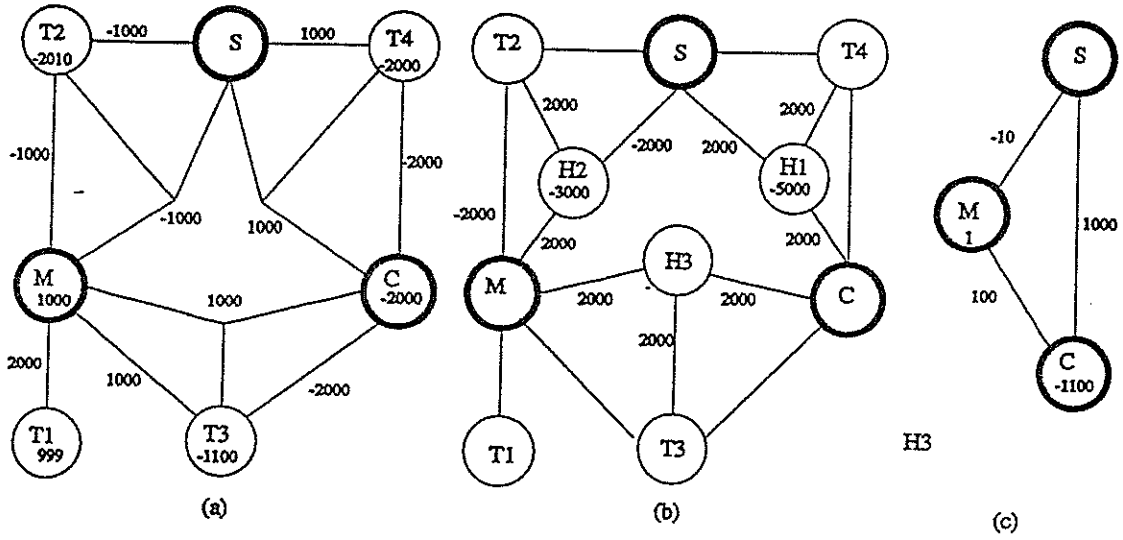

Figure 6.3: Equivalent symmetric networks for the meeting example (the numbers in the circles are thresholds): (a) cubic; (b) quadratic; and (c) quadratic for the simple conversion (no naming).

Representing SNs as Penalty Logic Formulas. This subsection shows that it is possible to describe efficiently any network by a penalty logic formula. The motivation here is to demonstrate that penalty logic is an efficient and compact language for specifying symmetric networks.

THEOREM **6.3.2** EVERY ENERGY FUNCTION E IS STRONGLY EQUIVALENT TO SOME PLOFF ψ; i.e., THERE EXISTS A CONSTANT c SUCH THAT $rank_E = Vrank_\psi + c$.

Construction:
The following algorithm generates a strongly equivalent PLOFF from an energy function.

1. Eliminate any hidden variables from the energy function, using the algorithm of Section 3.3.2.

2. The energy function (with no hidden variables) is now brought into a sum-of-products form and is converted into a PLOFF in the following way:
   Let $E(\vec{x}) = \sum_{j=1}^{m} w_j \prod_{n=1}^{k_j} x_{j_n}$ be the energy function.
   We construct a PLOFF $\psi = \{< -w_i, \bigwedge_{n=1}^{k_i} x_{i_n} > |\ w_i < 0\} \overset{*}{\cup} \{< w_l, \neg \bigwedge_{n=1}^{k_l} x_{l_n} > |\ w_l > 0\}$.

The formula that is generated is strongly equivalent to the original energy function and the network. The size of the formula is linear in the number of the connections of the original network.

Proof:
To show $Vrank_\psi = rank_E + c$:

$$Vrank_\psi(\vec{x}) = \sum_{w_i<0 \wedge \neg(\vec{x} \models \bigwedge X_{i_n})} -w_i + \sum_{w_l>0 \wedge \neg(\vec{x} \models \neg(\bigwedge X_{l_n}))} w_l$$

$$= - \sum_{w_i<0 \wedge \vec{x} \models \neg(\bigwedge X_{i_n})} w_i + \sum_{w_l>0 \wedge \vec{x} \models (\bigwedge X_{l_n})} w_l$$

$$= - \sum_{w_i<0} w_i + \sum_{w_i<0 \wedge \vec{x} \models \bigwedge X_{i_n}} w_i + \sum_{w_l>0 \wedge \vec{x} \models (\bigwedge X_{l_n})} w_l$$

$$= \sum_{w_i<0} w_i \prod_n X_{i_n} + \sum_{w_l>0} w_l \prod_n X_{l_n} - \sum_{w_i<0} w_i = rank_E + c$$

□

EXAMPLE **6.3.7** Looking at the network of Figure 6.2, we would like to describe this network as a PLOFF.
The energy function is:

$$E = -1000NQ - 1000NR + 10RP - 10QP - 1000N + 10Q.$$

The negative terms are:

$$< 1000, N \wedge Q >, < 1000, N \wedge R >, < 10, Q \wedge P >, < 1000, N > .$$

The positive terms are:

$$< 10, \neg R \vee \neg P >, < 10, \neg Q > .$$

The final PLOFF is therefore:

$$< 1000, N \wedge Q >, < 1000, N \wedge R >, < 10, Q \wedge P >, < 1000, N >< 10, \neg R \vee \neg P >, < 10, \neg Q > .$$

Note that as is usually the case with reverse-compilation, the formula we get is not very meaningful; however, it is clear that a compact description exists for every network.

6.4. A Connectionist Inference Engine

Suppose we have a background PLOFF ψ, an evidence PLOFF e, and a query which is a strict standard WFF φ. We would like to construct a connectionist network to give one of the possible three answers: 1) $\psi \overset{*}{\cup} e \models \varphi$; 2) $\psi \overset{*}{\cup} e \models (\neg\varphi)$; or 3) both $\psi \overset{*}{\cup} e \not\models \varphi$ and $\psi \overset{*}{\cup} e \not\models (\neg\varphi)$ ("ambiguous").

Intuitively, our connectionist engine is built from two sub-networks, each of which is trying to find a satisfying model for $\psi \overset{*}{\cup} e$. The first sub-network is biased to search for a preferred model which also satisfies φ, whereas the second sub-network is biased to search for a preferred model which satisfies $\neg\varphi$. If two such models exist, then we conclude that φ is "ambiguous" ($\psi \overset{*}{\cup} e$ entails neither φ nor $\neg\varphi$). If no preferred model also satisfies φ, we conclude that $\psi \overset{*}{\cup} e \models \neg\varphi$, and if no model also satisfies $\neg\varphi$, we conclude that $\psi \overset{*}{\cup} e \models \varphi$. For simplicity let us first assume that the evidence e is a strict conjunction of literals and that φ is a single atomic proposition. Later we will describe a general solution.

To implement this intuition we first need to duplicate our background knowledge ψ and create its copy ψ' by naming all the atomic propositions A using A'. For each atomic proposition A that might participate in a query, we then add two more propositions "$QUERY_A$" and "$AMBIGUOUS_A$". $QUERY_A$ is used to initiate a query about A; it will be externally clamped by the user, when he or she inquires about A. The unit "$AMBIGUOUS_A$" represents the answer of the system. It will be set to TRUE iff we can conclude neither that ψ entails A nor that ψ entails $\neg A$.

