Some Results on the Computational Complexity of Symmetric Connectionist Networks[1]

Saibal Banerjee
Arthur Delcher[2]
Simon Kasif
Gregory Sullivan

JHU/CS-89/10

Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218

November, 1989
Report JHU/CS-89/10

Abstract

Connectionist models currently are being investigated actively by many researchers in Artificial Intelligence, Information Theory and Neurology. These networks have been shown to be applicable to a wide range of domains such as content addressable memories, semantic nets, computer vision, natural language parsing, speech and approximation schemes for hard optimization problems.

In this paper we address the computational complexity of discrete Hopfield nets (connectionist networks with symmetric connections). These networks have been advocated as effective means for solving combinatorial optimization problems. The main results reported in this paper are as follows:

1. We relate discrete Hopfield net execution to a well-known local search technique that has been used effectively in computer science to solve NP-hard problems.
2. We prove an intractability result for networks with symmetric connections. Previous similar results have been known only in the context of much stronger versions of connectionist networks (general directed networks).
3. We propose several heuristics to speed-up the convergence of the network and improve the quality of the local minima found by the network.
4. We prove that the standard algorithm that achieves local stability in such networks has exponential worst-case complexity.
5. We propose a simple scaling method to improve the performance of the network
6. We define a non-deterministic version of the net for which we prove a fast convergence result.

[1]Revised and updated version of JHU/EECS-87/18
[2]supported by AFOSR grant AFOSR-89-1151 and NSF grant IRI-88-09324

1 Introduction

Though research in Artificial Intelligence (AI) has led to several important developments, many fundamental problems in AI remain open. Theoretical studies indicate that many of the basic problems in AI are inherently intractable (NP-complete) if not worse (undecidable). At the same time, humans seem to solve these problems quite accurately in the majority of cases in just a few milliseconds (vision, speech, natural language, planning, etc.). In recent years a growing new class of algorithms has been proposed for AI applications. These algorithms, sometimes called connectionist algorithms, are implemented on a massively parallel network that performs constrained, optimization-like procedures.

The algorithms in this class have several interesting characteristics:

1. The problem is formulated as an "energy minimization" process due to the analogy to well-known physical processes in statistical mechanics [Fel85, Hop82].
2. The algorithms can be implemented on large massively parallel networks of interconnected processors operating asynchronously [FHS]. [FHS]??
3. The algorithms use a few powerful constraint optimization techniques, without resorting to careful designs that vary dramatically depending on the problem (as traditionally done in theoretical computer science).

The above properties make this approach particularly interesting from the perspective of computational complexity. The main advantage of the connectionist network formulation is its general purpose applicability to a broad range of problem domains [BH86, HT85, RM86]. Thus, the method can be applied to new and unknown applications for which no previous algorithms are known. In this sense, the approach is analogous to such powerful constraint satisfaction procedures as linear programming, resolution-based theorem proving, and relaxation. Unlike these three methods, however, the recently developed learning algorithms allow the network to adapt to changing environments and improve its performance with time [HSA84, RHW85].

One key problem that remains largely unsolved is the computational complexity of connectionist network algorithms. This question has two main aspects.

1. Determining the class of computations performed particularly well by these networks and the class of computations that are inherently unsuitable for connectionist implementation. Intuitively, given a problem P of input size N, we would like to estimate whether a network of size $S(N)$ can solve the problem in time $T(N)$.
2. Determining the computational complexity of the optimization process performed by the network. Specifically, given a description of the network of size N, can we find subexponential algorithms to find stable (and maximally stable) states in the network that correspond to local and global energy minima (see Section 2 for details).

Note that the two questions are related but will not necessarily yield the same answers. A good analogy is linear programming that has polynomial-time algorithms, whereas the

Simplex algorithm is exponential in the worst case but has excellent expected behavior. Similarly, it is plausible that we can develop a fast algorithm for problem 2 whereas the network algorithm will choose an exponentially long path to convergence.

In this paper we address question 2 in the context of Hopfield nets. These networks have been advocated as effective means for solving combinatorial optimization problems [HT85]. The main results reported in this paper are as follows:

1. We relate discrete Hopfield net execution to a well-known local search technique that has been used effectively in computer science to solve NP-hard problems.
2. We prove an intractability result for networks with symmetric connections. Previous similar results have been known only in the context of much stronger versions of connectionist networks (general directed networks).
3. We propose several heuristics to speed-up the convergence of the network and improve the quality of the local minima found by the network.
4. We prove that the standard algorithm that achieves local stability in such networks has exponential worst-case complexity.
5. We propose a simple scaling method to improve the performance of the network
6. We define a non-deterministic version of the net for which we prove a fast convergence result.

For completeness, we include several basic definitions and well-known observations about the complexity of the networks. In these cases we merely sketch the proofs in order to convey the intuition underlying them.

