Chicago Journal of Theoretical Computer Science

The MIT Press

Volume 1999, Article 6

Hopfield Neural Networks and Self-Stabilization

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,*® *SciSearch,* ® *Current Contents*®*/Engineering Computing & Technology*, and *CompuMath Citation Index.*®

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

Hopfield Neural Networks and Self-Stabilization

Arun Jagota
Department of Computer Science
University of California, Santa Cruz
`http://www.cse.ucsc.edu/~jagota`

6 August 1999

Abstract

This paper studies Hopfield neural networks from the perspective of self-stabilizing distributed computation. Known self-stabilization results on Hopfield networks are surveyed. Key ingredients of the proofs are given. Novel applications of self-stabilization—associative memories and optimization—arising from the context of neural networks are discussed. Two new results at the intersection of Hopfield nets and of distributed systems are obtained: One involves convergence under a fine-grained implementation; the other is on perturbation analysis. Some possibilities for further research at the intersection of these two fields are discussed.

1 Introduction

Study of stability issues in systems began centuries ago, and convergence properties of differential equations have long been examined. In modern times these have been expanded to include studies on other kinds of dynamical systems (difference equations, numerical algorithms, and so on). Control theory and numerical analysis are two other examples of important fields in which stability issues have been of fundamental importance since the early part of the twentieth century. Convergence is also an important issue in statistical physics, where a central question is whether a spin system will settle into a low-energy state.

In distributed computing, studies of stability issues are regarded as having begun around 1974 when Edgar W. Dijkstra [Dij74] introduced the concept of self-stabilization. At an abstract level Dijkstra's concept was not much different from usual notions of convergence invented earlier in various fields. At the concrete level of the distributed computing system, however, it was a new development. First, distributed computing systems usually involve different kinds of computational primitives than those in classical fields (differential equations, dynamical systems, statistical physics). Consequently, the design and analysis of self-stabilizing algorithms for them require new techniques. Second, convergence results in classifical fields hold at a high level. In contrast, in distributed computing systems, one can address the issue of stabilization at various levels (coarse-grained to fine-grained implementation) and also for various structures (network topologies). Paying attention to the level and structure raises new and important questions.

In 1982, John J. Hopfield [Hop82] independently studied a globally convergent neural network model that fits nicely within the framework of Dijkstra's self-stabilization. Self-stabilization results and applications of this neural network model can be easily interpreted in the context of distributed systems, thereby raising the possibility for an exchange of ideas in the two fields.

This paper introduces this neural network model, surveys its results involving self-stabilization, and explains key ingredients of the proofs. Variations of the self-stabilization concept appropriate in the neural networks context are presented. These are compared in some cases to analogous variants invented in the distributed systems field.

The paper is organized as follows. Section 2 presents basic definitions, including those of the self-stabilization concept and its variants. Section 3 presents a two-state version of the Hopfield network, its self-stabilization results (Sections 3.1, 3.2, and 3.3), the notion of attraction domains (Section 3.4), and new applications of self-stabilization arising from this context (Section 3.5). Section 4 presents a randomized version of the two-state Hopfield network, together with a kind of probabilistic self-stabilization result for it. Section 5 presents a continuous-state version of the Hopfield network and a form of asymptotic self-stabilization result for it. Section 6 investigates what happens to self-stabilization of the two-state network of Section 3 under a fine-grained implementation. Section 7 performs some perturbation analysis on the two-state network of Section 3. Section 8 summarizes the contributions of this paper and presents possibilities for further research at the intersection of these two fields.

2 Basic definitions

The concepts used in this paper are at the intersection of the fields of distributed computing and neural networks.

A *distributed system* is a set of n processes $V = \{1, \ldots, n\}$ interconnected pairwise by a set of edges E to form an undirected graph $G = (V, E)$.[1] Each process i has a *set of states* α_i. The *set of system states* is

$$\Sigma = \alpha_1 \times \alpha_2 \times \cdots \times \alpha_n$$

and a subset of this set $L \subseteq \Sigma$ is called the set of *legal states*.