Our inference engine can be therefore described (in the language of penalty logic) by:

ψ	searches for a preferred model of ψ that satisfies also A
$\overset{*}{\cup}\psi'$	searches for a preferred model of ψ that satisfies also $\neg A$
$\overset{*}{\cup}\{< \epsilon, (QUERY_A \rightarrow A) >\}$	bias ψ to search for a model that satisfies A
$\overset{*}{\cup}\{< \epsilon, (QUERY_A \rightarrow (\neg A')) >\}$	bias ψ' to search for a model that satisfies $(\neg A')$
$\overset{*}{\cup}\{< \epsilon, (A \wedge \neg A') \rightarrow AMBIGUOUS_A >\}$	if two satisfying models exist that do not agree on A, we conclude "$AMBIGUOUS$"
$\overset{*}{\cup}\{< \epsilon, (A \leftrightarrow A') \rightarrow (\neg AMBIGUOUS_A) >\}$	if despite the bias we are unable to find two such satisfying models we conclude "$\neg\ AMBIGUOUS_A$"

Using the algorithm of Theorem 6.3.1, we generate the corresponding network. The network that is generated for the Nixon example is shown in Figure 6.4.

To initiate a query about A the user externally clamps the unit $QUERY_A$. This causes a small positive bias ϵ to be sent to unit A and a small negative bias $-\epsilon$ to be sent to A'. Each of the two sub-networks ψ and ψ', searches for a global minimum (a satisfying model) of the original PLOFF. The bias (ϵ) is small enough so it does not introduce new global minima for each of the subnetworks. It may, however, constrain the set of global minima. If a satisfying model that also satisfies the bias exists, then this model is in the new set of global minima of ψ. The new set of global minima is the set of all preferred models of ψ that also satisfy the query. If no preferred model also satisfies the

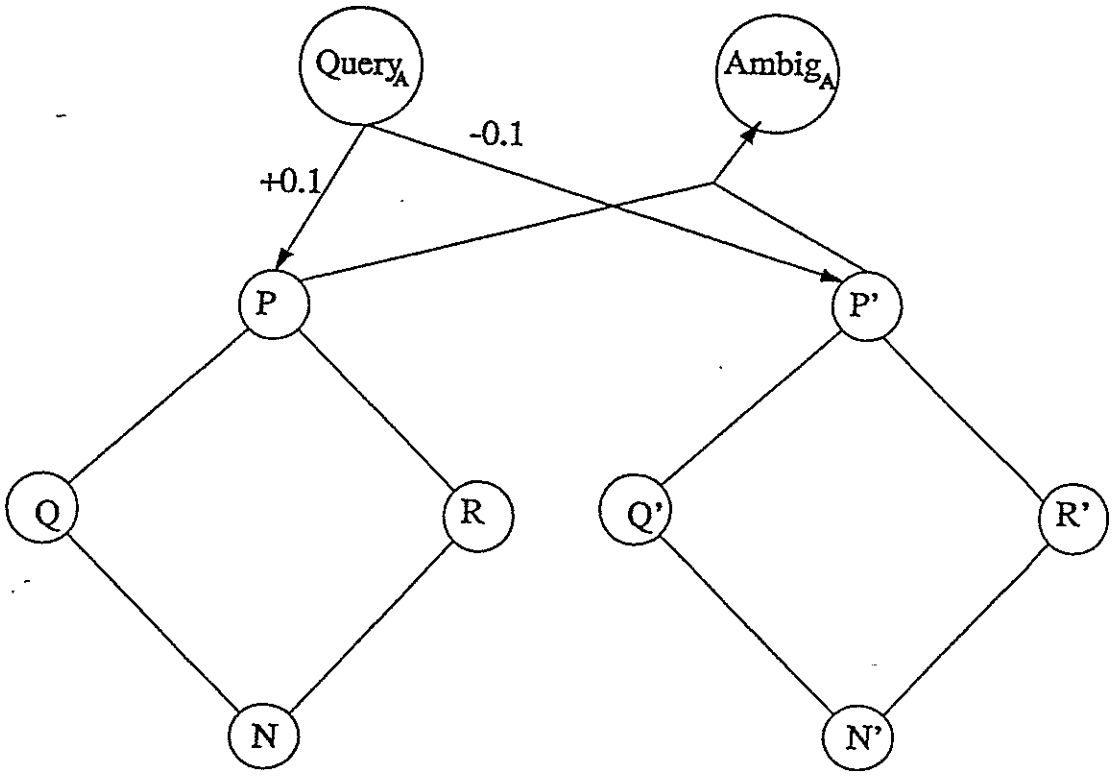

Figure 6.4: Inference network for the Nixon diamond case: the two rings represents two similar subnetwork: One searches for a preferred model that satisfies the query and the other searches for a preferred model the falsifies the query.

query then the set of global minima is unaffected by the bias and the network searches for one of those models that does not satisfy A.

The network therefore tries to find models that also satisfy the bias rules. If it succeeds, we conclude *"AMBIGUOUS"*, otherwise we conclude that all the satisfying models agree on the same truth value for the query. The $AMBIGUOUS_A$ proposition is then set to "false", and the answer to whether $\psi \models \varphi$ or $\psi \models \neg\varphi$ can be found in the unit A. If A is "true" then the answer is $\psi \models \varphi$, since A holds in all satisfying models. Similarly, if A is false, we conclude that $\psi \models \neg\varphi$.

When the evidence is a strict conjunction of literals, the user may add it to the background network simply by clamping the appropriate atomic propositions whenever new evidence is observed. In the general case we need to combine arbitrary evidence e and an arbitrary query φ. We do this by building an inference network for $\psi \overset{*}{\cup} e \overset{*}{\cup} \{< M, A \leftrightarrow \varphi >\}$ and in querying about A, a new atomic proposition. Adding evidence to a network ψ may be done simply by computing the energy terms of the evidence and then updating the weights of the network.

6.5. Related Work

6.5.1. Connectionist Systems

Derthick [Derthick 88] observed that weighted logical constraints (which he called "certainties") can be used in massively parallel architectures. Derthick translated those constraints into special energy functions and used them to implement a subset of the language KL-ONE. The approach described in this chapter has a lot of similarities to his system. Looking at his reduction from logic to energy functions (Derthick uses a different architecture, different energy functions and no hidden units), there are, however, several basic differences: 1) Derthick's "mundane" reasoning is based on finding a most likely *single* model; his system is never skeptical. The system described in this chapter is much more cautious and closer in its behavior to symbolic nonmonotonic systems, described in recent literature (see Section 6.5.2). 2) The nonmonotonic system that is described here can be implemented with standard low-order units, using relatively well-studied architectures such as Hopfield networks, Harmony, MFT or Boltzmann machines. It is possible therefore to take advantage of the hardware implementations as well as of the learning algorithms that were developed for these networks. 3) Formal proofs of two-way equivalence are given so that every network can be described as a PLOFF and not just the reverse.