2 Connectionist Networks

Connectionist models currently are being investigated actively by many researchers in Artificial Intelligence, Information Theory and Neurology. Informally, a *connectionist network* is a large collection of computational units (nodes), each with a finite number of states and elementary computational capabilities (see Rumelhart & McClelland [RM86] for an extended list of references and a survey on the subject). Information exchange between the various nodes in the network is supported by interconnections which are able to pass information in either direction. By such interconnections, every node can report its current state to every other node connected to it. The change of state of a computational unit is referred to as *firing*. When any node fires, it computes a weighted sum over all it connections to other nodes, the weights being positive or negative real numbers, and changes its state if the sum passes some designated threshold value. A cost or energy function, representing the summation of the weighted sums of all the nodes, can be used to describe the behavior of such a system of interacting nodes. According to the proposed model, a computation in a neural network proceeds by each node asynchronously updating its state. The computation stops when the energy of the system achieves a local minimum, at which point any change in state of a single node would increase (or leave unchanged) the total energy of the system.

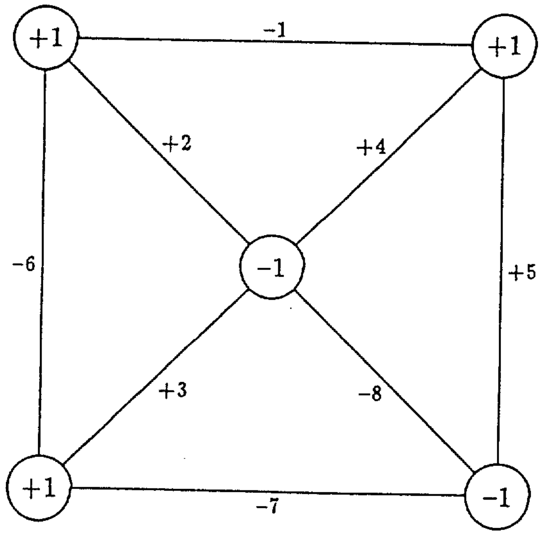


Figure 1: An example of the model

A particular connectionist architecture, known as the Hopfield model [Hop82], is closely related to various problems in several disciplines. In optimization theory, many problems including the traveling salesman problem can be modeled along these lines. In artificial intelligence, knowledge representation schemes and learning systems are based on this model. In physics, such models closely resemble Ising spin models, extensively used in the study of crystal structure. In this paper we are primarily interested in the computational properties of connectionist networks and their applicability to a range of computational problems.

3 The Discrete Hopfield Network (DHN)

A neural network with n nodes is modeled by a simple undirected graph $G = (V, E)$ on n vertices. (Since network nodes are formally represented by graph vertices, we shall use the words "node" and "vertex" interchangeably.) Each edge $e \in E$ has a real-valued weight w_e associated with it. The edges represent the interconnections between the various nodes. The weight w_{ij} of edge $(i,j) \in E$ (which could be negative) represents the strength of the synaptic connection between the two nodes i and j. Note that if nodes i and j are not connected in G then the weight between them can be assumed to be identically zero. Also note that since G is a simple undirected graph, $w_{ij} = w_{ji}$ and $w_{ii} = 0$, for all $i, j \in V$. Node i can be in one of two states: $x_i = +1 =$ "on" or $x_i = -1 =$ "off". Hopfield and Tank describe an analog version of the above where the state-transition function of each neuron (node) is a continuous sigmoid. An example of the model is shown in Figure 1.

3.1 Statement of the Problem

Let $W = \{w_{ij}\}$ represent the $n \times n$ weighted adjacency matrix of the graph G. Let $\vec{x}$ represent the $n \times 1$ state vector, whose i^{th} entry is x_i, the state of vertex i.

Let the quadratic cost function J be defined by:

$$J(\vec{x}) = - \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}\, x_i\, x_j = \vec{x}^T W \vec{x}$$

where $\vec{x}^T$ is the transpose of $\vec{x}$.

Now consider the following two problems:

Local Minimum

Input: W, the weight matrix of G

Output: $\vec{x}^* \epsilon \{+1, -1\}^n$ such that

$$J(\vec{x}^*) \leq J(\vec{x}), \quad \forall\, \vec{x} \,\epsilon\, N(\vec{x}^*)$$

where $N(\vec{x})$ denotes the set of n vectors in $\{+1, -1\}^n$ which differ from $\vec{x}^*$ in exactly one component, *i.e.*, $N(\vec{x}^*)$ is the set of n vectors whose Hamming distance from $\vec{x}^*$ is exactly one.

An alternative formulation is:

Output: $\vec{x}^* \epsilon \{+1, -1\}^n$ such that

$$- x_i^* \sum_{j=1}^{n} w_{ij}\, x_j^* \leq 0, \quad \text{for } 1 \leq i \leq n$$

where x_k^* is the k^{th} component of the vector $\vec{x}^*$.

Global Minimum

Input: W, the weight matrix of G

Output: $\vec{x}^* \epsilon \{+1, -1\}^n$ such that

$$J(\vec{x}^*) \leq J(\vec{x}), \quad \forall\, \vec{x} \,\epsilon\, \{+1, -1\}^n$$

Clearly a solution exists for both of these problems. We now address some applications of local/global minima in connectionist networks.

3.2 Local Minimum

It has been mentioned in the introduction that in the Hopfield model a local minimum configuration corresponds to an information storage point of the neural network. These storage points are stable, because if for some reason (such as external disturbances) only one node changes its state, the system in most cases will go back to its original state.