A distributed system computes as follows. A process examines its current state and received messages to determine its next state. Following its state transition, a process may send messages to (some of) its neighbors. Messages travel only on edges of the underlying graph G.

A distributed system is called *self-stabilizing* to a set of system states $L \subseteq \Sigma$ if, starting from any initial state $S \in \Sigma$, (i) the system eventually enters a state in L and (ii) the subsequent state changes (if any) confine the system state to remain within L.

Dijkstra [Dij74] introduced this concept for the design of distributed mutual-exclusion algorithms. One useful property of self-stabilizing systems is that they are inherently fault-tolerant. For example, if a transient fault in the state of some process "kicks" the system state out of the legal region L, the system eventually reenters L on its own. Thus, the system recovers from such faults autonomously, without external intervention.

In the context of neural networks, the following refinement is also useful. A distributed system is called *k-bounded self-stabilizing* to a set of system states $L \subseteq \Sigma$ partitioned into sets $L_k^1, L_k^2, \ldots$ of cardinality at most k if, starting from any initial state $S \in \Sigma$, (i) the system eventually enters a state in L_k^j for some j and (ii) thereafter remains confined within L_k^j.

[1]In neural networks it is customary to think of a distributed system as being composed of processors rather than processes.

A k-bounded self-stabilizing system is self-stabilizing, but the opposite is not necessarily true. A 1-bounded self-stabilizing system converges to a fixed point.[2] The set L and its partitioning into $L_k^1, L_k^2, \ldots$ is either implicit or prespecified from an application.

A distributed system is called *serial* if, at any given time, at most one process carries out its computation (determining its next state from its current state and received messages; sending messages to its neighbors). The process to carry out its computation is assumed to be selected *fairly.* Only a relatively mild version of fairness is assumed: In any infinite sequence of chosen processes, each process appears infinitely often. A distributed system is called *parallel* if, at any given time, all processes carry out their computations simultaneously.

3 The two-state Hopfield network

Hopfield's [Hop82] 1982 neural network model sparked interest in the neural network and statistical physics communities. The model has been implemented repeatedly in specialized hardware: digital, analog, optical, or hybrid (see [HKP91]).

The model is defined as follows. There are n vertices, representing neurons, connected pairwise by a complete undirected graph G with loops. An integer-valued weight w_{ij} is placed on each edge $\{i, j\}$ of the graph G, an integer valued $w_{ii} \geq 0$ on every loop $\{i, i\}$ of the graph, and an integer valued w_i on each vertex i of the graph G. The graph is assumed to be complete for notational convenience only. The true connectivity graph underlying a given network is one that contains edges only for the nonzero edge weights w_{ij}.

Each vertex i has the same state-set $\alpha_i = \{-1, 1\}$.[3] Let s_i denote the state of process i. The Hopfield network employs the following state-update rule for the process i.

$$s_i(t+1) := \begin{cases} 1 & \text{if } w_i + \sum_{j=1}^{n} w_{ij} s_j(t) \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{1}$$

[2]Convergence of a system to fixed points has also been studied in the distributed systems community; see [Sch93].

[3]An alternate formulation has the state set $\alpha_i = \{0, 1\}$.

3.1 The serial case

When (1) is applied serially, the Hopfield model 1-bounded self-stabilizes to a set of system states L_{fp}, called the fixed points of the network.[4]

Hopfield [Hop82] proved this result by using the energy function

$$E(S) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i s_j - \sum_i w_i s_i, \tag{2}$$

which is bounded from below, and by showing that whenever a process *changes* its state following the application of (1), the energy E decreases by a fixed minimum amount.

This result holds for networks with arbitrary connectivity graphs, arbitrary edge weights w_{ij}, arbitrary vertex weights w_i, and nonnegative loop weights w_{ii}. Counterexamples, when even one loop weight is negative or when even one edge in the connectivity graph is directed (i.e., when $w_{ij} \neq w_{ji}$ for some $i \neq j$), are easy to construct (see, e.g., [Par94, Chap. 8]).

The serial Hopfield network is easily realizable in a serial distributed system, as defined in Section 2, which is hence 1-bounded stabilizing.