Another connectionist nonmonotonic system is [Shastri 88]. It uses evidential reasoning based on maximum likelihood to reason in inheritance networks. My approach is different; Shastri's system is restricted to inheritance networks,[6] and propagates probabilities along paths of the network. Shastri's system needs to have a priori knowledge (e.g., conditional probabilities) that is not usually available; however, once conditional probabilities are known, the system can perform probabilistic reasoning efficiently and reliably. In contrast, reasoning with penalty logic is intractable; however, time is traded with reliability.

We may look at penalty logic as one of the layers of abstraction that is needed between descriptions of high-level cognitive processes and low-level neural implementations. Thus, penalty logic may be seen as a first level of abstraction that is higher than the neural implementation (see [Barnden 91] for a nice discussion on the multi-span approach). Using the language described in this chapter we can map several of the symbolic systems that will be mentioned in the next subsection into penalty logic, and then compile them into symmetric networks (possibly by sacrificing efficiency) [Pinkas 91e].

6.5.2. Symbolic Systems

Penalty logic is along the lines of work done in preferential semantics [Shoham 88] and is related in particular to systems with preferential semantics that use ranked models, like

[6] Propositional formalisms may be extended to handle predicates and variables by looking at the atomic propositions as predicates with free variables which are taken from a predefined list of variables. In inheritance systems only one variable (which is externally instantiated) is used.

[Lehmann, Magidor 88], [Lehmann 89] or [Pearl 90]. Lehmann and Magidor's results about the relationship between rational consequence relations and ranked models can be applied to our paradigm. A strict consequence relation induced by a PLOFF ψ is a binary relation between strict evidence and a strict conclusion. It is therefore a set of pairs $R_\psi = \{< \varphi', \varphi > |\ \varphi' \overset{\psi}{\models} \varphi\}$, where both φ' and φ are strict WFFs. Lehmann and Magidor defined a *rational* consequence relation as one that satisfies certain conditions (inference rules) and proved that a consequence relation is rational iff it is defined by some ranking function. As a result, we may conclude a rather strong conclusion for the system given in this chapter. For every rational consequence relation we can build a ranked model and implement it in a symmetric network. Also, any symmetric network can be viewed as implementing some rational consequence relation. We can therefore be sure that every implementation of our connectionist inference engine induces a rational consequence relation.

One system of ranked models that can be reduced directly to penalty logic is [Goldszmidt, Pearl 91]. This system actually computes the penalties for a given conditional knowledge-base using maximal entropy considerations (the user does not specify any penalty). The system uses the same ranking function as the one described in this article; i.e., summing the penalties of violated beliefs.

Penalty logic has some similarities with systems that are based on priorities given to beliefs. One such system [Brewka 89] is based on levels of reliability. Brewka's system for propositional logic can be mapped (approximately) into penalty logic by selecting large enough penalties. Systems like [Poole 88] (with strict specificity) can also be implemented using our architecture, and as in [Goldszmidt, et al. 90], the penalties can be generated automatically. Another system that is based on priorities is system Z^+ [Goldszmidt, Pearl 91] where the user does specify the penalties, but there is a "ghost" that changes them so that several nice properties hold (e.g. specificity). Penalty logic can only approximate priority systems by assigning scaled-penalties.[7] Every conclusion that is entailed in a priority system like system Z^+ will also be entailed by the approximating penalty logic knowledge base. However, some conclusions that are ambiguous in a priority system may be drawn decisively in penalty logic. In this sense penalty logic can be considered as bolder (less cautious) than those which are based on priorities.

For example consider the "penguins and the wings" case [Goldszmidt, Pearl 91]. We are given the following defaults: birds fly; birds have wings; penguins are birds and penguins do not fly. Many systems based on priorities (like Z^+) will not be able to conclude that penguins have wings (since "birds fly" is defeated). Penalty logic in contrast will conclude according to our intuition; i.e., that penguins do have wings despite the fact that penguins do not fly. The reason for this intuitive deduction is that penalty logic considers the models where penguins do not fly but have wings to be more "normal" than models where penguins do not fly and have no wings (as in

[7] The penalties are scaled so that there is no subset of low-priority assumptions whose sum exceeds the penalty of a higher priority.

[Goldszmidt, et al. 90]). Priority-based systems will usually be ambiguous since they don't have such preference.[8]

For another example, consider the Nixon case (Example 6.2.1) when we add to it: $< 1000, N \rightarrow FF >$ and $< 10, FF \rightarrow \neg P >$ (Nixon is also a football fan and football fans tend not to be pacifist). Most other nonmonotonic systems will still be skeptical about P (e.g., [Touretzky 86], [Loui 87], [Geffner 89] [Pearl 90], [Lehmann 89]). Our system boldly, and in contrast with intuition, decides $\neg P$ since it is better to defeat the one assumption supporting P than the two assumptions supporting $\neg P$. We can correct this behavior ad-hoc[9] by multiplying the penalty for $A \rightarrow P$ by two. In general, however, such ad-hoc treatment will not always help.

Because we do not allow for an arbitrary partial order ([Shoham 88] [Geffner 89]) among the models, there are other fundamental[10] problematic examples where the system proposed (and all systems with ranked models semantics) boldly concludes, while other systems are skeptical (these are cases where the intuition tells us that skepticism is the right behavior). The following is an example for which we may have an intuition that cannot be represented with any ranking function.

EXAMPLE **6.5.1** Assume the following defeasible rules: $A \rightarrow D$, $B \rightarrow \neg D$ and $C \rightarrow \neg D$.
The intuition we might want states that:
Given A, C, D, we should conclude $\neg B$; therefore, $rank(A\bar{B}CD) < rank(ABCD)$.
Given A, B, C, we should conclude that D is ambiguous; therefore, $rank(ABCD) = rank(ABC\bar{D})$.
Given $A, C, \bar{D}$ we should conclude that B is ambiguous; therefore, $rank(ABC\bar{D}) = rank(A\bar{B}C\bar{D})$.
Given $A, \bar{B}, C$ we should conclude that D is ambiguous; therefore, $rank(A\bar{B}C\bar{D}) = rank(A\bar{B}CD)$.
This is a contradiction since $rank(A\bar{B}CD) < rank(A\bar{B}CD)$. Thus, the intuition as stated by the examples cannot be implemented by any ranked model.

6.6. Summary

The chapter introduced penalty logic and showed efficient magnitude-preserving transformations between its sentences and SNs. Penalty logic may be used as a framework for defeasible reasoning and inconsistency handling. Several systems can be mapped into this paradigm and therefore suggest settings of the penalties. When the right penalties are given, penalty logic features a nonmonotonic behavior that (usually) matches our intuition. It is possible to show, though, that some intuitions cannot be expressed as ranking functions.

[8] If the priority is given explicitly, then the problem may be solved ad hoc by assigning higher priority to "birds have wings" than to "birds tend to fly". In systems I mentioned, priorities are derived from the knowledge-base and this distinction between the two properties of birds is not made.

[9] A network that learns may adjust the penalties and thus develop its own intuition and nonmonotonic behavior.

