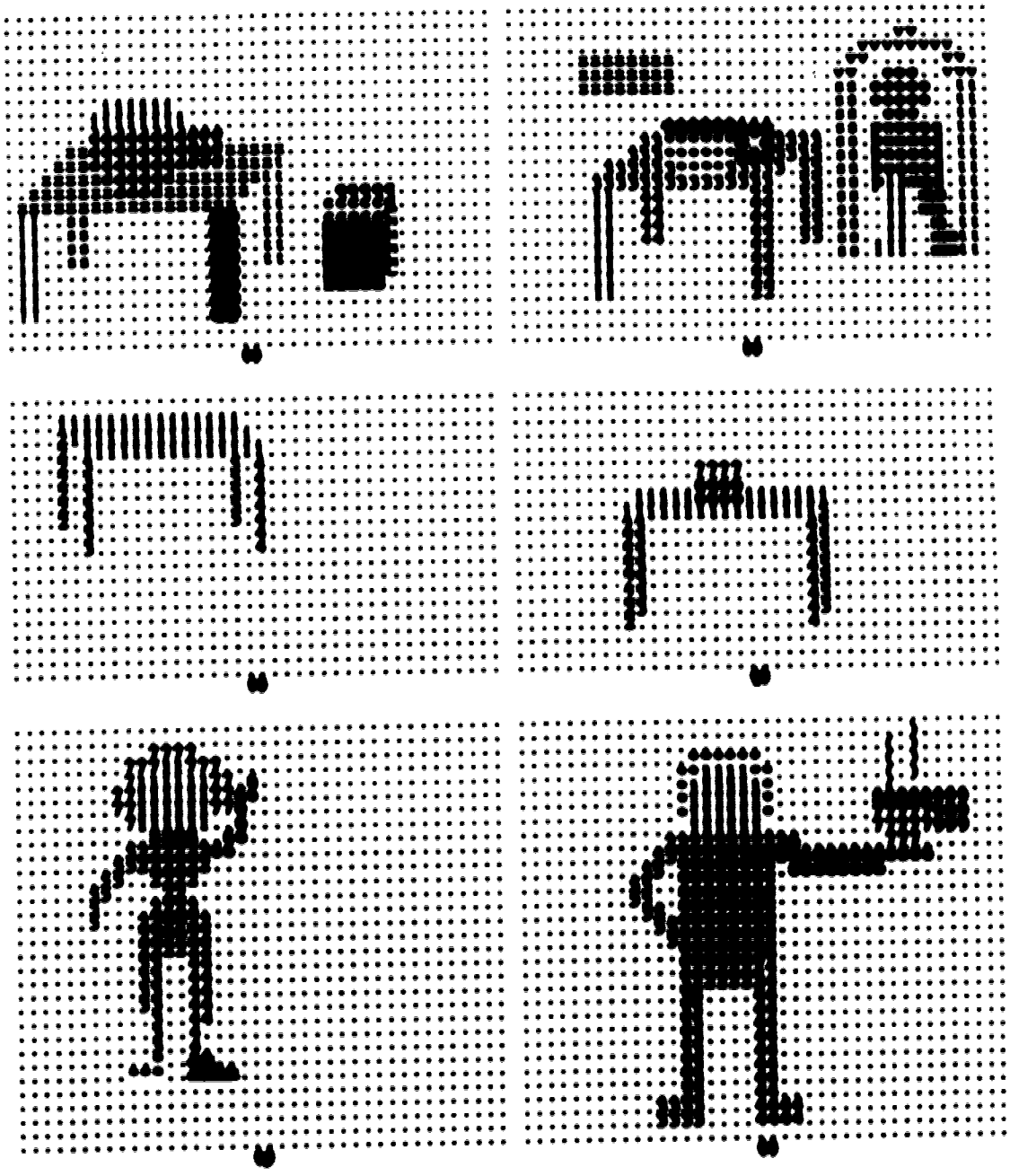


Figure 5.2 Data Images

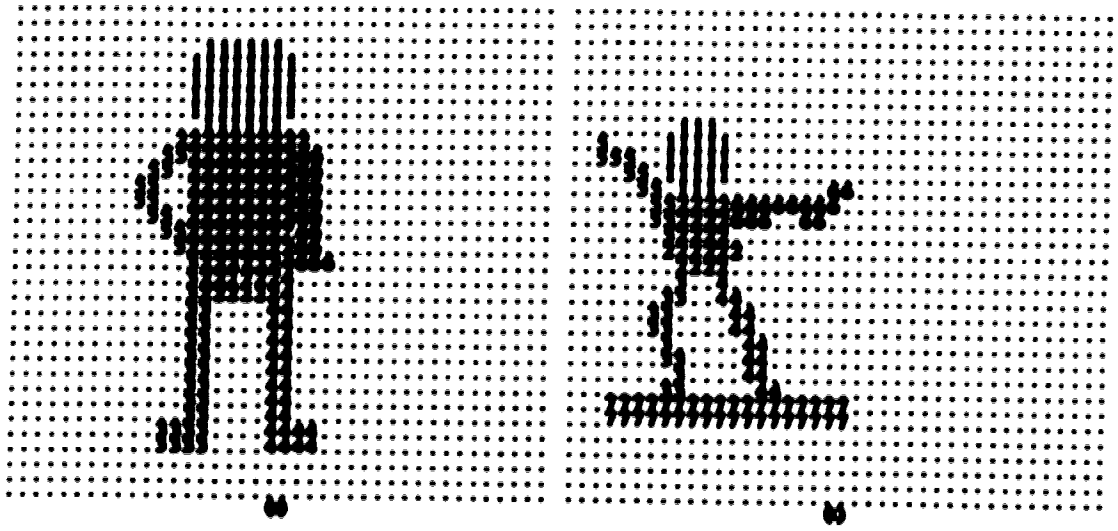
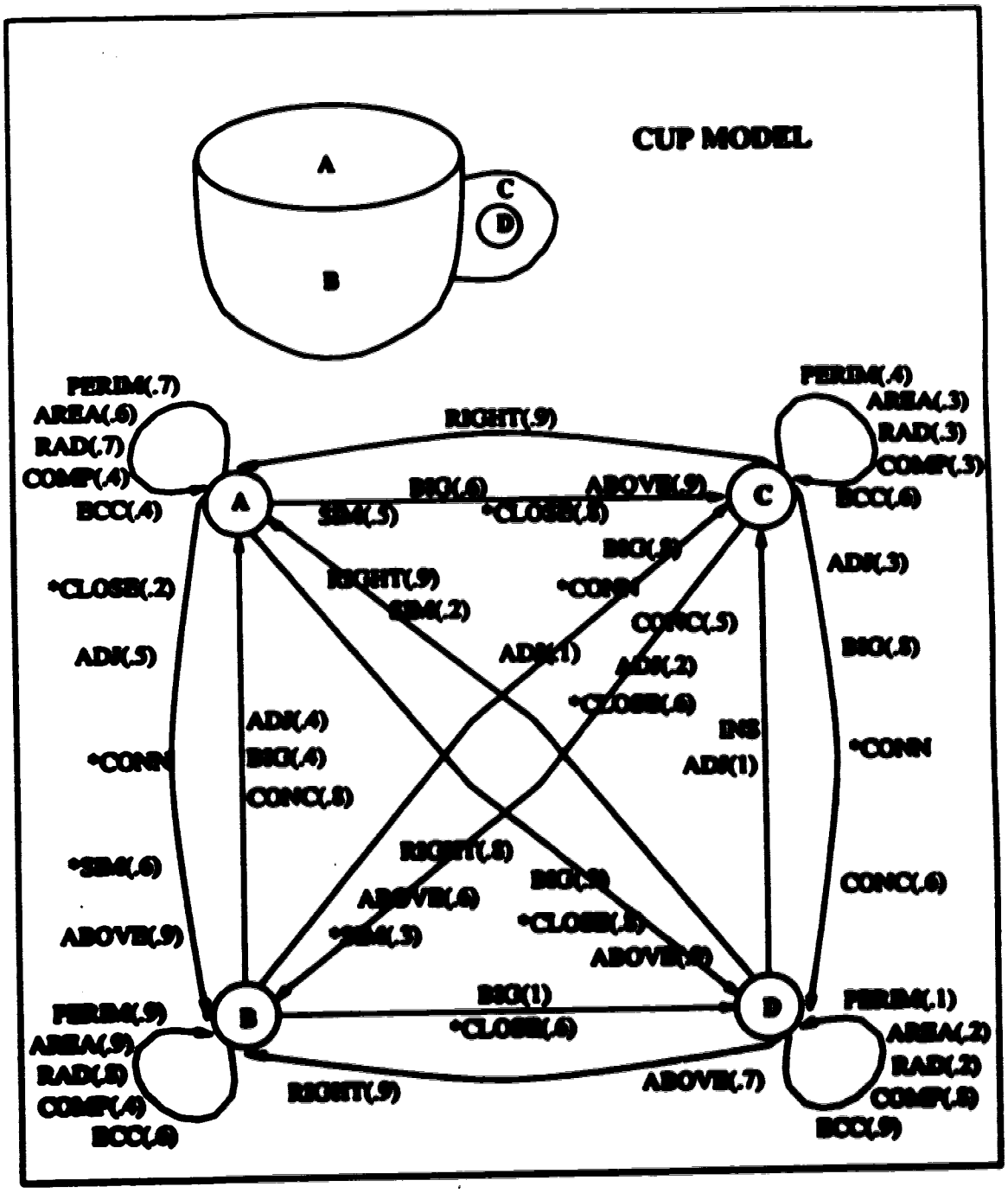


Figure 5.2 Data images.