Furthermore, this tendency of the Hopfield model can be used to explain the phenomenon of content-addressable memory found in neural organizations.

There are many computational problems, particularly in the area of simple undirected graphs, that can be cast into the framework of the Local Minimum problem. This is achieved by constructing a suitable weight matrix W. One such problem is finding a maximal independent set (MIS) in a graph. An MIS of an undirected graph is a maximal collection of vertices S such that no two vertices in S are adjacent in the graph. Luby shows that the problem of finding an MIS in a graph is a special case of finding a local minimum in the Hopfield model [Lub85].

3.3 Global Minimum

In most optimization problems (see, for example, [Lue84]) finding a global optimum is of primary importance. In many such problems, however, the best we can effectively achieve (without exhaustive search) is a local optimum. In several important cases (*e.g.*, linear programming or quadratic programming with a positive-definite matrix), a local optimum is also a global one. This fact may simplify the search strategy considerably. However, this is not true in general.

In the Hopfield model, only a local minimum is computed. The most stable state of the system, however, is one corresponding to a global minimum. The Global Minimum problem is therefore very important in this context. Moreover, even in problems where only a local optimum is sought, various researchers have commented on the *quality* of the local optimum found. (See [LK73], for example.) The quality of a local optimum can be judged by its *closeness* to a global optimum. Hence, the importance of studying the Global Minimum problem.

The global problem is also important from a practical standpoint. Although the problem may be intractable, a good approximation algorithm/heuristic for it may lead us close to a local minimum. Furthermore, in the context of constraint satisfaction networks, an approximate solution to a global minimum may be considered to be a better state than a "high and shallow" local minimum.

4 The Sequential (Asynchronous) Algorithm

In this section, we review the standard algorithm for solving the Local Minimum problem.

The dynamic behavior of a connectionist network consists of two parts:

- The scheduling regime (updating rate).
- The transition rule.

The scheduling regime determines the order in which nodes are considered for possible state changes. For example in the Hopfield model, the nodes update asynchronously and at random. In [HSA84] the nodes are updated according to a statistical Boltzmann distribution.

The transition rule describes the conditions for a node to change its current state.

The dynamic behavior of a connectionist network can be explained through the concept of *firing*, *i.e.*, the change of state of a node when proper conditions are met. We define the behavior of the system as follows:

Consider a single node i being fired. Let x_j^- represent the state of node j before node i is fired, and x_j^+ the state after i fires. Then if $j \neq i$, $x_j^- = x_j^+$ and

$$x_i^+ = \begin{cases} +1 & \text{if } \sum_{j=1}^n w_{ij}\, x_j^- > 0 \\ -1 & \text{if } \sum_{j=1}^n w_{ij}\, x_j^- < 0 \\ x_i^- & \text{if } \sum_{j=1}^n w_{ij}\, x_j^- = 0 \end{cases}$$

We call this rule for a node to change state the *threshold law.* Stated another way, node i is stable (does not change state) if

$$x_i \sum_{j=1}^{n} w_{ij}\, x_j \geq 0$$

i.e., the local energy state is zero or negative.

We do not consider node threshold values other than zero, since these can be simulated by adding additional nodes and clamping their states.

This threshold law provides the basis of the following sequential algorithm for finding a local optimum.

Algorithm 1—The Sequential Algorithm

Input:	W, the weight matrix of G
Output:	$\vec{x}^* \in \{+1, -1\}^n$, a solution to the Local Minimum problem
Method:	Iterate until the quadratic cost function J does not change. Each iteration is sequential scan of all vertices to see if any one should change state in accordance with the threshold law. A schematic description of the algorithm is given in Figure 2.

The following proposition is well-known.

Proposition 1 *Algorithm 1 terminates in a finite number of steps and is correct.*

Proof: (Sketch) At each iteration either the value in J_{new} decreases by at least

$$\min_{(i,j) \in E} \{|w_{ij}|\}$$

or else the algorithm terminates. Since the J values are bounded below by

$$-\sum_{i=1}^{n} \sum_{j=1}^{n} |w_{ij}|$$

the algorithm eventually must terminate. Correctness follows from our definition of a local minimum.

◁

It is easy to show that if the nodes are allowed to fire in parallel then convergence is not guaranteed. Such a scheme can lead to oscillations in the system.

```
algorithm Sequential
  begin
    Let $\vec{x}$ be any vector in $\{+1, -1\}^n$
    $J_{new} \leftarrow \vec{x}^T W \vec{x}$
    repeat
      $J_{old} \leftarrow J_{new}$
      for $i \leftarrow 1$ to $n$ do
        if $x_i \sum_{j=1}^{n} w_{ij} x_j < 0$
          $x_i \leftarrow x_i$
        $J_{new} \leftarrow \vec{x}^T W \vec{x}$
      until $J_{old} = J_{new}$
    end
  $\vec{x}$ is the local minimum
```

Figure 2: Algorithm 1—The Sequential Algorithm

5 Complexity Results

In the previous sections, the edge weights in the Hopfield model were real numbers. However, in order to describe the complexity results in this section we make the assumption that the weights are integers (positive or negative).