[10] Hector Geffner (private communication).

A strong equivalence between sentences of penalty logic and symmetric networks is formally proved. This two-way equivalence serves two purposes. 1) We can translate a sentence of penalty logic into an equivalent network (this serves the basic construction of our inference engine). 2) Any symmetric network can be described by penalty logic sentences.[11] The logic may thus be used as a specification language and gives another clarifying look at the dynamics of such networks.

Several equivalent high-level languages can be used to describe SNs: 1) quadratic energy functions; 2) high-order energy functions with no hidden units; 3) propositional logic, and finally 4) penalty logic. All of these languages are expressive enough to describe any SN, and every sentence of such languages can be translated into a SN; however, penalty logic has properties that make it more attractive than the other languages. Algorithms are given for translating between any two of the languages above.

An inference engine is constructed that is capable of answering whether a query follows the knowledge or not. When a query is clamped, the global minima of the network correspond exactly to the correct answers.

Revision of the knowledge-base and adding new evidence are incremental tasks if we use penalty logic to describe the network. Adding (or deleting) a PLOFF is simply computing the energy function of the new PLOFF and then adding (deleting) the energy terms to the function that describes the existing knowledge. Thus, a local change to the PLOFF describing the network is translated to a local change in the network.

I have implemented several nonmonotonic *toy* problems (like Nixon, Penguins, etc.) on a Boltzmann machine simulator, and the network managed to always find a global minimum. I have not noticed any problems with local minima although they definitely exist. The good results for large-scale satisfiability problems are encouraging; however, we may discover that nonmonotonic problems are harder than strict satisfiability, and it may be that they are much more sensitive to the selection of penalties. For a discussion on future hopes for speed-up see Section 8.2.

[11] Any non-oscillating asymmetric network can also be represented as a logic sentence [Pinkas 91e].

7. ON THE FEASIBILITY OF FINDING A GLOBAL MINIMUM

7.1. Introduction

The networks[1] constructed so far implement knowledge-level theories [Newell 80] that were described in previous chapters. In order to perform a *sound* computation with respect to the desired knowledge-level theory, a network must find a global solution for the minimization of the energy associated with it. In this thesis and in many other works (e.g. [Hopfield, Tank 85], [Derthick 88], [Ballard et al. 86]), the problem at hand is formulated so that the best solutions[2] are the global minima of the energy function. Unfortunately, existing connectionist algorithms do not guarantee the finding of a global minimum, and even the search for a local minimum may take an exponential number of steps [Kasif et al. 89], [Papadimitriou et al. 90].

Even when we are interested just in approximation[3] of our knowledge-level theory, a desired algorithm is one that reduces the impact of shallow local minima and improves the chances of finding a global minimum.

Models such as Boltzmann machines and Harmony networks use simulated annealing to escape from local minima in the search for a global solution. These models asymptotically converge to a global minimum; i.e., if the annealing schedule is slow enough, a global minimum is guaranteed to be found. Nevertheless, such an annealing schedule is hard to find and therefore, practically, finding a global minimum for such networks is not guaranteed even in exponential time (note that the minimization problem is NP-hard).

The main questions that are asked in this chapter are: Is there a uniform distributed algorithm with one processor for each node that guarantees a global minimum? Are there network topologies that enable us to efficiently find a global minimum? Can we improve the standard algorithms so that special topologies are minimized efficiently while the performance for other topologies does not degrade?

The chapter looks at the *topology* of symmetric networks and studies the feasibility of finding a global minimum under various topologies and scheduling assumptions. It presents an improvement to the standard activation functions which guarantees that a global minimum is found in linear time for tree-like subnetworks. In addition, the new algorithm performs no worse than the standard algorithms for an arbitrary network topology. The chapter also studies self-stabilization (to be defined later) of the algorithm under various assumptions and investigates the feasibility of guaranteeing a

[1] This chapter is based on work done in collaboration with Rina Dechter [Pinkas, Decter 92].

[2] Sometimes the *only* solutions.

[3] Note that a recent result shows that no approximation *scheme* exists for maximal 2-satisfiability [Arora et al. 92] unless P=NP. Since quadratic energy minimization can be reduced to MAX-2-SAT (see Section 6.2) no approximation scheme is likely to be found to our problem.

global minimum in non-tree topologies. In particular, negative results show that we *cannot* do much better than the algorithm presented here; i.e., we will never find a uniform algorithm that guarantees a global minimum for topologies that are less restricted than trees.

The section is organized as follows. Section two presents the improved algorithm. Section three gives an example where the improved algorithm out-performs the standard algorithms. Section four considers self-stabilization, convergence and feasibility under various scheduling demons. Section five summarizes the results.

7.2. An Improved Algorithm for Energy Minimization

This section presents an algorithm that optimizes tree-like subnetworks in linear time. A tree-like subnetwork is characterized by an undirected graph without cycles; i.e., only one path exists between any two nodes. The terms "cycle-free" and "unrooted tree" are synonymous in this context. The algorithm is based on an adaptation of a dynamic programming algorithm presented in [Dechter et al. 90] and [Bertelé, Briosch 72]. The adaptation is connectionist in style; that is, the algorithm can be stated as a simple, uniform activation function [Rumelhart et al. 86], [Feldman 89]. It does *not* assume the desired, tree topology and performs no worse than the standard algorithms for all topologies. In fact, the algorithm may be integrated with many of the standard algorithms in such a way that if the network happens to have tree-like subnetworks, the new algorithm out-performs the standard algorithms in the following sense. In the case where there are trees whose roots are joined in a cyclic mesh (tree subnetworks), the new algorithm performs better than the standard algorithms by avoiding local minima along the trees and by optimizing the free energy of these trees in linear time.

7.2.1. Energy and Goodness

Suppose a quadratic energy function of the form:

$$E(X_1, ..., X_n) = -\sum_{i<j}^{n} w_{i,j} X_i X_j + \sum_{i}^{n} -\theta_i X_i.$$

Each of the variables X_i may have an activation value of zero or one, and the task is to find a zero/one assignment to the variables $X_1, ..., X_n$ that minimizes the energy function. To avoid confusion with signs, consider the equivalent problem of maximizing the goodness function:

$$G(X_1, ..., X_n) = -E(X_1, ..., X_n) = \sum_{i<j}^{n} w_{i,j} X_i X_j + \sum_{i} \theta_i X_i. \qquad (7.1)$$

Each of the nodes in the network is assigned a processing unit, and the network collectively searches for an assignment that maximizes the goodness. The algorithm that is

repeatedly executed in each unit/node is called the *protocol* or the *activation function.* A protocol is *uniform* if *all* the units execute it.

The *standard* activation functions that are common in the literature are presented in Section 2.5; however, I will review here two of the most popular ones; the discrete Hopfield model and the Boltzmann machine.

In the discrete Hopfield model, each unit computes its activation value using the formula

$$X_i = \begin{cases} 1 & \text{iff } \sum_j w_{i,j} X_j \geq -\theta_i, \\ 0 & \text{otherwise.} \end{cases}$$

For Boltzmann machines, the determination of the activation value is stochastic and the probability of setting the activation value of a unit to one is

$$P(X_i = 1) = 1/(1 + e^{-(\sum_j w_{i,j} X_j + \theta_i)/T}),$$

where T is the annealing temperature.