5.2.2. Second-Order Model

The second-order model, while not used for the bulk of the results in this chapter, does display some interesting behaviour, and produces correct results for small problems. It is produced by dumping onto the model many objects like those in Figure 5.2. The model is then relaxed into equilibrium (i.e. taken to a very low temperature). For the experiments that have been run, $T_{min} = 0.05$ is sufficient. This value was obtained empirically, and was kept constant throughout the experiments. Strictly speaking, this parameter should vary with the size of the problem, as larger systems do require lower temperatures (e.g. in the Geman-Geman schedule the optimal starting temperature is proportional to the number of nodes). However, while there may be an ideal minimum temperature for each problem, the experiments for this system showed that T_{min} is not of crucial importance, as the idiosyncrasies of each individual case completely swamp out any difference in minimum temperature due to the problem size. Thus, T_{min} was set to a constant value (0.05) that seemed to work well across the board.



* symmetric relation

Figure 5.3 A first-order model of a cup.

The same principle applies to other system parameters, as well. If the optimal ξ values, for incomplete matching inhibition, could be known for each case, it would drastically improve the results. Enough knowledge of the ideal settings could well be enough to produce the correct match without a Stage II exhaustive search, as different settings of these parameters produce different local minima. The problem, of course, is that one must know the answer ahead of time to know what the ideal parameter setting is. Furthermore, the landscape could well be chaotic[20] (very small, seemingly insignificant, changes in the problem would produce large changes in the optimal parameter settings).

The best that can be done for such parameters is to base them on what works best (on average) in practice. It is important not to adjust parameters based on a certain problem instance and then attach some significance to the resulting performance for that particular image.[21]

The system is annealed in clamped mode several times. The recognition unit Ψ is also trained at the same time, to recognize whether the final match is or is not an arch. Although only a few annealings were needed, these are only preliminary results. The second-order model will no doubt eventually require significant learning times, as it is expanded to handle more difficult inputs (such as occlusion), larger images, near-misses and other higher-order features.

For now, the behaviour of the model in the unclamped ("daydream") mode, without feedback from the Monitor, will be studied. This, recall, is akin to visualizing a typical example of a concept in your head; no environmental input is involved whatsoever. This produces one instance of the object, which may not be the correct type of cup corresponding to the data image.

---

[20] This does not refer to a chaotic equilibrium state, as the convergence is clearly a fixed point (occasional 2-cycles, which are non-chaotic, were found, but only in the uninteresting case of eliminating Stage II). The chaotic aspect is in the setting of the parameters. If the inputs are given the optimal parameter settings, it is the space that is chaotic.

[21] This is also a characteristic of other, non-local, search techniques. For instance, a heuristic version of Ullmann's algorithm [ULL76] has been tried on some of these same images. The things the algorithm is the necessity to fix certain parameters, it was necessary to provide a threshold parameter above which a node expansion is aborted. Prior knowledge of the optimal value for this parameter produces very fast execution, far faster than the exhaustive method. However, it is just not possible to choose this parameter setting ahead of time, when there is appreciable uncertainty in the inputs.

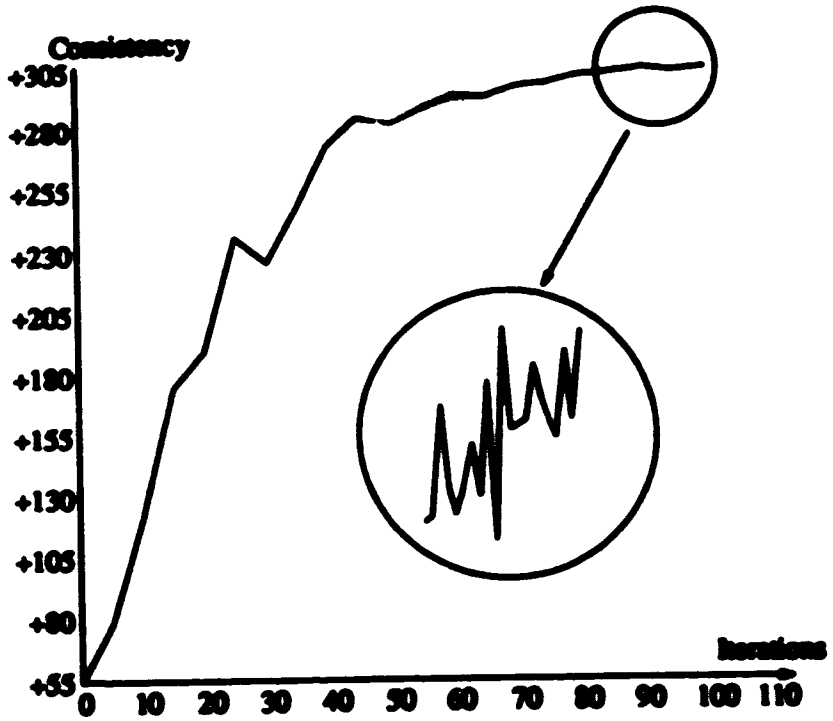


Figure 5A Daydreaming about cups.

A "day-dream" cup model is much like the first-order model, but with much more certainty (it is mostly 1's and 0's), as well as the odd mistake.[22] The mistakes are infrequent and apparently random. The fact that the system gives mostly 1's and 0's is in keeping with the spirit of probabilistic relaxation. Although we may think of BIGGER(A,B) as measuring how much larger A is than B, the proper probabilistic interpretation would call it the probability that A is bigger than B (even by a little).

A plot of the consistencies (negative energies) during the annealing (sampled at every 5 iterations) is shown in Figure 5A. Note that the system does not reach a fixed point. Neither does it appear to cycle. The magnified section at the end of the graph (showing the energy at each iteration) settles into a randomly oscillating, but stable, chaotic equilibrium. The system was also run for times much longer than this and

[22] A mistake is any run value that is obviously counter to the input images. The simple fact that there may be some better second-order model of the input cups is not really a mistake. This is a crisp, stable matching, where a highly consistent local environment is all that is needed for closure, but model is not of much interest.

did not leave its chaotic orbit. No apparent cycling was detectable. What this means for the adaptive model is that there are many roughly equally consistent interpretations of the cup and the system cannot select one over the other.

5.3. SECOND-ORDER MATCHING

Figure 5.5 plots the consistency of the matching of a second-order model of a cup (with the Matcher-Model interaction). No longer is the model a "daydream" cup. Information flows both to and from the matcher and model, using dynamic arc consistency, allowing a non-chaotic convergence. Notice that the consistency curve is fairly smooth, as in the chaotic convergence, except that it increases slowly at first and then takes a sweep up, instead of increasing rapidly and leveling off. Keep in mind that the two curves are not easily compared, since the second-order consistency includes the model, as well as the model-matcher interactions.

It is too early to conclude much about the performance of this system. These results are meant mainly to indicate the large scale dynamics of such a system, not its performance. All that can be said at this point is that it produces correct answers at least for small, trivial problems (n,m≤6) where local maxima are not a big concern. Some of the difficulties that have crept up thus far involve a tendency for both the model and matcher to escalate to the stable maxima of total activation (all nodes set to 1). It is entirely possible that further difficulties could arise that will make this second-order system impractical. For instance, the learning times may well become unacceptable for all but those images for which a first-order model is sufficient. Convergence times for hard learning problems are notoriously slow, often involving stratification.

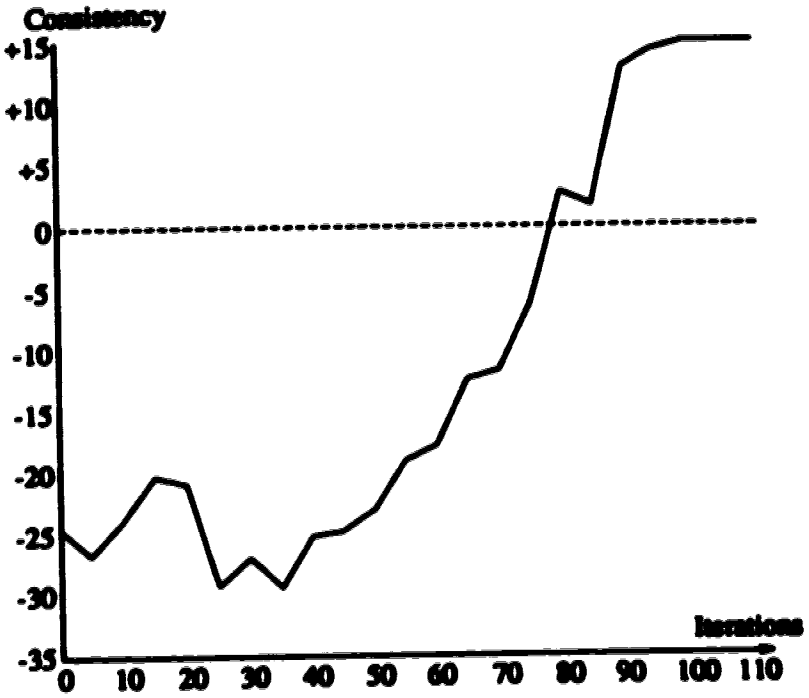


Figure 5.5 Matching with model interaction.

5.4. FIRST-ORDER MATCHING

While the work on higher-order learning is preliminary, first-order models have been studied extensively, and their performance is well understood. While less expensive than higher order models, a first-order model provides the basic essentials necessary: the frequencies of each arc in the semantic networks. As such, the current work deals primarily with the matching phase, not learning. Figure 5.6 plots the consistency for a single, 100 iteration, annealing of the matcher.[23] Notice that, while consistency in the second-order annealing grew smoothly, here the climb to equilibrium is much more erratic, although the convergence is still a non-chaotic fixed point. This erratic behaviour indicates that the matching involves a large global element and needs to bounce around the landscape a lot more to find the right solution. This

[23] The same considerations for T_{min}, T_0 and I discussed previously also apply here. In fact, all annealings in the system, even if they serve very different purposes, use these same schedule: T_0 = ∞, T_{min} = 0.01, I=100.

would tend to make us less confident of the system's ability to avoid local maxima.