When discussing complexity, a key issue is stating precisely what is the input. In this case it would be any standard binary encoding of the network (*e.g.*, adjacency matrix, list of adjacency lists). Given this standard assumption it is plausible that given a graph described by n bits, the largest weight is of size $2^{O(n)}$ ($O(n)$ bits). Thus, it is theoretically possible that Algorithm 1 will require exponential time to converge. In fact, in the case of directed networks we can construct a polynomial size circuit that simulates a counter and achieves this bound. This occurs because the actual weights of the edges can be of $2^{O(n)}$.

Let w_{max} be defined by

$$w_{max} = \max_{(i,j) \in E} \{|w_{ij}|\}$$

Thus if any edge weight in the graph G requires $O(n)$ bits for its binary representation, w_{max} will be exponential in n. However, we can show the following

Proposition 2 *Let G be an n node network. If w_{max} is polynomially bounded in n, then Algorithm 1 takes a polynomial number of steps to converge to a local minimum.*

Proof: (Sketch) Since w_{max} is polynomial in n, so is the minimum value of J. Each iteration of the algorithm reduces the value of J by at least one (since all values are integers), so that after polynomially many steps, the minimum J value is achieved. $\lhd$

Since it is possible for w_{max} to assume values which are exponential in n, we would like to find out whether there exists an $n \times n$ weight matrix W and an initial starting vector $\vec{x}$ for which Algorithm 1 requires exponential time to converge to a local minimum. This is the first major open problem that we would like to address, since the majority of current algorithms for finding local optima are variations of Algorithm 1. This is analogous to the Simplex method which behaves very well on average, but has exponential worst-case time complexity.

In the case of the Global Minimum problem, the quest for a fast algorithm becomes provably harder. Consider the decision problem corresponding to the Global Minimum.

Global Minimum Decision Problem (GMD)

Input: W, the weight matrix of G and an integer $K < 0$
Question: Does there exist a vector $\vec{x} \in \{+1, -1\}^n$ such that

$$J(\vec{x}) = \vec{x}^{\mathrm{T}}\, W\, \vec{x} \leq K$$

Proposition 3 *GMD is NP-complete.*

Proof: (Sketch) Transformation from MAX-CUT [Kar72]. $\triangleleft$

6 Graph-Theoretic Reformulation

In this section we make our first main observation by relating Algorithm 1 to a well-known local search technique used in combinatorial optimization, namely, the Lin-Kernighan heuristic for finding cuts in graphs [PS82]. Consider any vector $\vec{x}$ in $\{+1, -1\}^n$. Note that since

$$J(\vec{x}) = \vec{x}^{\mathrm{T}}\, W\, \vec{x} = (-\vec{x})^{\mathrm{T}}\, W\, (-\vec{x}) = J(-\vec{x})$$

both the Local Minimum and Global Minimum problems remain invariant if the signs of all the states are reversed, *i.e.*, if all states with label $+1$ were changed to a label of -1 and vice versa. Consequently, the problem can be viewed as a problem of partitioning the vertex set V of the graph G into two sets V_1 and V_2 ($V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$), with vertices having the same label being in the same subset. Without loss of generality, we shall henceforth follow the convention that all vertices with state $+1$ are in V_1 and all vertices with state -1 are in V_2. Note that there is a bijection between every vector $\vec{x} \in \{+1, -1\}^n$ and every bipartition (V_1, V_2) of the vertex set V of graph G.

Consider any vector $\vec{x} \in \{+1, -1\}^n$. The quadratic cost function $J(\vec{x})$ associated with this vector can be rewritten in terms of the bipartition (V_1, V_2) corresponding to $\vec{x}$ as:

$$J(\vec{x}) = -2 \sum_{e \in E} w_e \; - \; 4 \sum_{\substack{u \in V_1 \\ v \in V_2}} w_{uv}$$

Additionally, observe that by multiplying all the weights by -1 we can convert a minimization problem into an equivalent maximization problem. In order to correspond more closely to graph theoretic methodology we prefer to assume that initially we multiplied all the weights by -1. Any minimization of the cost function $-J$ thus entails a corresponding maximization of the interpartition sum, *i.e.*, the sum of the weights of all the edges "across" the bipartition. Formally, if $H = (V, E_H)$ is the spanning bipartite

subgraph of G corresponding to the bipartition (V_1, V_2), where
$E_H = \{(u,v) \in E \mid u \in V_1 \text{ and } v \in V_2\}$, then the interpartition sum is given by

$$\sum_{e \in E_H} w_e$$

For simplicity, we shall refer to the subgraph H as a bipartition of G. We now can reformulate our Local and Global Minimum problems as follows.