Both approaches can be integrated with our topology-based algorithm; in other words, nodes that cannot be identified as parts of a tree-like subnetwork use one of the standard algorithms.

7.2.2. Model of Communication

Our model of communication assumes shared memory, multi-reader/single-writer, scheduling under a central demon, and fair execution. In *shared memory, multi-reader/single-writer communication,* each unit has a shared register called the activation register. A unit may read the content of the registers of all its neighbors, but write only its own. *Central demon* scheduling means that the units are activated (asynchronously) one at a time in an arbitrary order.[4] An execution is said to be *fair* if every unit is activated infinitely often.

7.2.3. Key Idea

The algorithm identifies parts of the network that have no cycles (tree-like subnetworks) and optimizes the free energy on these subnetworks. Once a tree is identified, it is optimized using an adaptation of a constraint optimization algorithm for cycle-free graphs presented in [Dechter et al. 90]. The algorithm belongs to the family of nonserial dynamic programming methods [Bertelé, Briosch 72].

Let us assume first that the network is an unrooted tree. Any such network may be directed into a rooted tree. The algorithm is based on the observation that given an activation value (0/1) for a node in a tree, the optimal assignments for all its adjacent

[4]Standard algorithms assume the same condition in order to guarantee convergence to a *local* minimum (see [Hopfield 82]). This condition can be relaxed the restriction that only adjacent nodes may not be activated at the same time.

nodes are independent of each other. In particular, the optimal assignment to the node's descendants are independent of the assignments for its ancestors. Therefore, each node i in the tree may compute two values: G_i^0 and G_i^1. G_i^1 is the maximal goodness contribution of the subtree rooted at i, including the connection to i's parent whose activation is *one*. Similarly, G_i^0 is the maximal goodness of the subtree, including the connection to i's parent whose activation value is *zero*. The acyclic property will allow us to compute each node's G_i^1 and G_i^0 as a simple recursive function of its children's values, implemented as a propagation algorithm initiated by the leaves.

Knowing the activation value of its parent and the values G_j^0, G_j^1 of all its children, a node can compute the maximal goodness of its subtree. When the information reaches the root, it can assign a value (0/1) that maximizes the goodness of the whole network. The assignment information then propagates towards the leaves. Given the activation value of its parent, a node can compute the ideal activation value for itself to maximize the goodness of the subtree rooted by the node. At termination (at stable state), the tree is optimized.

The algorithm has three basic steps:
1) **Directing a tree.** Knowledge is propagated from leaves towards the center of the network, so that after a linear number of steps, every unit in the tree knows its parent and children.
2) **Propagation of goodness values.** The values G_i^1, G_i^0 are propagated from leaves to the root. At termination, every node knows the maximal goodness of its subtree, and the appropriate activation value it should assign, given that of its parent. In particular, the root can now decide its own activation value so as to maximize the goodness of the whole tree.
3) **Propagation of activation values.** Starting with the root, each node in turn determines its activation value. After *O(depth of tree)* steps, the units are in a stable state which globally maximizes the goodness.

Each unit's *activation register* consists of the fields X_i (the activation value); G_i^0, G_i^1 (the maximal goodness values); and $P_i^1, .., P_i^j$ (a bit for each of the j neighbors of i that indicates which is i's parent).

7.2.4. Directing a Tree

The goal of this algorithm is to inform every node of its role in the network and of its child-parent relationships. Nodes with a single neighbor identify themselves as leaves first and then identify their neighbor as a parent by pointing to it. A node whose neighbors all point towards it identifies itself as a root. A node whose neighbors all point towards it except for one neighbor, selects the one as a parent. Finally, a node that has at least two neighbors *not* pointing towards it, identifies itself as being outside a tree.

Each unit uses one bit per neighbor to keep the pointing information: $P_i^j = 1$ indicates that node i sees its jth neighbor as its parent. By looking at P_j^i, node i knows whether j is pointing to it.

Identifying tree-like subnetworks in a general network may be done by the following algorithm:

Tree Directing (for unit i):

1. Initialization: If first time, then for all neighbors j, $P_i^j = 0$. /* Start with clear pointers (this step is not needed in trees) */
2. If there is only a single neighbor (j) and $P_j^i = 0$, then $P_i^j = 1$. /* A leaf selects its neighbor as parent if that neighbor doesn't point to it */
3. Else, if one and only one neighbor (k) does not point to i ($P_k^i = 0$), then $P_i^k = 1$, and, for the rest of the neighbors, $P_i^j = 0$. /* k is the parent */
4. Else, for all neighbors j, $P_i^j = 0$. /* Node is either a root or outside the tree.

In Figure 7.1(a), we see a cycle-free network after the tree-directing phase. The numbers on the edges represent the values of the P_i^j bits. In Figure 7.1-(b), a tree-like subnetwork is identified inside a cyclic network. Note that node 5 is not a root, since not all its neighbors are pointing towards it. A detailed proof of convergence and self-stabilization (see Section 7.4.2) appears in Appendix A.10.

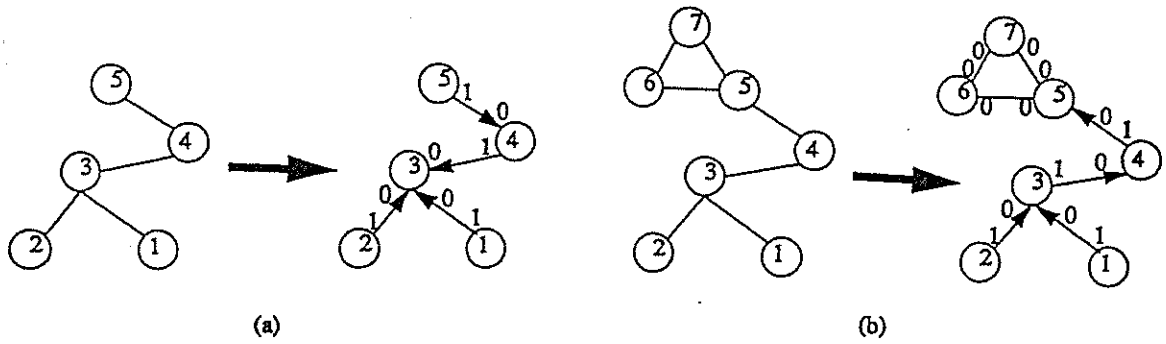

Figure 7.1: Directing a graph: (a) a tree, (b) a cyclic graph.

7.2.5. Propagation of Goodness Values

In this phase, every node i computes its goodness values G_i^1, G_i^0 by propagating these two values from the leaves to the root (see Figure 7.2).