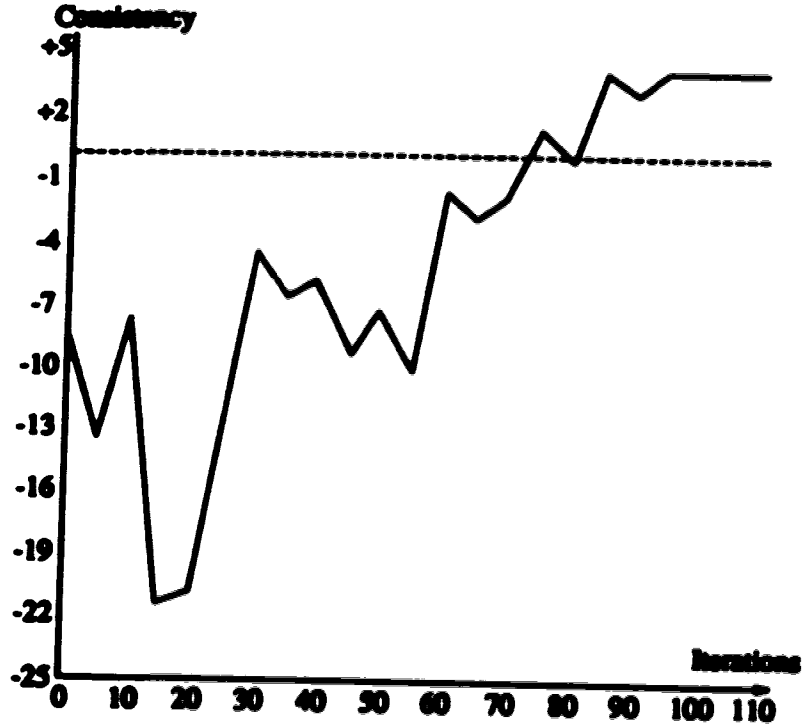


Figure 5.6 Matching a cup with a first-order model.

The global element of a relaxation search is largely a consequence of introducing uncertainty to the environment. Because there are many locally consistent matches available, many fast annealings are performed and the result with the best consistency is taken. This would be a sure-fire way to tell a good match from a poor one if we were computing complete matchings, but this is not practical. There are just too many highly consistent, but dead-wrong, maxima. Unfortunately, if a certain amount of ambiguity is allowed, as in Figure 4.6, there is no longer any way to be sure about the goodness measure for matching ($-E^C$). A potential solution matrix, containing the correct solution, may also include some really poor matches that drag the consistency down below that of other solution matrices that do not contain the correct answer. This is another factor that determines the balance struck in Figure 4.6. It may be tempting to allow a few more matches in the solution matrix, in the hopes of trading off speed for performance, but if many

more matches are allowed, the correct solution gets drowned out by bad matches, making consistency ($-E^C$) a poor measure of the goodness-of-match.

A major stumbling block with solving a problem like this with stochastic relaxation is that there is really only one correct solution. Stochastic relaxation is renowned for its ability to deal with uncertainty, and this application includes lots of uncertainty in the input data. Stochastic relaxation is even better when there is also uncertainty in the performance requirements, because the relaxation network seeks only to give a reasonably good answer. If such a solution satisfied us, we could get all our results in 100 time-steps, with no redundancy and no Stage II search step. But matching is a very specific problem with only one, or very few, correct solutions. Fortunately, redundancy provides a highly parallel way out of part of this problem.

5.5. PERFORMANCE

Figure 5.7 is a table of results for all the images illustrated in Figure 5.2 (along with a few more that are not illustrated to round off the chart). Some difficult images have been included, as well as some easier ones. E^V is the perceptron's judgement of how close the object comes to being an example of the modelled object. The "% correct" figure is the percentage of runs that produced the correct answer. Note that the "correct" solution is *not* necessarily the exhaustive search solution. The results were compared instead to the intuitive, human result, which was easy to determine since all images contained some collection of parts that was obviously correct to a human. While this was, more often than not, the same as the exhaustive solution, sometimes it was not (note the astonished icons in the chart). Although the validity of judging the system against human standards would not be questioned in the real world, it is hampered here by the artificial nature of the toy world of hand-constructed images. Also take note of the Stage I complexity--this is the number of iterations in the relaxation search, provided each trial is performed sequentially. If all the Stage I annealings are done in parallel, the complexity is constant time (I=100).[24] The initial Stage I results

[24] Actually, it is highly doubtful that this constant time is maintained indefinitely. However, there is no apparent growth beyond 100 iterations when dealing with the size of images in Figure 5.2.

given in the chart, however, indicate the number of points in the search space that are visited, which is also a highly relevant measure of complexity.

Image	n	m	Trials	Stage I	Stage II	Exhaustive Search	Matches Per Row	$-E^C$	$-E^V$	% Correct
(-)	4	4	1	100	8	24	1.75	13.4	45.1	100
(-)	4	6	2	200	36	360	2.5	19.1	44.8	100
(-)	4	7	2	200	54	840	2.75	20.0	53.1	75
(-)	4	8	2	200	81	1680	3	25.7	45.8	100
(-)	4	8	2	200	36	1680	3.5	23.2	45.1	100
(-)	4	10	5	500	na	5040	na	na	na	0
(-)	4	10	5	500	192	5040	3.75	28.5	43.0	50
(-)*	4	12	10	1000	240	11880	3.75	39.0	44.8	100
(-)	4	12	10	1000	192	11880	4	36.7	40.3	70
(-)	4	12	10	1000	240	11880	4	37.2	47.0	50
(-)	4	14	16	1600	400	24024	4.5	41.5	40.5	50
(-)	4	21	53	5300	720	1×10^5	5.25	58.5	38.0	50
(-)	4	21	53	5300	1225	1×10^5	5.75	56.7	39.3	90
(-)	5	5	1	100	1	120	1.0	21.8	58.3	75
(-)	5	8	4	400	108	6720	2.6	35.2	36.3	100
(-)	5	10	9	900	708	30240	3.8	47.1	32.0	70
(-)*	5	14	26	2600	na	2×10^6	na	na	na	0
(-)*	5	14	26	2600	3000	2×10^6	5	65.6	33.7	30
(-)*	5	16	37	3700	2500	5×10^6	4.8	66.4	34.6	100
(-)*	5	20	68	6800	3600	2×10^7	5.2	94.7	28.2	80
(-)*	5	20	68	6800	na	2×10^7	na	na	na	0
(-)	6	6	2	200	1	720	1	28.3	77.1	25
(-)	6	8	5	500	324	20160	2.5	39.8	75.3	90
(-)	6	10	11	1100	1152	2×10^5	3.3	63.5	74.3	85
(-)	6	12	19	1900	6000	7×10^5	4.3	91.9	61.1	90
(-)	6	14	31	3100	17280	2×10^6	5.2	118.8	72.3	65
(-)	6	20	82	8200	72000	3×10^7	6.5	126.8	59.8	90
(-)	6	20	82	8200	na	3×10^7	na	na	na	0
(-)	7	7	4	400	128	5040	2	48.8	79.7	70
(-)	7	8	6	600	288	40320	2.3	41.9	79.4	80
(-)	7	10	12	1200	9216	6×10^6	3.7	88.1	76.4	100
(-)	7	12	22	2200	na	4×10^7	na	na	na	0
(-)	7	14	35	3500	211680	2×10^8	5.9	164.5	74.5	30
(-)	7	20	95	9500	1×10^7	4×10^8	10.4	357.1	69.5	25

Number of iterations = 100
* exhaustive search fails, "% Correct" figure applies to human solution only

Figure 5.7 Performance statistics for the images in Figure 5.2.

The Stage II search phase is also an important part of the chart, even though it was a trivial part of the system implementation. Time spent in Stage II is a measure of the success of Stage I. The complexity measure used in the chart is, as for Stage I, the number of possible complete matchings considered in the search (i.e. the number of leaf-nodes in the tree). This complexity, as predicted in Chapter 4, grows fast enough to eventually take over and become the predominant consideration. This is not a surprise, as human capabilities also break down quickly after a certain point. The column for "exhaustive search" is the number of leaf nodes in a complete search of the state space, with no Stage I relaxation.[25]

5.5.1. The Effect of Uncertainty

In analyzing Figure 5.7, a distinction will be made between three basic types of uncertainty in the data images: *noise*, *context* and *structural aberrations*. Noise consists of purely random aberrations resulting in extraneous regions and improper shapes. Context is the surrounding parts in the image, usually consisting of other objects. Structural aberrations consist of major changes in the form of the object being searched for, such as a part that is flat in the data image, but concave in the model. Most of the analysis of Figure 5.2 will lump context and noise together, since the distinction is blurred for this system, due to the hand-construction of the images. "Noise" in these images was put in by human hands and thus is not truly random and could as easily be labelled context.

All three types of uncertainty can render a discrete or deterministic algorithm useless. Discrete algorithms eliminate impossibilities, which can then never be reconsidered. Systems like Cooper's Blocks-World [Coo357, Coo80] work in linear time, but rely on the certainty inherent in a small set of highly constrained parts and relations. This system has much more uncertainty, and thus a probabilistic model was needed, where evidence is weighed and no possibility is ever completely removed from play. The system came quite close to a similar performance for images with little uncertainty. It is the introduction of this uncertainty that was hoped to make up for the lack of real-world data in the present implementation.

---

[25] Note that solving the entire problem with Stage I only (no Stage II) requires more or less or bad or them.

It is the uncertainty that requires the use of the multiple, parallel trials and a final search phase, neither of which were necessary when highly constrained, exact matching problems were tried.[26]

5.5.2. Analysis

Image (a), a typical arch, has no noise and little uncertainty. The only problem is finding it amongst the parts in the image. Data like this comes close to reducing to the clean certainty of discrete systems like Cooper's. Cooper's proof of correctness, while still not provable for this system,[27] is probably a reasonable heuristic to use here. Images with little uncertainty are boring and the system comes close to 100% performance.

Now look at image (i), also a typical arch. Here, there is context in the way of a rectangular object behind the arch supports. These three regions are related to the regions of the arch in many of the same ways that the arch parts are inter-related. Cooper's correctness proof probably is not as valid here. Many apparent inconsistencies are just noise or context getting in the way, and a possibility can never be eliminated with 100% certainty. Notice that the system manages this matching with only a 79% reliability. This is still good, especially when you consider that many of the wrong matches are close--they just get the two supports of the arch confused, because of the symmetry inherent in the model. Such "almost correct" matches are quite common, especially when run-times are on the borderline of good performance, but they are not taken into consideration in the chart because there is really no objective way to evaluate their validity or usefulness. If the number of iterations in these cases is upped somewhat, the answer is almost always correct. For poor performance and completely meaningless matches, however, the run-time will usually have to be increased to the level of exhaustive search to see any reasonable results at all.