6.1 Local Minimum

Informally, this problem seeks a bipartition (V_1^*, V_2^*) of the vertex set V such that, if we move any one vertex v from one subset to the other i.e. either from V_1^* to V_2^* or from V_2^* to V_1^*, the interpartition sum decreases. It can easily be shown from the following identity

$$x_i \sum_{j=1}^{n} w_{ij} x_j = \sum_{x_i = x_j} w_{ij} - \sum_{x_i \neq x_j} w_{ij}$$

and our convention about multiplying the weights by -1 that the definition of Local Minimum given in Section 3 is equivalent to the following reformulation:

Local Minimum

Input: Edge-weighted graph G

Output: A bipartition H of G such that

$$\forall v \in V, \qquad \sum_{(u,v) \in H} w_{uv} \geq \sum_{(u,v) \in G \setminus H} w_{uv}$$

or equivalently

$$\forall v \in V, \qquad \sum_{(u,v) \in H} w_{uv} \geq \frac{1}{2} \sum_{(u,v) \in G} w_{uv}$$

namely, find a bipartition of vertex set of V into two subsets such that, for each vertex, the weighted sum of its edges "across" the bipartition is at least as great as the weighted sum of its edges "inside" the bipartition.

In most applications [HT85], the networks are presented with an input, namely a designated set of nodes is clamped to a given set of values.

The next proposition states that this version is intractable in the worst case.

Proposition 4 *Given a network G and two marked nodes s and t, the problem of determining whether there exists a local minimum in which node s is in state $+1$ and node t is in state -1 is NP-hard.*

Proof: We reduce from the PARTITION problem, namely can a given collection of positive integers be bipartitioned such that the sum of values in each subset is the same [GJ79]. We assign the given values as weights w_i, $1 \leq i \leq n$, in the network shown in Figure 3. It is easy to see that the w_i values can be bipartitioned into equal-sum subsets iff the network has a local minimum. $\triangleleft$

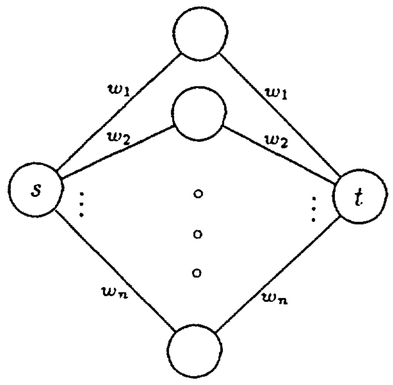


Figure 3: Network equivalent of PARTITION problem

6.2 Global Minimum

It has been observed that minimizing the quadratic cost function J corresponds to maximizing the interpartition sum. Hence, a global minimum for J corresponds to a global maximum for the interpartition sum. Consequently, the Global Minimum problem can be reformulated as follows.

Global Minimum

Input: Edge-weighted graph G

Output: A bipartition H of G such that for all bipartitions H' of G

$$\sum_{e \in E_H} w_e \geq \sum_{e \in E_{H'}} w_e$$

or in other words, a Maximum Cut.

The MAX-CUT problem (see [Kar72]) is a restricted version of this problem in which all the weights non-negative integers.

In this context, Algorithm 1, which simply selects some "unhappy" node and changes its state is simply the well-known Lin-Kernighan local search method described in [PS82]. For instance, this method has been used for the MIN-CUT problem, namely finding a bipartition that minimizes the weighted sum of edges going across the bipartition.

7 Global *vs* Local Search

In this section we argue that in many constraint satisfaction and combinatorial optimization problems it is useful to improve the quality of the local solutions by first running an approximation algorithm that will move closer to a globally good solution. For example, in computer vision applications, where connectionist networks are used for matching (*e.g.*, stereo matching), one would prefer a globally good solution (a high total satisfaction measure but a few nodes may be "unhappy") to a locally good solution (all nodes are "happy") with a lower overall global satisfaction measure. We propose a simple approximation scheme which is guaranteed to achieve a state which is relatively close to a global minimum. If desired, we can then run the local search procedure that will get us to a locally stable state in that neighborhood.

We first introduce the concept of a *feasible partition.* From the condition for a Local Minimum in Section 3 and the discussion in the preceding section, it is clear that a necessary condition for H to correspond to a local minimum is

$$\sum_{e \,\epsilon\, E_H} w_e \geq \frac{1}{2} \sum_{e \,\epsilon\, E_G} w_e$$

Any partition of the vertex set V which satisfies this condition is called a *feasible partition.* Obviously the set of local minima is a subset of the set of feasible partitions.

7.1 The Approximation Algorithm

In this section we present a method of reaching a feasible partition in polynomial time. Indeed it is a $\frac{1}{2}$-approximation algorithm to the MAX-CUT problem. From the viewpoint of finding a local minimum, this algorithm is important because, in polynomial time, it provides a good starting vector for Algorithm 1.

Algorithm 2—The Approximation Algorithm

Input:	W, the weight matrix of G
Output:	(V_1^*, V_2^*), a feasible partition of the vertex set V.
Method:	We use a state vector $\vec{x}$ to represent a feasible partition. The set of vertices having the state label $+1$ constitutes V_1^*, and those with state label -1 form V_2^*. The algorithm considers only the entries above the diagonal in matrix W and does a causal update on the state of each vertex, from n down to 1. The details are presented in Figure 4.