Given a node X_i, its parent X_k, and its children $child(i)$ in the tree, it can be shown based on the energy function (7.1) that the goodness values obey the following

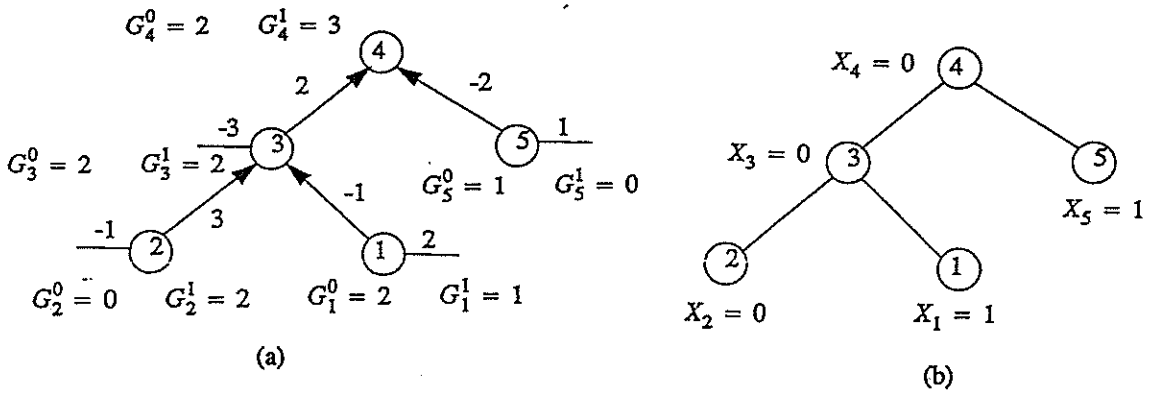

Figure 7.2: (a) Propagating goodness values. (b) Propagating activation values.

recurrence:

$$G_i^{x_k} = max\{\sum_{j \in child(i)} G_j^{x_i} + w_{i,k} X_i X_k + \theta_i X_i\}.$$

Consequently, a nonleaf node i computes its goodness values using the goodness values of its children as follows. If $X_k = 0$, then i must decide between setting $X_i = 0$, obtaining a goodness of $\sum_j G_j^0$, or setting $X_i = 1$, obtaining a goodness of $\sum_j G_j^1 + \theta_i$. This yields

$$G_i^0 = max\{\sum_{j \in child(i)} G_j^0, \sum_{j \in child(i)} G_j^1 + \theta_i\}.$$

Similarly, when $X_k = 1$, the choice between $X_i = 0$ and $X_i = 1$ yields

$$G_i^1 = max\{\sum_{j \in child(i)} G_j^0, \sum_{j \in child(i)} G_j^1 + w_{i,k} + \theta_i\}.$$

The initial goodness values for leaf nodes can be obtained from the above with no children. Thus,

$$G_i^0 = max\{0, \theta_i\},$$
$$G_i^1 = \{0, w_i k + \theta_i\}.$$

For example, if unit 3 in Figure 7.2 is zero, then the maximal goodness contributed by node 1 is $G_1^0 = max_{X_1 \in \{0,1\}}\{2X_1\} = 2$, and it is obtained at
$X_1 = 1$. Unit 2 (when $X_3 = 0$) contributes $G_2^0 = max_{X_2 \in \{0,1\}}\{-X_2\} = 0$, obtained at $X_2 = 0$, while $G_2^1 = max_{X_2 \in \{0,1\}}\{3X_2 - X_2\} = 2$ is obtained at $X_2 = 1$. As for nonleaf nodes, if $X_4 = 0$, then when $X_3 = 0$, the goodness contribution will be $\sum_k G_k^0 = 2+0 = 2$, while if $X_3 = 1$, the contribution will be $-3+\sum_k G_k^1 = -3+1+2 = 0$. The maximal contribution $G_3^0 = 2$ is achieved at $X_3 = 0$.

7.2.6. Propagation of Activation Values

Once a node is assigned an activation value, all its children can activate themselves so as to maximize the goodness of the subtrees they control. When such a value is chosen for a node, its children can evaluate *their* activation values, and the process continues until the whole tree is assigned.

There are two kinds of nodes that may start the process: a root which will choose an activation value to optimize the entire tree, and a non-tree node which uses a standard activation function.

When a root X_i is identified, it chooses the value zero if the maximal goodness is $\sum_j G_j^0$, while it chooses one if the maximal goodness is $\sum_j G_j^1 + \theta_i$. In summary, the root chooses its value according to

$$X_i = \begin{cases} 1 & \text{iff } \sum_j G_j^1 + \theta_i \geq \sum_j G_j^0, \\ 0 & \text{otherwise.} \end{cases}$$

In Figure 7.2, for example, $G_5^1 + G_3^1 + 0 = 2 < G_5^0 + G_3^0 = 3$ and therefore $X_4 = 0$.

An internal node whose parent is k chooses an activation value that maximizes $\sum_j G_j^{x_i} + w_{i,k} X_i X_k + \theta_i X_i$. The choice, therefore, is between $\sum_j G_j^0$ (when $X_i = 0$) and $\sum_j G_j^1 + w_{i,k} X_k + \theta_i$ (when $X_i = 1$), yielding:

$$X_i = \begin{cases} 1 & \text{iff } \sum_j G_j^1 + w_{i,k} X_k + \theta_i \geq \sum_j G_j^0 \\ 0 & \text{otherwise.} \end{cases}$$

As a special case, a leaf i chooses $X_i = 1$ iff $w_{i,k} X_k \geq -\theta_i$, which is exactly the discrete Hopfield activation function for a node with a single neighbor. For example, in Figure 7.2, $X_5 = 1$ since $w_{4,5} X_4 = 0 > -\theta_5 = -1$, and $X_3 = 0$ since $G_1^1 + G_2^1 + 2X_4 + \theta_3 = 1 + 2 + 0 - 3 = 0 < G_2^0 + G_1^0 = 2$. Figure 7.2-(b) shows the activation values obtained by propagating them from the root to the leaves.

7.2.7. A Complete Activation Function

Interleaving the three algorithms described earlier achieves the goal of identifying tree-like subnetworks and maximizing their goodness. In this subsection the complete algorithm is presented, combining the three phases while simplifying the computation. The algorithm is integrated with the discrete Hopfield activation function; however it can also be integrated with the other activation functions mentioned in Section 2.5.[5]

[5]Note how similar the new activation function is to the original Hopfield function.

Let i be the executing unit, j a non-parent neighbor of i, and k the parent of i:

Optimizing on Tree-like Subnetworks (unit i):

1. Initialization: If first time, then $(\forall j)\ P_i^j = 0$. /*Clear pointers (needed only for cyclic nets)*/

2. Tree directing: If there exists a single neighbor k, such that $P_k^i = 0$,
   then $P_i^k = 1$, and for all other neighbors j, $P_i^j = 0$;
   else, for all neighbors, $P_i^j = 0$.

3. Computing goodness values:
   $G_i^0 = max\{\sum_{j \in child(i)} G_j^0 P_j^i, \sum_{j \in child(i)} G_j^1 P_j^i + \theta_i\}$.
   $G_i^1 = max\{\sum_{j \in child(i)} G_j^0 P_j^i, \sum_{j \in child(i)} (G_j^1 P_j^i + w_{i,j} P_i^j) + \theta_i\}$.