Note that the value of e^{-E} for image (f) is the highest of all the scenes on the chart.[28] So although this

---

[26] These trials consisted of a single model, rather than an average of many different training examples, as well as data images that contained objects that differed in no substantial way from the model.

[27] It is impossible for two reasons. Most important is its reliance on the certainty inherent in Cooper's three-way model, but it is also impossible simply because Cooper's correctness theorem differs in implementation detail from mine.

[28] Recognition, that is the percentage it tells us how close to the model the matching matched arch is. This is not the same as e^{-E}, which measures only how good the match is, not the arch.

arch was harder to match than many of the others, it is closest to the model once the context is removed. Notice that the worst arch was (c). This is mostly because its parts are only one pixel wide. The item directly below (c) in the chart (not shown in Figure 5.2) is identical, except that the noise and context adjacent to the object is cleaned up. The biggest difference is that performance jumps up from 50% to 90%. The clean version is still a poor arch, only a little bit better than the dirty version. Thus, the real reason for the low (50%) performance was the noise and context, not the unorthodox style of the arch.

The system's performance does not degrade gradually as the images get larger; instead, the behaviour becomes more erratic. This makes the interpretation of the performance more difficult. Even for the largest problems, many are still solvable close to 100% of the time, even with a fair amount of uncertainty, but there are also more totally unsolvable items. These images are often unsolvable even by exhaustive search (note the unstarred items in Figure 5.7).[20] There is nothing contradictory about an image being unsolvable by exhaustive search. Some are just so noisy, there is a solution that better fits the system's definition of consistency than the one that seems most intuitive to human eyes. In these cases, the system usually does not find the exhaustive version either.

On the other hand, there are some images that exhaustive search fails on that the system handles easily, meaning that the intuitive answer is in a more stable spot in the landscape. Take, for example, image (k). When this is matched against the cup model, exhaustive search fails, whereas the relaxation matcher has an 80% success rate. Image (n), on the other hand, is recognizable neither to the system nor exhaustive search. There is a pattern here that explains this. Part of the uncertainty in image (n) is the inside part of the cup, which is just a single straight line. In most of the training cups, it is curved, both with respect to the background and the body of the cup. Few of the images without this curve are recognizable. This deviation from the norm is more than just noise or context, it is an actual structural alteration from the model. In this case, the change is just too much for the item to be considered a cup. Image (k) also has uncertainty, but a kind that the system finds much easier to handle, since it does not so radically alter the

[20] Exhaustive search was not tried on problems for n26, n28.

structure of the object as (n). This is all as it should be. After all, structural changes can be a valid reason for creating a new concept. Many humans might agree that image (n) is not a cup. Like humans, the system can tolerate only a certain amount of structural aberration, to which it is much more sensitive than noise and content.

An interesting aspect of the table recognition is brought out by image (n). It is the second hardest table on the chart to recognize, at 25%. This is most likely because it is lined up in our field of sight with the back legs to the left and the front legs to the right. Only about 20% of the training examples are lined up in this manner. This highlights the stochastic, statistical nature of the program. If a small handful of the training examples are arranged like this, then it will recognize such an object only a handful of times. The system does not generalize and decide that 20% is high enough to warrant full tablehood. You can decide for yourself whether this is good or bad. Another interesting thing about table (n) is that, although the system seems sensitive to its misalignment, when it does get matched properly, it is the best example of a table in the chart (according to $E^{\mathcal{T}}$). So, while the matcher does not like this variety of uncertainty, the recognizer has no problem with it.

Most of the results can be understood in a similar fashion to the examples above. When there is a failure, it can usually be understood when the image is examined.[39] Of course, the system is complex enough so that the root of the problem cannot always be pin-pointed. Image (q), for example, seems to follow the norm of stick-figures closely enough for a good match, especially since there does not seem to be much uncertainty. There are, certainly, major deviations, and although none seem important enough by themselves, perhaps they add up enough to fail the system. For instance, the figure is female. Most of the training samples were male. Thus, the central part of this figure's body is concave with respect to the background, whereas male stick-figures are generally convex or flat. The figure also has two extraneous parts, one of which spoils the normal connectivity of the object. However, similar content in the tables does not

[39] There are many other interesting points to be made about these figures. The reader may find it an entertaining pastime to try to spot some more. If so, try to note whether the uncertainty comes from a change in the fundamental structure of the object or just deviations from an essentially unchanged structure, along the lines of images (k) and (n).

seem to be a problem. Some of the tables do not have the standard connectivity used in the training examples, because of an ove-hang area on the table-top. However, the system seems to pay no mind to this and still locates the tables easily.

If the problem with image (q) is the figure's sex, perhaps the concept it has learned would better be called *man* rather than *person*. This would certainly be the most favourable interpretation of this failure, as the differences that make the figure female are more structural than noisy. Again, the system seems more sensitive to structural changes than to noise and context.

Unfortunately, there is no way to measure what it was that caused the system to fail so totally with image (q), although it can be helpful to analyze the results in the intuitive way done above. Even these causes given are not simple to predict and understand, since sometimes a small, totally random change in a data image can cause the system to change from success to failure, or vice-versa. Besides these unmeasurable, random factors, however, what makes one image difficult and another easy seems to have more to do with the characteristics of the images than with a short-coming in the search technique. In other words, while the system may not be as smart as a human, many, although certainly not all, of its mistakes are of the same general character as human mistakes.

Chapter 6
Conclusion

6.1. SUMMARY

A system was implemented that successfully uses stochastic relaxation techniques in matching relational graphs of data images against an internal model built up from training examples. The relaxation is used as a pre-processing stage for a tree-search procedure. First-order models, with minimal learning, are fully explored, while some preliminary ground-work is done on second-order learning. When dealing with noise and uncertainty, the local maxima problem becomes unacceptable for a total relaxation solution to the matching problem. The system must typically examine as many possibilities as an exhaustive search to find the correct answer. This problem is an example of the difference between an exact fit (no uncertainty or noise) and a good fit.

To overcome this problem, it is necessary to settle for eliminating a large number of possibilities, and having a second stage search program select the answer from among the remaining possibilities. Once our sights are lowered and we do not expect relaxation to solve the entire problem, the results look good. The system has a good success rate for clear-cut cases and stands up well in the face of a wide variety of different inputs. When it fails, its failures do not seem unreasonable, given the training data that was used. The empirical complexity is constant time (100 iterations) for realistically-sized images, although this requires a high degree of parallelism. The number of possibilities examined in the search is $O(n^3)$. The time for the search phase (Stage II) starts out lower than this, but grows faster, and so will eventually take over.

6.2. FUTURE WORK

6.2.1. Interaction With Segmentation

Some of the system's short-comings, such as the inability to handle occlusion, are not illustrated in the examples of Chapter 5 because the system cannot handle them, even in theory. When one object partially obscures part of another, it results in a single region split into two parts. For example, in image (i) of Figure 5.1, a rectangular object behind the arch is partially obscured by the arch. This is not fatal, because the arch's parts are intact; it is just noisy. However, if the arch were behind the rectangle, instead of in front, it would have five parts, not three. In this case, what is left is an arch with short legs, but other arrangements could be even worse. For instance, if the occlusion were to cut through the top segment, there would be no way the system could recognize it as an arch, as the number of parts would not even be correct.

To solve this problem, the system needs more than interaction between the model and the matcher. It might be possible to implement a system that can perform many-to-one matchings, so that two different regions in the data can both match with a single model part. However, this does not seem to be a very promising route, as the occlusion (or other problem)[31] damages the information in the arc consistencies, so it is not just a matter of too many parts.

A better idea would be to hook up the matcher with the lower-level segmentation phase. Proper segmentation of such an arch requires evidence of what the parts might possibly match to. It is not enough only to parse out highly differentiated areas and pass them on to the matcher as the regions of the image. The general scheme for object recognition discussed in Section 3.1 is overly simplistic. We have already seen how data must ultimately flow back from the matcher to the model. Now we discover that it must also flow back to the segmenter, a level of processing that is usually talked about as separate from matching.[32]

---

[31] Other problems that fall under this category include any image where an identifiable part of the object is split into several regions that cannot be distinguished as being all a piece of the same part, without prior knowledge of what the object is. This is a common problem in computational vision, occlusion being only the most basic example.

[32] Many researchers recognize the ultimate need for this kind of feedback, and an independent segmentation phase is considered merely a necessary simplification.

This is where symbolic techniques run into trouble. Keeping the two tasks separate is no problem, but communication is difficult when the two sub-systems speak totally different languages. Controlled relaxation provides a way of translating between the languages, and at a level that is dynamically tied into the fundamental workings of the sub-system. Here, both tasks are in separate networks (an adaptive model and a WTA-style matcher), but the two systems are allowed to interact through the μi units right at the bottom level, rather than by passing the symbolic result of a previously performed computation. The controlled relaxation allows a bottom-up emergent style of computation, while providing a method of supervising and coordinating the activity. This allows sub-systems to communicate, not by passing large, cumbersome, pre-packaged symbolic structures, but at the most basic level of their operation. Two completely different functions can "speak the same language." One could imagine an iterative process where a relaxation segmenter is controlled by the matcher/model, and vice-versa, much as the matcher and model interacted in the second-order model. This sort of coupling between sub-systems is much stronger than is traditional in AI, particularly if taken to the extremes suggested here and applied to the entire visual process, and even beyond.

6.2.2. Interaction With Higher Levels

This general scheme could even be extended (although this is more speculative) to tie in the visual process with higher-level reasoning and creative processes. Unfortunately, with this strong coupling comes a theoretical difficulty, of a kind often seen in AI systems. As we saw in Chapter 5, coupling the matcher and model changed each in fundamental ways. This means the entire functioning of a sub-system may need to be re-studied just by the action of interfacing it to another sub-system. If we were to continue tying the various parts of the system together in the same way, each step would not only make things more complicated, but would change the entire behaviour of what we have already accomplished.