It is clear that Algorithm 2 runs in polynomial time. More importantly, we have the following:

Proposition 5 *Algorithm 2 computes a $\frac{1}{2}$-approximation solution to the MAX-CUT problem.*

Proposition 5 means that if H^* is a bipartition corresponding to a global minimum, and H is a partition obtained by Algorithm 2, then

$$\sum_{e \,\epsilon\, E_{H^*}} w_e \leq 2 \sum_{e \,\epsilon\, E_H} w_e$$

where all weights w_e are non-negative integers.

Several other ideas along similar lines can be used to *provably* improve the performance of this algorithm.

```
algorithm Approximation
  begin
    $x_n \leftarrow 1$
    for $i \leftarrow n$ to 1 step $-1$ do
      $x_i \leftarrow \begin{cases} -1 & \text{if } \sum_{j=i+1}^{n} w_{ij} x_j > 0 \\ +1 & \text{if } \sum_{j=i+1}^{n} w_{ij} x_j \leq 0 \end{cases}$        /* causal update */
    $V_1^* \leftarrow \{i \mid x_i = +1\}$
    $V_2^* \leftarrow V \setminus V_1^*$
  end
$(V_1^*, V_2^*)$ is a feasible partition of vertex set $V$
```

Figure 4: Algorithm 2—The Approximation Algorithm

8 Heuristics

We have investigated several heuristics for speeding up the process of finding a local minimum. One obvious heuristic is always to choose the "least happy" node for update. (See, for example, Papadimitriou and Steiglitz [PS82] for local search algorithms of this kind.) We have performed numerous computer simulations on graphs with randomly chosen edge weights. The results seem to indicate convergence to a local minimum in linear time.

Another simple heuristic which should give us a good estimate for a local minimum is the following:

Heuristic

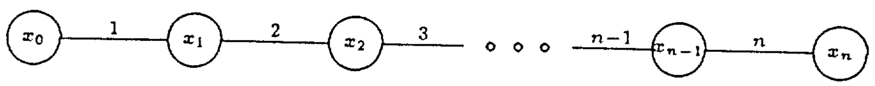
Input:	An edge-weighted graph G
Output:	A good estimate of a local minimum
Method:	1. Find a maximum spanning tree for G based on $\lvert w_e \rvert$ using Kruskal's or Prim's algorithm.
	2. A tree is a bipartitite graph. Find the bipartition. This can be done easily in linear time.
	3. Take this bipartition of the vertex set V as the estimate of the local minimum.

This heuristic should work quite well because very many heavily weighted edges will appear in the bipartition H and, from the discussion in Section 6, will contribute greatly in reducing the cost function J.

9 Exponential Lower Bound for the Greedy Algorithm

The greedy sequential algorithm for local stability in a network appears to work well in practice. In fact, we ran the algorithm for weeks on randomly generated nets and they *always* converged after a linear (in the number of nodes in the network) number of steps (switches from one partition to another). Thus, the results reported in this section are particularly surprising.

We first observe that even a simple chain network such as the one below may exhibit quadratic worst-case behavior



We number the nodes from left to right $x_0, x_1, \ldots, x_n$, and initially allocate all nodes to the same bipartition subset. The quadratic length sequence of flips from one partition to another is as follows:

$$(x_0), (x_1, x_0), (x_2, x_1, x_0), (x_3, x_2, x_1, x_0), \ldots, (x_{n-1}, x_{n-2}, \ldots, x_0)$$

Note that here we are choosing the worst sequence of moves. However, in a totally asynchronous environment this particular behavior is possible. If we change the "firing" rule, *e.g.*, if the most "unhappy" node fires first, this particular chain network can converge in n moves.

We now describe a network that can take an exponential number of steps to converge to a locally stable state. The idea is to construct an n-layer network where nodes on the top layer move once, those on the second layer move twice, and in general, nodes on level i move 2^i times. The network is shown in Figure 5.

Proposition 6 *In the worst case Algorithm 1 takes an exponential number of steps to converge to a local minimum.*

Proof: Apply Algorithm 1 to the network of Figure 5, considering the nodes in the order

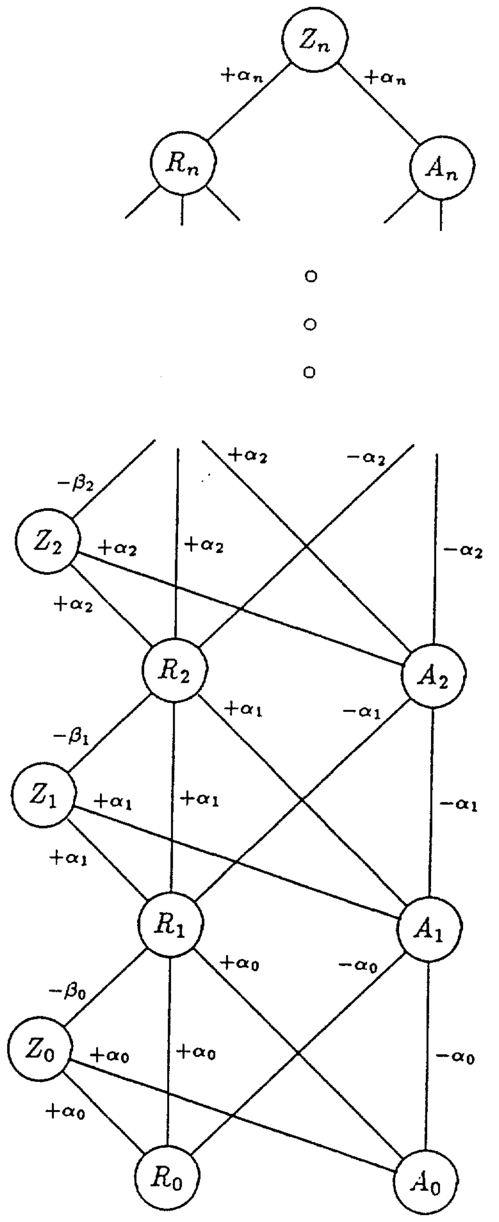
$$A_n, A_{n-1}, \ldots, A_0, R_0, Z_0, R_1, Z_1, \ldots, R_{n-1}, Z_{n-1}, R_n, Z_n$$