4. Assigning activation values:
   If at least two neighbors are not pointing to i, then /*use standard activation function (Hopfield) */
   $$X_i = \begin{cases} 1 & \text{if } \sum_j w_{i,j} X_j \geq -\theta_i, \\ 0 & \text{otherwise;} \end{cases}$$
   else, /* Node in a tree (including root and leaves) */
   $$X_i = \begin{cases} 1 & \text{if } \sum_j ((G_j^1 - G_j^0) P_j^i + w_{i,j} X_j P_i^j) \geq -\theta_i, \\ 0 & \text{otherwise.} \end{cases}$$

THEOREM **7.2.1** THE ALGORITHM ALWAYS CONVERGES TO A SOLUTION THAT MINIMIZES THE ENERGY ALONG TREE-LIKE SUBNETWORKS IN LINEAR TIME.

Proof:
See Appendix A.10. □

7.3. An Example

The example in Figure 7.3 demonstrates a case where a local minimum of the standard algorithms is avoided. Standard algorithms may enter such a local minimum and stay in a stable state that is clearly wrong.

The example is a variation on a Harmony network [Smolensky 86] and an example from [McClelland et al. 86]. The task of the network is to identify words from low-level line segments. Certain patterns of line segments excite units that represent characters, and certain patterns of characters excite units that represent words. The line strokes used to draw the characters are the input units: L1,..., L5. The units "N," "S," "A," and "T" represent characters. The units "able," "nose," "time," and "cart" represent words, and Hn, Hs, Ha, Ht, H1,...,H4 are hidden units required by the Harmony model.

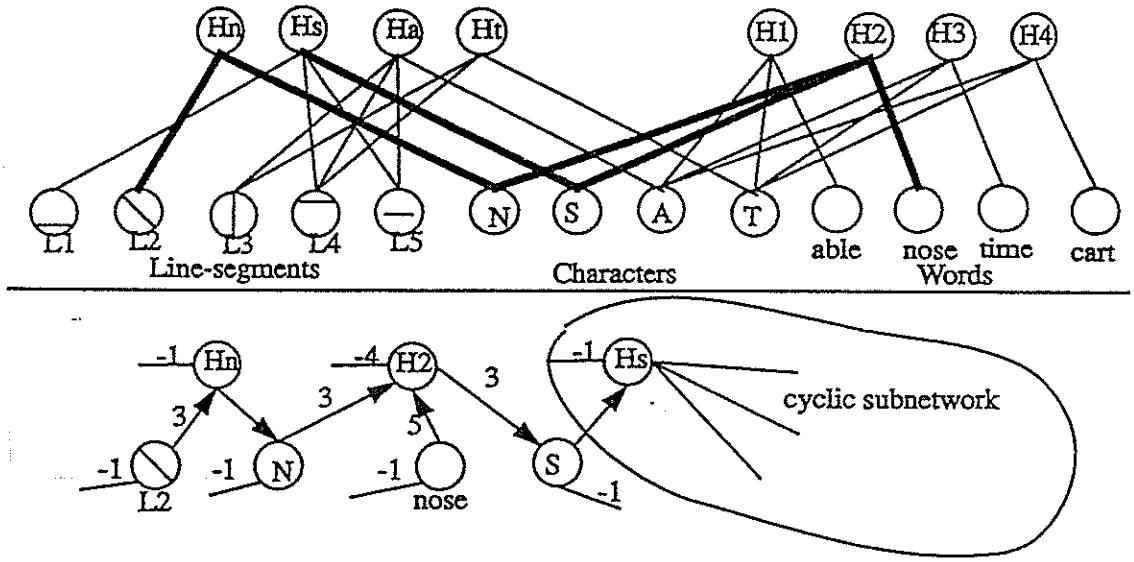

Figure 7.3: A Harmony network for recognizing words. Local minima along the subtrees are avoided.

For example, given the line segments of the character S, unit L4 is activated (input), and this causes units Hs and "S" to be activated. Since "NOSE" is the only word that contains the character "S," both H2 and the unit "nose" are also activated and the word "NOSE" is identified.

The network has feedback cycles (symmetric weights) so that ambiguity among characters or line-segments may be resolved as a result of identifying a word. For example, assume that the line segments required to recognize the word "NOSE" appear, but the character "N" in the input is blurred, and therefore the setting of unit L2 is ambiguous. Given the rest of the line segments (e.g., those of the character "S"), the network identifies the word "NOSE" and activates units "nose" and H2. This causes unit "N" to be activated along with all of its line segments. Thus the ambiguity of L2 is resolved.

The network is indeed designed to have a global minimum when L2, Hn, "N," H2, and "nose" are all activated; however, standard connectionist algorithms may fall into a local minimum when all these units are zero, generating goodness of $5 - 4 = 1$. The correct global minimum is found by our tree-optimization protocol with goodness: 3-1+3-1+3-1+5-1-4+3-1+5=13. The thick arcs in the upper network of Figure 7.3 mark the arcs of a tree-like subnetwork. This tree-like subnetwork is drawn with pointers and weights in the lower part of the figure. Node "S" is not part of the tree, and its activation value is set to one because the line-segments of "S" are activated. Once "S" is set, the units along the tree are optimized (by setting them all to one) and the local minimum is avoided.

7.4. Feasibility, Convergence and Self-Stabilization

So far, a way has been shown to enhance the performance of connectionist energy minimization networks without losing much of the simplicity of the standard approaches. The simple algorithm presented is limited in two ways, however. First, the central demon[6] is not a realistic restriction. We would like the network to work correctly under a *distributed demon*, where any subset of units may be scheduled for execution at the same time. Second, we would like the algorithm to be *self-stabilizing*. It should converge to a legal, stable state given enough time, even after noisy fluctuations that cause the units to execute arbitrary program states and the registers to have arbitrary content.

7.4.1. Negative Results for Uniform Protocols

Following [Dijkstra 74] and [Collin et al. 90], we show the following theorem:

THEOREM **7.4.1** NO DETERMINISTIC UNIFORM ALGORITHM EXISTS THAT GUARANTEES A GLOBAL MINIMUM UNDER A DISTRIBUTED DEMON, EVEN FOR SIMPLE CHAIN-LIKE TREES.

Proof:
Consider the network of Figure 7.4(a). There are two global minima possible : (11...1101...11) and (11...1011...11). If the network is initialized such that all units have the same register values, and all units start with the same program state, then there exists a fair execution under a distributed demon such that in every step, all units are activated. The units left of the center $(1, 2, 3, ...i)$ "see" the same input as those units right of the center $(2i, 2i-1, 2i-2, ..., i+1)$ respectively. Because of the uniformity and the determinism, the units in each pair $(i, i+1), (i-1, i+2), ..., (1, 2i)$ must transfer to the same program state and produce the same output on the activation register.[7] Thus, after every step of that execution, units i and $i+1$ will always have the same activation value and a global minimum (where the two units have different values) will never be obtained. □

This negative result should not discourage us, since it relies on an obscure infinite sequence of executions which is unlikely to occur under a truly random demon. The algorithm presented optimizes the energy of tree-like subnetworks under a distributed demon in each of the following cases: 1) If step 2 of the protocol in Section 7.2.7 is atomic; 2) if for every node i and every neighbor j, node i is executed without j infinitely often; 3) if one node is unique and acts as a root, that is, does not execute step 2 (an almost uniform protocol); and 4) if the network is cyclic (one node will be acting as a root).