By the time the visual system has been coupled with some higher-order reasoning centre, there will be many such sub-systems coupled together. This is not to say that very low-level functions like segmentation will be directly coupled to higher-level functions, like reasoning. However, reasoning would be strongly

coupled with the higher order visual processes, which in turn are coupled to the lower-order segmentation. Thus, there is still (indirect) interaction between the reasoning and the segmentation functions. Understanding each of the two sub-systems (REASONING + RECOGNITION) and (RECOGNITION + SEGMENTATION) is not necessarily equivalent to understanding the entire system (REASONING + RECOGNITION + SEGMENTATION).

This characteristic of strong coupling can already be seen in going from the first-order matcher to matcher-model interaction. Many of the basic gross behaviours of the system change, and much of the analysis will have to begin again. This hardship, however, is probably not a bad thing. To paraphrase Minsky [Min86], there must be a balance between *distribution* of the overall task and *isolation* of the sub-tasks. Some have speculated that most computation in the brain may be communication between the various sub-components in Minsky's "Society of Mind."

Extreme connectionists, who do not see insulation as a problem, but insist on total distribution, might not like the coupling between the second-order model and the matcher, because they would not separate the two tasks in the first place. They would argue that total distribution is possible. This would mean simply clamping on training examples and hoping for an internal model to develop that includes the concept of matching. This would be expecting the system to learn how to match. This is asking far too much of one relaxation network. A single network is just not capable of doing these two computations at once without resorting to extreme coarse-coding.[33] This would be asking the system to *learn* the differences between all the exponential permutations of the object. Such a system must perform a statistical measure of frequency (to build the model) and learn the matching procedure at the *same* time, all coded into the *same* weights. This is more like the evolutionary process than cognition.

The strong coupling described here seems to provide a way of maintaining insulation in a highly distributed system. While we may have to re-think each sub-system before coupling it to another, some form

---

[33] Recall the idea of coarse-coding from Chapter 2. This could be used to coarsely build a look-up table within the network, which requires unacceptable connectivity and very long learning times.

of this trade-off is probably necessary anyway. Another potential problem with the scheme for brain functioning presented here (at least in its most extreme form) is that it requires all sub-systems in the brain to "speak the same language," the language of probabilistic relaxation. Although there are many good reasons to consider object recognition a highly parallel task, some may argue that higher-level functions, such as reasoning, planning and language, cannot be expressed in terms of relaxation. [FoP88]. They believe that the relaxation mechanism is inadequate for representing combinatorial syntax, necessary for the manipulation of logical structures and language. This is a problem of representation, not complexity. In language, for example, humans have modest vocabularies, but the syntactic rules of grammar allow us to put these words together in an infinite variety of ways. It is argued that representations amenable to relaxation are inherently insufficient for this sort of logical manipulation. While combinatorial syntax may not be so important for visual tasks, a good case can be made that it is a must, in some form or another, for reasoning, planning and language. Pure connectionists, on the other hand, would argue that even language must be a relaxation process, and indeed there have been connectionist models of language processing, although they are currently less than adequate [FoP88].

Even if some brain functions are not amenable to the relaxation formalism, the general ideas expressed here could still be useful. However, once we have implemented all we can using relaxation, and have coupled these sub-systems together, the remaining sequential functions will have to use some other method of communication. The nature of this distinction, if there is one, is very much an open question. It seems likely that associative, parallel modes of operation (generally formulatable as relaxation) are not restricted to perceptual tasks, but are a key aspect of higher-level thought processes. However, it also seems that there are other aspects of cognition, particularly language, that do not seem to involve much associative processing.

As far as the relaxation matcher goes, the system's most urgent need is for a segmentation phase, probably using some form of this stochastic control. Segmentation is a priority, not just so the system will handle occlusions and similar problems, but because the current work lacks the backing of any real data.

Although the images used are in many ways more realistic than the simulated images sometimes found in the "real-world" of computer vision laboratories, the segmentation is still completely failed, so the project needs this more than anything else.

6.2.3. Higher-Order Learning

The preliminary work on second-order models and learning should also continue and be expanded to include counter-examples (near-misses) and other higher-order concepts. This project has only begun to explore the possibilities (and difficulties, no doubt) with learning in a system like this. Hidden units could also be added to handle concepts higher than second-order, like sub-types and distributed representations. For example, perhaps our model of a cup should be a disjunction, consisting of (cup_type_1 OR cup_type_2). The purpose of this could be to distinguish coffee mugs and tea-cups as different sub-types of the same class. Or, perhaps the types are just different views of the cup (e.g. cup_from OR cup_side). The more high-level information like this that is required, the more hidden units are needed.

Such networks require formation of internal representations; learning for them is many times more difficult than in single-layer networks. This type of model will continue to require dependence on the matching phase, so it can tell which object, or object sub-type, to choose. The earlier example of the right and left-sided handles is just a simple version of this. Using distributed representations of completely different concepts is in the other extreme. For instance, recognizing multiple views of an object as sub-types would be a difficult learning task because the various views can be structurally very different. So distributed representation may not be the best way to handle multiple views. Sub-types are more appropriate for distribution, as these patterns are not so completely orthogonal to one another. Since the system does not currently have distributed representations, it only matches against a single-object, single-type, single-view model. If the system is to distinguish between mugs, cups and tubes, the image must be matched against all three models in parallel, after which the highest value of "S" determines the correct model. Perhaps some of the higher-order concepts not suited to distributed representation could be implemented by coupling between these separate models, networks.[24]

---

[24] The matching does a certain gift use of the network interaction as a "sum-of" for the problem of cup/tube ambiguity. Indeed, it is by no means clear exactly how this strong coupling would be minimized. Most of this discussion is highly

6.2.4. Other Techniques

Finally, it should not be forgotten that future expansions need not be restricted to relaxation techniques. There may well be important aspects of the problem that are not best suited to relaxation. For instance, we have already discussed the necessity of resorting to heuristic search to find the final solution in the remaining possibilities. This conflicts with the grand scheme of tightly coupled relaxation systems, but even within that scheme, there could be aspects of the problem that do not need to communicate heavily with the surrounding sub-systems. On a grander scale, we will probably eventually need to go beyond the relaxation mechanism, although it is useful to try and push it as far as possible. At least by the time we have reached speech and planning, relaxation will probably be inadequate by itself. Although these are highly serial problems, they still make use of visual representations.[35] We will eventually need to discover ways to get these systems to interact.

6.3. CONCLUDING REMARKS

The eventual need for seriality discussed above should not be confused with the Minsky-Papert limitations of connectivity and parity, which are shared with humans. Most human perceptual abilities are not only blindingly fast, but are also limited because they do not rely on fine distinctions, but rather on general patterns. In other words, the types of tasks humans do very fast are just the sorts of tasks that are suited to parallelism. The Feldman constraint tells us that the brain can do incredibly complex visual tasks in only 100 time steps. One can imagine something like the relaxation matcher implemented in a human brain, because the parallel complexity is constant time. This would mean that the brain accomplishes much of what it does by large-scale redundancy--performing many versions of the same parallel computation all at once, and taking the best result. This is parallelism within parallelism, and is certainly in keeping with the highly redundant nature of the brain, as we now understand it.

---

speculative.

[35] The issue of whether higher-level reasoning employs visual representations is one of those controversy within the cognitive science community.

While this all fits well with our conception of brain processes, the technique is not ready for practical use in the real-world. Even assuming the extension to real-world data does not present fundamental difficulties, there is the problem of parallelism. The system was written on a relaxation *simulator*, running on a serial machine. In this context, the system is far slower than 100 iterations and quite impractical. Even within a parallel implementation, the system does not extend to large matching problems. This is appropriate, since humans themselves also fail at larger problems.

Another problem is that the system has failed to fully parallelize the matching problem. The stochastic relaxation mechanism has proven too weak to find the best match, and a Stage II search phase was necessary. This serial processing phase may prove a bottleneck when attempting to tie the system in with other cognitive sub-systems.

It could be argued that a simple exhaustive or heuristic search should be sufficient, since the problem sizes are small and modern electronic computers need not be hampered by the 100 step constraint, since they are many times faster than slow neurons. Indeed, provided one has a serial computer that can perform the exhaustive search in sufficient time, one might actually prefer this method. This would be fine if one was interested only in engineering and not in human cognition. It would, however, provide little insight into human thought processes. This project has concentrated on a small aspect of cognition for which it might be practical to consider serial search techniques. Unfortunately, it is a common phenomenon in Artificial Intelligence for such serial techniques to fail totally when the problem is scaled up. A central tenant of this work is that the limitations of human processing are intimately connected with its flexibility. Fast computers that allow solving small portions of the cognition problem by brute force do not ultimately provide the tools we need to understand the brain. Serial search techniques cannot allow two functionally different subsystems to communicate at the most basic level of their operation, in parallel. The relaxation matcher provides this mechanism and does so within a system that can handle uncertainty in a reasonable way. This is its central advantage over other techniques. Any one piece of the brain may be solving a problem which, by itself, can be done with a brute force method. However, it is doubtful that such a serial algorithm will be of

much use working within the context of the entire brain.

Some have argued that parallelism is a red herring, a mere implementational concern that distracts from the real issues. While it is true that any parallel program can run on a serial machine, this is more than merely implementational. Forcing oneself to adhere to the unique advantages *and disadvantages* of massively parallel systems, leads to a very different approach to the problem of cognition. One could easily imagine, for instance, acting about the mundane task of "parallelizing" a back-track tree search. Such an algorithm cuts off partially formed solutions from the search-tree that cannot possibly contribute to the correct answer. This is exactly the purpose of the relaxation mechanism. So in a sense, it is just a parallel implementation of a tree-search. However, it is also much more, as the style of computation is very different. Spreading the problem so thin amongst all the processors forces us to use "weak," "local" methods in evaluating the partial solution, which leads directly to the kinds of behaviour characterized by the Minsky-Papert limitations. It is interesting that the same "limitations" forced on us when going parallel could be the key to a more human style computation.