$\lhd$ check this!!

The result of Proposition 6 was proved independently by Haken (personal communication, Noga Alon) using a different construction. Note that we again rely strongly on choosing a particular sequence of node movements from one partition to another. As before, if we always move the "most unhappy" node (*i.e.*, the neuron with most local energy) this network converges in a linear number of steps. We have not yet been able to prove exponential time behavior for networks where the unhappiest node is always moved first.

10 Scaling Approach to Local Stability

In this section we discuss another useful approach for finding locally stable states in discrete Hopfield nets. Consider a network that is currently in a stable state. We change



$$\alpha_0 = 1$$
$$\beta_i = 2\alpha_i + 1$$
$$\alpha_i = 2\beta_{i-1}$$

Figure 5: An exponential time network

one of the weights w_e in the network by adding one to it and consider the problem of finding a new stable state. We refer to this problem as the *Incremental Local Stability* problem (ILS). Intuitively, one feels that since the newly constructed network is very similar to the previous one, a new locally stable state could be found in small number of steps. The next proposition shows that if we could solve this problem in polynomial time, we also could solve the problem of finding locally stable sets in polynomial time.

Proposition 7 *The problem of achieving local stability is polynomial-time reducible to ILS.*

Proof: Let G be a DHN with edges $e_1, e_2, \ldots, e_k$ where the weight values $w_1, w_2, \ldots, w_k$ associated with these edges are given by n-dimensional bit vectors. Let G_0 be a DHN where the weights are constructed by considering only the most significant bit of each weight vector in G. We first find a locally stable set for G_0. Clearly, this solution can be found in polynomial time by Proposition 2. Now, create a network G_1 from G_0 by the following procedure: Let G_1 initially be the same as G_0, but with all edge weights doubled. Clearly, G_1 has the same locally stable state. Now, for each edge e_i, if the second significant bit of the original w_i is one (1), add one to the corresponding edge weight in G_1 and invoke the procedure for ILS to find a new stable state for network G_1; otherwise, G_1 is not modified.

Now, repeat the above procedure for the second most significant bit of each edge to create G_2 from G_1 using the procedure or ILS to update the partition for each 1-bit. Continuing in this fashion we find a locally stable state for each G_i, $1 \leq i \leq n$. Since $G_n = G$ we have the desired locally stable state, and since the procedure for ILS gets invoked kn times, our reduction is polynomial in the size of the binary representation of G. $\lhd$

An immediate corollary of Proposition 7 is:

Corollary 8 *If the ILS problem can be solved in polynomial time, then we can solve the local stability problem in polynomial time.*

The proposition above also suggests that the problem of finding a locally stable state for a network is very sensitive to small perturbations. Specifically, if we are given a network in a locally stable state and we modify one of the weights by one bit, the problem of finding a new locally stable state may be as difficult as finding a locally stable state without additional information.

11 Non-Deterministic DHN's

In this section we consider a non-deterministic variant of DHNs proposed by Rao Kosaraju. Consider a DHN G. We do not know whether there is a deterministic polynomial-time procedure to derive a locally stable state for G. However, we can address the question of whether there exists a polynomial-time sequence of legal node flips from one bipartition to the other that achieves a locally stable state. Recall that a node is allowed to change state (move from one bipartition to another) if the node is unstable. This is a direct analog to the class NP, *i.e.*, non-deterministic polynomial-time computations.

Proposition 9 *Let G be a DHN with n nodes and only positive edge weights. If all nodes initially are assigned to the same bipartition, then there is a sequence of at most n node moves that achieves a globally stable state.*