Another negative result similar to [Collin et al. 90] is given in the following theorem.

[6] The same results are obtained if the protocol is atomic.

[7] We assume that when execution begins, all units are initialized with the same values and that all of them start in the same program state.

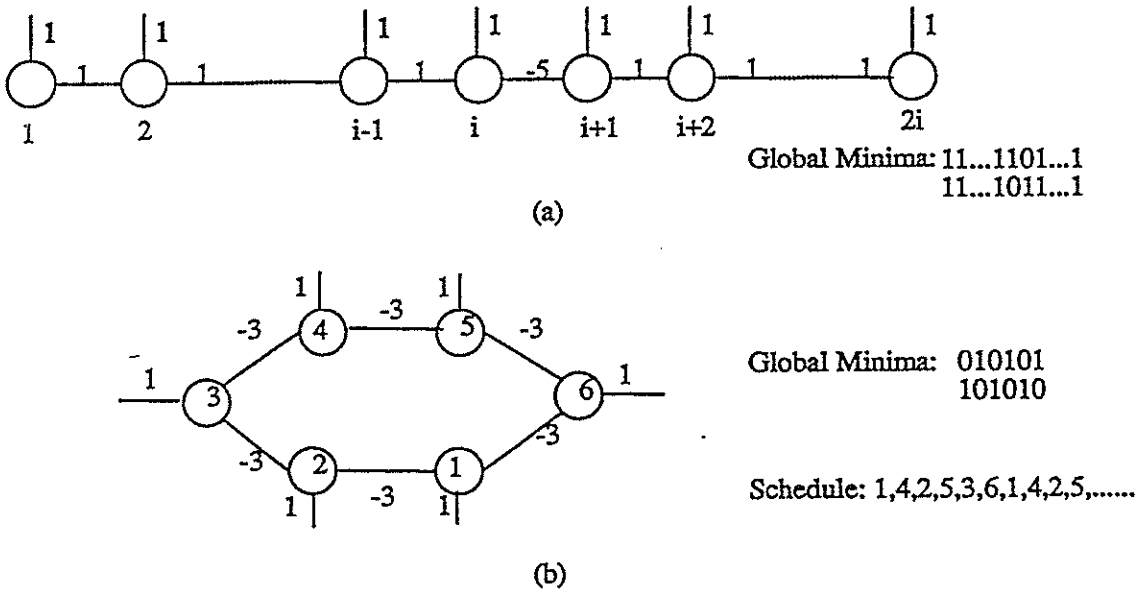

Figure 7.4: (a) No uniform algorithm exists to optimize chains under distributed demon; (b) No uniform algorithm exists that is guaranteed to optimize rings even under a central demon.

THEOREM **7.4.2** IF THE NETWORK IS CYCLIC, NO DETERMINISTIC UNIFORM ALGORITHM EXISTS THAT GUARANTEES A GLOBAL MINIMUM, EVEN UNDER A CENTRAL DEMON.

This may be proved even for cyclic networks as simple as rings.
Proof:
In Figure 7.4-(b), we see a ring-like network whose global minima are (010101) and (101010). Consider a fair execution under a central demon that activates the units 1,4,2,5,3,6 in order and repeats this order indefinitely. Starting with the same program state and same inputs, the two units in every pair of (1,4), (2,5), (3,6) "see" the same input, therefore they have the same output, and transfer to the same program state. As a result, these units never output different values, and a global minimum is not obtained. □

7.4.2. Self-Stabilization

A protocol is *self-stabilizing* if in any fair execution, starting from any input configuration and any program state (of the units), the system reaches a valid stable configuration. The motivation here is that even if hardware fluctuations occur and change the content of the registers (as well as the state of the program), the algorithm is guaranteed to recover and to find a correct stable state.

The algorithm in Section 7.2.7 is self-stabilizing for cycle free networks (trees), and it remains self-stabilizing under a distributed demon with the same weak restrictions as in the previous section; i.e., executes without a neighbor infinitely often or is almost uniform (has a marked root).

DEFINITION **7.4.1** A scheduler has the *fair exclusion* property if for every two neighbors, one is executed without the other infinitely often.

THEOREM **7.4.3** THE ALGORITHM IS SELF-STABILIZING FOR TREES EVEN UNDER A DISTRIBUTED DEMON WITH FAIR EXCLUSION. THE ALMOST UNIFORM VERSION IS SELF-STABILIZING FOR GENERAL TOPOLOGIES EVEN UNDER A (PURE) DISTRIBUTED DEMON.

Proof:
See Appendix A.10. □

The algorithm is not self-stabilizing for general cyclic topologies. For example, consider the configuration of the pointers in the ring of Figure 7.5. It is in a stable state, although clearly not a valid tree.[8] To solve the problem of self-stabilization in

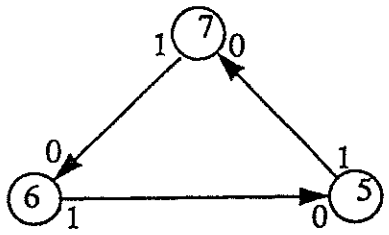

Figure 7.5: The simple protocol is not self-stabilizing in cyclic networks.

cyclic networks, we need to make our algorithm more complex. For example, we may use a variation of the self-stabilizing tree-directing protocol of [Collin et al. 90]. This algorithm remains self-stabilizing even in cyclic networks. Thus, self-stabilization may be obtained in the general case, but at the expense of more complexity and more space.

7.5. Summary and Discussion

The chapter shows a uniform, self-stabilizing, connectionist activation function that is guaranteed to find a global minimum of acyclic symmetric networks in linear time. The algorithm optimizes tree-like subnetworks within general (cyclic) networks; however, the simple version of the algorithm loses its self-stabilization property in cyclic networks. Nevertheless, we can extend the algorithm to be self-stabilizing for all network topologies at the expense of more space requirements.

[8] Such a configuration will never occur if all units start at the *starting point*; i.e., clearing the bits of P_i. It may only happen due to some noise or hardware fluctuation.

Two negative results were proved. First, under a pure distributed scheduling demon, no *uniform* algorithm exists to optimize even simple chains. Second, no uniform algorithm exists to optimize simple cyclic networks (rings) even under a central demon.

The conclusion is that there is not much hope to do better than what we did with the algorithm presented here, assuming we insist on global minima and uniform algorithms. If we insist on uniformity, no algorithm can guarantee global solutions for networks that are less restricted than trees. I conjecture, however, that these negative results are not of significant practical importance, since in truly random scheduling demons, the probability of having such pathological executions approaches zero. It is more likely that real schedulers will have the fair exclusion property which assumes that for every two neighbors, one is executed without the other infinitely often. The algorithm remains correct under a distributed scheduling demon if some weak assumptions are made about the demon; i.e., fair exclusion, almost-uniformity or atomicity.