References

[Bai87] H. Bai-lin, ed., *Directions in Chaos*, World Scientific, Singapore, 1987.

[BaB82] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[BaF71] H.G. Barrow an . R.J. Popplestone, Relational descriptions in picture processing, *Machine Intelligence 6*, (1971), 377-396.

[BaS87] J. Baumann and K. Schulten, Influence of noise on the function of a 'physiological' neural network, *Biological Cybernetics 56*, (1987), 313-327.

[CaG87] G.A. Carpenter and S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing 37*, (1987), 54-115.

[Cha77] G.J. Chaitin, Algorithmic information theory, *IBM J. Res. Dev. 21*, (1977), 350-359, 496.

[CoH87] P.R. Cooper and S.C. Hollbach, Parallel recognition of objects composed of pure structure, *Proc. DARPA Image Understanding Workshop*, Los Angeles, CA, 1987, 381-391.

[Coo88] P. Cooper, Structure recognition by connectionist relaxation: formal analysis, *Proc. CSCSI Conf.*, Edmonton, AB, 1988, 148-155.

[CoG79] D.G. Cornell and C.C. Gotlieb, An efficient algorithm for graph isomorphism, *J. ACM 17*, (1970), 51-64.

[CrM83] F.H.C. Crick and G. Mitchison, The function of dream sleep, *Nature 304*, (1983), 111-114.

[CrA85] F.H.C. Crick and C. Asanuma, Certain aspects of the anatomy and physiology of the cerebral cortex, in *Parallel Distributed Processing (Vol. II)*, J.L. McClelland and D.E. Rumelhart (ed.), MIT Press, Cambridge, MA, 1986, 333-371.

[Deu85] D. Deutsch, Quantum theory, the Church-Turing principle and the universal quantum computer, *Proc. R. Soc. Lond. A 400*, (1985), 97-117.

[FeC54] B.G. Farley and W.A. Clark, Simulation of self-organizing systems by digital computer, *IRE Transactions PGIT-4*, (1954), 76-84.

[FoP88] J.A. Fodor and Z.W. Pylyshyn, Connectionism and cognitive architecture: a critical analysis, in *Connections and Symbols*, S. Pinker and J. Mehler (ed.), MIT Press, Cambridge, MA, 1988, 3-71.

[Gle87] H. Gardner, *The Mind's New Science*, Basic Books, New York, 1987.

[GaJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1979.

[GeG84] S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Trans. Pat. Analysis Mach. Intell. PAMI-6*, (1984), 721-741.

[Gle87] J. Gleick, *Chaos. Making a New Science*, Viking Penguin, New York, 1987.

[HuS78] R.M. Haralick and L.G. Shapiro, The consistent labeling problem: part I, *IEEE Trans. Pat. Analysis Mach. Intell. PAMI-1(2)*, (1979), 173-184.

[HuS80] R.M. Haralick and L.G. Shapiro, The consistent labeling problem: part II, *IEEE Trans. Pat. Analysis Mach. Intell. PAMI-2(3)*, (1980), 193-203.

[Hil85] W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[HSS86] G.E. Hinton and T.J. Sejnowski, Learning and relearning in Boltzmann machines, in *Parallel Distributed Processing (Vol. I)*, D.E. Rumelhart and J.L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, 282-317.

[Hof79] D.R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*, Basic Books, New York, 1979.

[Hof85] D.R. Hofstadter, *Metamagical Themas: Questing for the Essence of Mind and Pattern*, Basic Books, New York, 1985.

[Hol87] J.H. Holland, Genetic algorithms and classifier systems: Foundations and future directions, *Second International Conference on Genetic Algorithms and their Applications*, 1987, 82-89.

[Hop82] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. 79*, (1982), 2554-2558.

[HFP83] J.J. Hopfield, D.I. Feinstein and R.G. Palmer, Unlearning has a stabilizing effect in collective memories, *Nature 304*, (1983), 158-159.

[HoS78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.

[HuZ83] R.A. Hummel and S.W. Zucker, On the foundations of relaxation labeling processes, *IEEE Trans. Patt. Analysis Mach. Intell. PAMI-5(3)*, (1983), 267-287.

[Joh83] P.N. Johnson-Laird, A computational analysis of consciousness, *Cognition and Brain Theory 6(4)*, (1983), 499-508.

[KGV83] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science 220*, (1983), 671-680.

[KiR79] L. Kitchen and A. Rosenfeld, Discrete relaxation for matching relational structures, *IEEE Trans. Systems, Man, and Cybernetics SMC-9, 12*, (1979), 869-874.

[Koh77] T. Kohonen, *Associative memory -- a system-theoretic approach*, Springer, New York, 1977.

[Mac77] A.K. Mackworth, Consistency in networks of relations, *Artificial Intelligence 8*, (1977), 99-118.

[MaF85] A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence 25*, (1985), 65-74.

[Mac87] A.K. Mackworth, Constraint satisfaction, in *Encyclopedia of Artificial Intelligence*, S.C. Shapiro (ed.), John Wiley & Sons, New York, 1987, 205-211.

[McC79] P. McCorduck, *Machines Who Think*, W.H. Freeman & Co., San Francisco, 1979.

[McP43] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys. 5*, (1943), 115-133.

[MRR53] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, Equation of state calculations by fast computing machines, *J. Chem. Phys. 21*, (1953), 1087-1092.

[Min68] M. Minsky, *The Society of Mind*, Simon & Schuster, New York, 1986.

[MiP69] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry, Expanded Edition*, MIT Press, Cambridge, MA, 1988.

[New80] A. Newell, Physical symbol systems, *Cognitive Science 4*, (1980), 135-183.

[Ros62] F. Rosenblatt, *Principles of neurodynamics*, Spartan, New York, 1962.

[RoK82] A. Rosenfeld and A.C. Kak, Iterative segmentation 'relaxation', in *Digital Picture Processing (Vol. 2)*, Academic Press, Orlando, FL, 1982, 153-188.

[RMG86] D.E. Rumelhart, J.L. McClelland and The PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition (2 vols.)*, MIT Press, Cambridge, MA, 1986.

[RHW86] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in *Parallel Distributed Processing (Vol. I)*, D.E. Rumelhart and J.L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, 318-362.

[Sha48] C.E. Shannon, A mathematical theory of communication, *Bell Sys. Tech. J. 27*, (1948), 379-423, 623-656.

[Smo86] P. Smolensky, Information processing in dynamical systems: foundations of harmony theory, in *Parallel Distributed Processing (Vol. I)*, D.E. Rumelhart and J.L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, 194-281.

[SRT87] F.F. Soulie, Y. Robert and M. Tchuente, eds., *Automata Networks in Computer Science. Theory and Applications*, Princeton U. Press, Princeton, NJ, 1987.

[Ull76] J.R. Ullmann, An algorithm for subgraph isomorphism, *J. ACM 23*, (1976), 31-42.

[Wal75] D. Waltz, Understanding line drawings of scenes with shadows, in *The Psychology of Computer Vision*, P. Winston (ed.), McGraw-Hill, New York, 1975.

[WH60] G. Widrow and M.E. Hoff, Adaptive switching circuits, in *Inst. Radio Eng., West. Elect. Show Convention, Convention Rec., Part 4*, 1960, 96-104.

[Wie61] N. Wiener, *Cybernetics: or Control and Communication in the Animal and the Machine (2nd. Ed.)*, MIT Press, Cambridge, MA, 1961.

Appendix A1

ARE--A Relaxation Simulator

1.1. INTRODUCTION

The "Alberta Relaxation Engine" (ARE) is a general software package for writing relaxation programs, such as the connectionist systems. It provides a mechanism for creating a relaxation network, interfacing to C arrays and routines, and choosing the nature of the relaxation and learning rules.

The features of ARE include:

-- stochastic or deterministic updating
-- user-defined or adaptive compatibility functions
-- Hebbian, Widrow-Hoff and back-propagation learning rules
-- hidden units
-- weight decay
-- supervised training
-- external control over the relaxation process
-- standard statistical functions (mean, variance, entropy, etc.)

Section 2 provides an introduction to the basics of ARE programming. This should get you started programming in ARE, but is not an exhaustive description of the system. A complete list of the ARE commands and their C parameter lists is given in Appendix A2.

1.2. THE BASICS

1.2.1. Getting Started

ARE is built on top of C. An interactive, interpreter-like mode, *prompt()*, is also available, which allows interactive debugging of programs. This is not a full-blown ARE interpreter, but supports a sub-set of the ARE command set. It allows the user to switch back and forth with UNIX and features a help mode. The system can be dumped at any time with *printNetwork()* and read in later with *readNet()*.

ARE operates under two general types of network architecture: linked lists (for sparsely connected nets) and adjacency matrices (for densely connected graphs). Linked list mode is the default. The adjacency

matrix option is chosen with *initWEIGHTS(num)*, which will set up the system to use the adjacency matrix WEIGHTS[][][] for a network of size num. It is the resulting index for the sub-net created. Each time *initWEIGHTS()* is called, a new completely connected sub-net is created. This will be the final connectivity of this sub-net. This initialization must be done before any weights are established. The system keeps track of the number of such sub-nets created in the global variable SIZE_SUB. If *initWEIGHTS()* is not called before the first connection is made in the network, the system will be locked into linked-list mode, and all connections will be dynamically allocated as they are created.