Proof: Assume the bipartition that achieves the global state is given by partitioning the nodes of the G into two sets L and R (left and right), and call the nodes in these sets L-nodes and R-nodes respectively. Without loss of generality, assume all nodes initially are allocated to L. As long as any R-node is still in L and is unstable, we can move it from L to R. We have to show that this process can continue until a global stable state is achieved. Suppose, to the contrary, that we "get stuck" in a locally stable state. Let M be the set of R-nodes still in L; L' the set of all nodes still in L (*i.e.*, $L' = L \cup M$); and R' the set of all nodes in R so far (*i.e.*, $R' = R \setminus M$). Let $W_{LR'}$ be the sum of the weights of edges between nodes in L and R'; $W_{MR'}$ the same sum for edges between M and R'; and W_{LM} the same sum for edges between L and M. Since all nodes in M are currently stable, we conclude that

$$W_{MR'} \geq W_{LM} \tag{1}$$

This is true since for each node in M the sum of external weights is larger than the sum of internal weights. $W_{MR'}$ is the total sum of all external weights for nodes in M, and W_{LM} is smaller than the total sum of all internal weights for nodes in M. However, since the partition of nodes into L and R constitutes a globally stable state, $W_{LR'} + W_{LM}$ is an upper bound for the external sum of any state. Thus, we have:

$$W_{LR'} + W_{MR'} \leq W_{LR'} + W_{LM}$$

or

$$W_{MR'} <= W_{LM} \tag{2}$$

Combining inequalities 1 and 2, we conclude that either the bipartition (L', R') is also a global stable state, or else we have a contradiction. $\triangleleft$

Corollary 10 *Let G be a DHN with N nodes and only positive edge weights. Assume the bipartition that achieves the global state is (L, R) as above. Then, any order of moving R-nodes from left to right will reach a globally stable state.*

Proof: In proving Proposition 9 we showed that at any stage before achieving a globally stable state there exists at least one node x in M that is currently unstable and will therefore move from left to right. However, since all edge weights are positive, moving this particular node from left to right only improves the local energy (*i.e.*, increases the external sum) of every node in M. Thus, we conclude that we could have moved any of the nodes in M instead of moving x. $\triangleleft$

Corollary 10 has an interesting practical implication. If the size of R is relatively large, there are exponentially many paths to a global state without any locally stable states on their path to a global state.

12 Summary

In this section we summarize the main open problems. Time and space (memory) are the fundamental resources in the study of computational complexity of sequential machines. Thus, the first question to be answered is what is the analog of these in the context of connectionist networks. Naturally, time and network size are the two immediate candidates. However, we should also consider the number of states allowed at each of the nodes and the size of the arc-weights that contribute to the complexity of building the networks. Additionally, we may need to account for the number of layers (depth) of nodes in the network and the number of nodes in each layer (width). Connectivity and other graph-theoretical properties of the network may play an important role as well.

As mentioned in the introduction there is an important distinction in the complexity studies of connectionist networks between two related but possibly very different questions.

1. The range of computable functions on a connectionist network. More precisely given a problem of input size N can we construct a connectionist network of size $S(N)$ that can solve the problem in time $T(N)$. This is directly analogous to computational complexity of Turing machine computations as given in terms of tape length and computational time.

2. The computational complexity of problems posed in terms of energy minimization on a connectionist network. That is, given a network comprising N-nodes with maximum weight w_{max}, what is the computational complexity of achieving local/global minima.

These investigations are critical since they will shed light on the question of whether energy minimization on a connectionist network can be done in polynomial time (independent of a specific algorithm employed to solve the problem).

There are several other important problems that we are planning to address in our investigations. The first of these is what is the parallel complexity of obtaining local optima. This question has two important aspects that relate to categories 1 and 2 respectively:

1. Can we devise highly parallel updating procedures that will be guaranteed to converge to a local minima?

2. Can we devise a parallel algorithm (not necessarily a network algorithm) to find local minima?

As mentioned before Luenberger devised an efficient parallel algorithm for a special case of the connectionist network: the maximal independent set problem [Lue84].

Once the fundamental questions above are resolved there are many interesting questions that address the range of computations admitted by networks with given structural (graph-theoretical) properties. Of special interest are planar, bounded-degree and positive-weight networks. For instance, we already have discovered a polynomial time algorithm for the problem of finding local minima in tree-like networks. The next natural set of questions is related to complexity of the learning procedures.

To summarize, we believe that a systematic theoretical treatment of the computational aspects of massively parallel networks is a necessary prerequisite for their acceptance as a feasible computational mechanism. Progress in this area will advance our knowledge of highly parallel machines and their applicability to perform complex tasks.

Acknowledgements

Thanks are due to Rao Kosaraju, Gary Miller, Noga Alon and Adi Shamir for their constructive comments that contributed greatly to the final form of this paper. The initial stages of the research reported in the paper were done while the third author was a visiting scientist at the Weizmann Institute of Science, Rehovot.

References

[BH86] D. H. Ballard and P. J. Hayes. Parallel logical inference and energy minimization. Technical Report 142, Computer Science Dept., University of Rochester, March 1986.

[Fel85] D. A. Feldman. Energy and the behavior of connectionist models. Technical Report 155, Computer Science Dept., University of Rochester, November 1985.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, 1979.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, pages 2554–2558, April 1982.

[HSA84] G. E. Hinton, T. Sejnowski, and D. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Carnegie-Mellon University, 1984.

[HT85] J. J. Hopfield and D. Tank. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[LK73] S. Lin and B. Kernighan. An effective heuristic for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[Lub85] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *ACM Symposium on the Theory of Computation*, 1985.

[Lue84] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[RHW85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. Technical Report 8506, Institute for Cognitive Science, University of San Diego, 1985.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* MIT Press, Cambridge, MA, 1986.