The main data structure is the Net[] array (all indices in ARE start at 1). Each element is a node with various fields. Net[i].a is the activation of the ith node. Net[] has NET_SIZE elements. NET_SIZE is a constant, which must be defined in the ARE program file (in which the ARE system, *relax.h*, is included). NET_SIZE is the maximum allowable size of the system. There must also be a definition of MAX_SUB, for the maximum number of sub-nets, when the adjacency matrix option is in use. The definitions precede the includes. So a program might start with (note that a MAX_SUB of 0 is used for the linked-list option):

```
#define NET_SIZE 1000
#define MAX_SUB 0
#include "relax.h"
```

The ARE program file must also contain the function *GATE()* to provide external (non-relaxation) input for the nodes. This routine can be anything at all. If no external control is desired, use:

```
#include "no_control.h"
```

Once the network is set up, it is initially empty. Thus SIZE, which represents the number of nodes in the system, is initially zero. New nodes are created with the command *new()*. An initial activation value may be passed as a parameter. The nodes are entered into Net[] in the order they are created, starting with Net[1]. The following code, then, would initialize ten new nodes:

```
for (i=1; i<=10; i++)
  new(0.0);
```

Links between nodes are set up with the *link()* and *connect()* routines. *link()* creates a one-way con-

nection and *connect()* is two-way. An initial weight for the connection must also be specified. The following code would create an empty (zero) one-way link from unit #1 to all other units:

```
for (i=2; i<=SIZE; i++)
  link(i,1,0.0);
```

The activation value for a node can be accessed at any time with *a()*. So a(5) returns the activation of unit #5. *p()* is a similar function that returns the probability with which the unit is being fired, rather than the actual current state of the unit (for deterministic updating, then, *a()* and *p()* are identical).

The relaxation is performed with the *learn()* routine. To relax only a specific sub-net, the range of nodes should be selected with the *select* routine. The following code selects units #1 through #10 as the current relaxation nodes, and does 100 iterations on them:

```
select(1,10);
learn(100);
```

Other commands are provided for more convenient relaxation. *Jlearn()*, for instance, will keep learning until the global energy of the system has reached equilibrium. This occurs as soon as the ENERGY of the system has been unchanged, or has cycled between two points, or has not gone below the minimum energy so far, for ENERGY_CHANGE iterations. ENERGY_CHANGE must be set, and is essentially the number of iterations after which you are confident equilibrium is reached.

3.4.5. Learning

ARE automatically uses a back-propagation rule for updating weights (weight adaptation can be set and reset with the system flag LEARN). When there are no hidden (non-clamped) units, the back-propagation rule becomes the simpler delta rule. Simple Hebbian updating will result if there are no units clamped. Clamping is done with the *clamp()* command. The following code clamps all units from #1-10 with a pattern of values, and freezes the system to equilibrium:

```
for (unit i=1; i<=10; i++)
  clamp(value_to_be_learned_for_unit_i);
ENERGY_CHANGE = 10;
learn();
```

After training is complete, learning should be shut off (LEARN = 0). Now values can then be presented to the previously clamped units that were intended as inputs. First, the input nodes must be specified as such with the *fix()* command. This sets the node as an input node, meaning it must always remain fixed at a certain value, set by the environment, rather than the relaxation process. The *present()* routine is then used to present a particular input value to the fixed input node.

Following our previous example of clamping, assume that units #2-5 were inputs and units #6-10 were outputs. Both the input bank and the output bank were previously clamped during training. Now we turn off the learning, and fix nodes #2-5 as input units. We then present this input bank with new values. Freezing the system will produce values on the output nodes, representing the best completion of the learned pattern:

```
LEARN = 0;
for (i=0; i<=5; i++)
  fix(i);
for (i=2; i<=5; i++)
  present(i,new_value_for_unit_i);
freeze();
print("Output values are: ");
for (i=6; i<=10; i++)
  print("%f ",x[i]);
```

The clamping, fixing and presenting of nodes can be undone with the *unclamp()*, *unfix()* and *unpresent()* routines:

```
unclamp(2,10);
unfix(2,5);
unpresent(2,5);
```

1.2.3. Stochastic Updating

ARE uses Boltzmann Machine updating, and so there must be a global system temperature (TEMPER), as well as an associated annealing schedule. There are three annealing routines, *sched()*, *gemanl()*, and *klrk()*. To define the annealing schedule, set the appropriate flag: SCHED, GEMAN, or KIRK. The schedule can be changed at any time.

gemanl() gives the theoretically optimal solution when initial temperature is equal to the number of nodes in the system: T0 = SIZE, and the annealing rate is 1: ANNEAL_RATE = 1. This is way too slow for most real world problems. ANNEAL_RATE should thus be set higher than one. If ANNEAL_RATE = 1000, for example, the optimal annealing schedule will be sampled at every 1000 iterations.

sched() allows the user to create his own annealing schedule suited to the application. It reads the temperatures in from a data file. The most useful annealing schedule is *klrk()*, which gives exponential Kirkpatrick annealing. This schedule must be set with the *set_klrk()* command: set_klrk(T0,RDM,p1,p2). T0 is the desired initial temperature. RDM is one of the three characters 'R'. The schedule must be computed from the two remaining parameters, which are given as p1 and p2. R is the rate of change of the temperature. *I* is the desired number of iterations for the freezing. *M* is the minimum temperature allowed. At iteration *i*, the global parameter TEMPER will be updated to $T0(R^i)$.

Although ARE uses stochastic updating (via annealing), deterministic updating can be approximated by setting the temperature to very low (TEMPER = 0.000) and shutting off the annealing mechanism with *stop_anneal()*.

1.2.4. Matrix Routines

1.2.4.1. The ARE Matrix

The above commands are all you need to use ARE. However, there are many other commands that allow faster and more convenient programming. These are listed in Appendix A2. Particularly useful are the matrix functions. There is a special structure in ARF called the *matrix*, which is slightly different from a C array; it is indexed starting at 1 (not 0) and it must be allocated as an ARE matrix. The matrix must be declared, along with your other variables, using *MatrixN()*. Allocation can be done at any time, using *alloc-MatrixN()*. The N is a number from 1 to 4 representing the dimensions of the matrix. The following code declares and allocates a 10 x 25 two-dimensional matrix of type float (most ARE matrix functions expect floating point matrices, although it is not required):

```
Matrix2(B,float);
...
allocMatrix2(B,float,10,25);
```

1.2.4.2. Matrix Routines

These matrix routines are mostly for two-dimensional matrices. A three dimensional matrix, then, should be treated like a list of two-dimensional matrices. ARE matrices have special fields (hidden from the user) that keep track of the relocation (*Nod[]*) nodes that correspond to the matrix elements. These routines are used to access this information: *START(A)* returns the Nod[] unit number that matrix A starts at. *END(A)* returns the ending location. *H_END(A)* returns the ending location for the hidden bank (if any) attached to matrix A (the hidden bank starts immediately following A, at END(A)+1).

Most matrix operations get the dimensions passed to them (m,n). Matrices are stored as in C, in row-column format. *MatIndex()* gives the location in Nod[] of one element of a matrix. The *posMatrix()* and *posNod()* routines convert one-dimensional matrix locations and one-dimensional Nod[] locations back and forth.

A matrix is associated with a range of unit numbers in Net[] with the *initMatrix()* routine. The firing probabilities of the matrix can be retrieved from the relaxation network at any time with *getMatrix()*. Other routines (such as *selectMatrix()*, *connectMatrix()*, *clampMatrix*, etc.) are just matrix versions of many basic ARE commands.

Appendix A2

ARE Command List

All parameters are written in bold. ARE commands are in *italics*. Global parameters are in CAPITALS (see *set<PARAM>* for a description of these parameters).

adjust_best()
Adjust current activations (Net[].a) and probabilities (Net[].p) to fit the best network configuration so far encountered. The best configuration is the one with lowest ENERGY so far and is stored in Net[].best at issue of the *save_best()* command.

allocMatrix<N>(Mat,type,d1,...)
Allocates memory space for an N-dimensional matrix of type type and dimensions (d1,d2,etc.). So "allocMatrix3(A,float,5,10,10)" allocates room for A, a three dimensional 5x10x10 matrix.

anneal_sweep()
Relax all selected nodes for one complete annealing cycle.

clamp(n,v)
int n;
float v;
Clamp value v onto node Net[n], by placing it on the training queue. While LEARN is set, the system will clamp the current value at the head of the queue onto Net[n]. Every SWITCH_TRAIN iterations, this value will go to the tail of the queue and the next value will be clamped.

clampMatrix(Mat,m,n)
float **Mat;
int m,n;
Clamps *mxn* matrix Mat onto the relaxation network, Net.

connect(n1,n2,w)
int n1,n2;
float w;
Connect nodes Net[n1] and Net[n2] by a bi-directional connection with weight w.

connectGroup(i1,i2,j1,j2)
int i1,i2,j1,j2;
Connect all nodes between Net[i1] and Net[i2] with all nodes between Net[j1] and Net[j2]. Connections are bi-directional.

connectMatrix(Mat1,Mat2)
float **Mat1,**Mat2;
Connect all nodes of the relaxation network, Net, that are associated with matrix Mat1 with all nodes associated with Mat2. Connections are created using *connect()* and are bi-directional.

cut_off(n1,n2)
int n1,n2;
Cut off nodes Net[n1] to Net[n2] from all environmental input, including *clamped* values and *presented* values.

delete(n)
int n;
Delete node Net[n] from the net by de-activating it. This means only that it will no longer take part in the relaxation process. There is still a place in the network for it and no re-numbering of any nodes is performed. The node is still there, and can be re-activated with *undelete*().

end(*blist*)
float **blist;
Returns the ending position in Net of the associated matrix blist.

fix(n1,n2)
int n1,n2;
Fix nodes Net[n1] to Net[n2] so that they must remain fixed on a single value, which was presented to them with *present*().

forcew(n1,n2,new_w)
int n1,n2;
float new_w;
Sets the weight on connection from Net[n2] to Net[n1] to the new value new_w. If a connection does not exist, one is created.

freeze()
Keeps iterating all selected nodes until the system freezes into equilibrium. This occurs as soon as the ENERGY of the system is unchanged, or has cycled between two points, or has not gone below the minimum so far, for ENERGY_CHANGE iterations.

getbestiter(*savblist*,*blist*,*m*,*n*)
float **savblist,**blist;
int m,n;
Gets values from the best iteration of the relaxation network (Net[].p) that are associated with rows matrix blist and places them into the corresponding positions of *savblist*, which must be at least as large as blist. The "best" iteration is defined as the one with the lowest ENERGY and is stored in Net[].best at the issue of the *store_best*() command.

getiter(*blist*,*m*,*n*)
float **blist;
int m,n;
Gets values from the relaxation network (Net[].p) and places them into associated values in the rows matrix blist.

help()

Help sub-system. Used in the interactive *prompt*(mdin,1) mode. The user is given a complete list of user-visible commands and repeatedly prompted to enter a command name to get a brief description, or to enter the word "commands" to re-display the list of commands. The word "exit" returns the user to the *prompt*() mode.

init()

Initialize the random number generator and global system parameters.

initCmvin(Mat,m,n,com)
float **Mat;
int m,n,com;

Initialize the *mxn* matrix Mat (must already be allocated with *allocMatrix2*()) by associating each element with a corresponding element of the relaxation network, Net. If com is set, the matrix will be automatically completely connected.

initMatrix(Mat,m,n,k)
float **Mat;
int m,n,k;

Initializes a bank of k hidden units associated with *mxn* matrix Mat (must already be initialized with *initMatrix*()).

initWEIGHTS(size)
int size;

Initializes the *size* x *size* array WEIGHTS, to hold the weights on the connections between nodes, where *size* is the size of the entire network. Use of the weights (accessing, saving, modifying, etc.) is the same as before. Direct access via the WEIGHTS array is not recommended for the sake of consistency. This command should be issued before any weight values are assigned. If it is not, the system defaults to a pointer based representation for the weights. Sparsely connected networks should use the default, but dense nets are better off with the array representation. The functioning of ARIS is the same in either case. Only the performance of the system is affected by this distinction.

iter()
int i;

Perform i iterations on all selected nodes. Uses *relax*().

iterAct()
int i;

Perform i iterations on all selected nodes. Display activations. Uses *relaxAct*().

link(n1,n2,w)
int n1,n2;
float w;

Link node Net[n1] to node Net[n2], with weight w.

linkGroup(i1,j1,i2,j2)
int i1,j1,i2,j2;

Link all nodes between Net[i1] and Net[j1] to all nodes between Net[i2] and Net[j2]. Links are one-way from Net[i1..j1] to Net[i2..j2].

linkMatrix(Mat1,Mat2)
float **Mat1,**Mat2;
Link all nodes of the relaxation network, Net, that are associated with matrix Mat2 to all nodes associated with Mat1. Links are created using *link*() and are one-way from Mat2 to Mat1.

load(file_name)
char *file_name;
Read the ARE commands in from file file_name and execute them. Usually used in the interactive *prompt*(stdin,1) mode.

new(a)
float a;
Initialize a new node and place it at the end of the relaxation network, at Net[SIZE+1] (and increment SIZE). Activation and probability of firing (Net[].a and Net[].p) are initialized to a.

present(n,v)
int n;
float v;
Present the value v to node Net[n]. Used with *fix*(), which causes the node to remain fixed on a single value. This means that Net[n] will always have activation v, no matter what inputs it has, as long as the node is *fixed*. This is different from *clamp*() in that clamped values are used during training as part of the weight updating, when LEARN is set. Presented values are normally used when LEARN is not set, during running of the system. For example, during training, an input bank and an output bank are both clamped, while hidden units are left free. During recognition, with LEARN *off*, the input units are *fixed* and *presented* with an input pattern, and the output units are left free.

presentMatrix(Mat,m,n)
float **Mat;
int m,n;
Presents elements of *mxn* matrix Mat to the relaxation network, Net, using *present*().

print_stats(n)
int n;
Prints statistics on node Net[n], including means (Net[n].mean), standard deviation (Net[n].dev) and variance (Net[n].var). Stats are updated every UPDATE_STATS iterations.

printDataFile(file_name)
char *file_name;
Print out all the data in the network to the file file_name. This is written in a form readable by *readFile*().

printSomeFile(file_name)
char *file_name;
Print out all the data in the *selected* nodes in the network to the file file_name. This is written in a form readable by *readFile*().

printFile()
Print out a user-readable description of all nodes in the network Net.

printInit()
Print out a user-readable description of all selected nodes in the network, Net.

prompt(file_ptr,inter)
FILE *file_ptr;
int inter;
Prompts for ARE commands from file indicated by pointer file_ptr. If inter is set, *prompt()* assumes an interactive session and will print prompts, etc. For this, file_inter should be set to stdin.

query(file_ptr,file)
FILE *file_ptr;
int file;
Query the user for the various system parameters. If file is set, input is meant to be read from the file indicated by file_ptr and the prompting is turned off. For user entry, file_ptr will normally be stdin.

query_sched(file_ptr,file)
FILE *file_ptr;
int file;
Query the user for the various parameters needed for the annealing schedule. Supports Kirkpatrick, Geman, and user-defined annealing schedules. If file is set, input is meant to be read from the file indicated by file_ptr and the prompting is turned off. For user entry, file_ptr will normally be stdin.

quit()
Quits and leaves an ARE *prompt()* session.

readNet(file_name)
char *file_name;
Read the nodes in from file file_name, which were written in the form of *printOneNet()* and *printOneInit()*.

relax()
Perform one iteration on all selected nodes.

relaxInit()
Perform one iteration on all selected nodes. Display activations.

relaxMatrix(Mat,m,n,num)
float **Mat;
int m,n,num;
Relax for num iterations only the nodes of the relaxation network, Net, that are associated with elements of the zero matrix Mat.

restart()
Restart the system, initializing with *Init()* and also erasing the current relaxation network.

restart_sched()
Restart the annealing schedule.

select(n1,n2)
int n1,n2;
Select nodes from Net[n1] to Net[n2] as the active nodes to be relaxed. The default is all nodes selected.

selectMatrix(Mat)
float **Mat;
Select all nodes of the relaxation network, Net, associated with matrix Mat as the active nodes to be relaxed. Uses select().

selectNet()
Select the entire network Net to be relaxed (this is the default, so this command will generally be used to undo a previous select()).

set_bit(T0,RBM,p1,p2)
float T0;
char *RBM;
float p1,p2;
Sets the annealing schedule according to the schedule of Kirkpatrick, et al. T0 is the initial temperature at the start of annealing. RBM is one of three characters: 'R', 'T' or 'M'. This indicates which of these three parameters for the schedule must be computed from the other two, which are given as p1 and p2. R is the ratio T1/T0, where T1 is the temperature after the first iteration. I is the number of iterations the system takes to reach the minimum temperature M. At iteration *i*, the global parameter TEMPER will be updated to $T0(R^i)$.

setbias(n,b)
int n;
float b;
Set the bias (negative threshold) of node n, or Net[n].bias to b.

setMat(Mat,start,end,hidden_end)
float **Mat;
int start,end,hidden_end;
Sets (or resets) the association between matrix Mat and the relaxation network, Net. Mat now starts at Net[start] and ends at Net[end]. The hidden units for Mat, if any, start at Net[end+1] and end at Net[hidden_end].

setThreshold(n,t)
int n;
float t;
Set the threshold (negative bias) of node n to t. This is the same as setting the bias, Net[n].bias to -t.

set<VARIABLE>(f)
Sets the global system parameter to value f. <VARIABLE> is the name of the parameter. For example, setWt(-1.0) sets the default initial weight parameter Wt to -1.0. Global parameters are:

ITR: Current number of iterations that have been performed.
TEMPER: Computational temperature.
T0: Initial temperature at start of annealing.
ENERGY: The current global energy of the system.

EQUIL: The number of iterations allowed with unchanged energy.
EQUIL2: The number of iterations allowed with unchanged 2-cycling of energy.
ENERGY_CHANGE: The number of iterations allowed with energy greater than minimum.
WI: Default initial weight of connections.
SIZE: The current size of the relaxation network.
START: The position of the first relaxed node in the relaxation network.
END: The position of the last relaxed node in the relaxation network.
ADAPT_TH: Set to 1 if adaptive thresholds are desired, 0 otherwise.
LEARN: Set to 1 if weights are to be updated according to hook propagation rule.
WEIGHT_ADJUST: weights are adjusted (hook propagation) every WEIGHT_ADJUST iterations.
SWITCH_TRAIN: the training value is switched to the next one every SWITCH_TRAIN iterations.
UPDATE_STATS: Statistics are updated every UPDATE_STATS iterations.
SCHED, GEMAN, KIRK: Indicates whether corresponding schedule is active.

```
setW(n1,n2,new_w)
int n1,n2;
float new_w;
```
Sets the weight on connection from Net[n2] to Net[n1] to the new value new_w. If a connection does not exist, an error code of 0 is returned.

```
save(ldst)
float **ldst;
```
Returns the starting position in Net of the annealed matrix Mint.

start_anneal()
Start updating stochastically. Annealing schedule must be set with *query_sched()* or *set_kirk()*.

stop_anneal()
Stop annealing (i.e. start deterministic updating of nodes). Leaves the annealing schedule, if any, unchanged.

save_best()
Saves the best network configuration so far encountered (best means lowest ENERGY) into Net[].best.

```
mbb()
int i;
```
Relax all relaxed nodes for i iterations. Display the table of weights. With no arguments, *mbb()* just prints out the current weight table.

training_sweep()
Relax all relaxed nodes until the system has cycled through all training values that were clamped onto Net. The system will switch to the next training value every SWITCH_TRAIN iterations.

```
unclamp(n1,n2)
int n1,n2;
```
Unclamp nodes Net[n1] to Net[n2] by removing the head of the training queue. Must be performed once for each training value to completely unclamp the node.

undelete(n)
int n;
Re-activate node Net[n], which was previously de-activated with *delete()*.

unfix(n1,n2)
int n1,n2;
If any nodes between Net[n1] and Net[n2] are fixed with *fix()*, undo this, making the node once again free to take on its own value.

unix()
Takes the user temporarily out of ARE and into a UNIX shell. Type *exit* within UNIX to return to ARE.

unprocess(n)
int n;
If node Net[n] has been previously processed with a value, remove this processed value.

w(n1,n2)
int n;
Returns the weight from node Net[n2] to Net[n1].

weight_sweep()
Relax all selected nodes until a weight adjustment needs to be done. Weight adjustments are performed every WEIGHT_ADJUST iterations. LEARN must be set.