3.2 The parallel case

When (1) is applied in parallel, the Hopfield model 2-bounded self-stabilizes to a set of system states $L \supseteq L_{fp}$, which, in addition to the fixed points L_{fp} of the network, includes pairs of states that form 2-cycles (refer to [GFSP85]). The proof once again relies on the energy function approach. Goles et al. [GFSP85] defined the following interesting energy function, which depends not only on the current system state but also on the previous one:

$$F(S(t), S(t-1)) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i(t) s_j(t-1) - \sum_i w_i (s_i(t) + s_i(t-1)).$$

They showed that F is bounded from below and decreases by some minimum amount whenever $S(t) \neq S(t-1)$ and $S(t) \neq S(t-2)$, and is zero otherwise, implying convergence to a fixed point or to an oscillation between two states (2-cycle).

[4]The terms "serially" and "1-bounded self-stabilizes" are used here precisely as defined in Section 2.

This result holds under the same conditions on the network as does the serial 1-bounded self-stabilization result. Counterexamples showing the conditions to be tight are once again easy to construct (see [Par94, Chap. 8]).

Goles et al. [GFSP85] also proved that when the weight matrix W is symmetric, has nonnegative diagonal elements, and is positive-definite on the set $\{-1, 0, 1\}^n$, the parallel network is 1-bounded self-stabilizing.

3.3 Number of state changes to stabilization

How may one calculate the maximum number τ of state changes within which a network stabilizes? The energy function is useful here, too. Let $\Delta = \max_{S,T\in\Sigma}(E(S) - E(T))$ denote the maximum difference between the energies at two system states. Let δ denote the minimum amount by which energy decreases when a state changes. Then Δ/δ is an upper bound on τ.

Using this idea, Fogelman-Soulie et al. [FSGW83] obtained upper bounds on τ for the serial case. This description is summarized from [KH90] where it differed, in minor ways, from the original version. Let M denote the sum of the $\lfloor n^2/4 \rfloor$ largest coefficients $|w_{ij}|$, $i \neq j$, and let $w = \min_i w_{ii}$. For the serial case, the result is that

$$\tau \leq \frac{2(M + \sum_i |w_i|)}{1 + 2w}.$$

If $|w_{ij}| \in \{0, 1\}$ for all i, j, then the result is $\tau \leq 5n^2/2$.

Goles et al. [GFSP85] obtained the analogous bound on τ for the parallel case. If W is positive-definite on the set $\{-1, 0, 1\}^n$, then the result is

$$\tau \leq \frac{M + \sum_i |w_i|}{\lambda_{\min}},$$

where $\lambda_{\min}$ is the smallest eigenvalue of W. In the general case, the result is that

$$\tau \leq 4 \sum_{i=1}^{n} \sum_{j>i} |w_{ij}| + 2 \sum_i w_{ii} + 4 \sum_i |w_i|.$$

If $|w_{ij}| \in \{0, 1\}$ for all i, j, then the result is $\tau \leq 6n^2$. Floreen [Flo92] obtained a small improvement to the unrestricted serial result and a somewhat larger improvement to the unrestricted parallel result.

It should be noted that though all the above results are based on the simple idea of computing Δ/δ, the actual calculations are often tricky, because a good upper bound on Δ is not easy to compute for this family of neural networks.

Finally, we remark that the notion of "number of state changes to stabilization" is, in essence, the same as the *convergence span* of the distributed systems community (refer to [GE88, Sch93]).

3.4 Attraction domains

For a deterministic distributed system that is 1-bounded self-stabilizing, the system-state set Σ may be partitioned into sets $AD(S_1), AD(S_2), \ldots$ where $S_1, S_2, \ldots$ are all the fixed points in L_{fp}. The set $AD(S_f)$ is called the *attraction domain* of the fixed point S_f and is defined as the set of initial states from which the system eventually converges to S_f (see [KH90]).

Although this definition is easily generalized to the k-bounded case, it is most natural to study it in the $k = 1$ case. In particular, it arises naturally in the context of one of the application areas of the Hopfield network—associative memories—described in the next section. In this context, $B(S_f)$ represents the set of "cues" from which a given memory S_f can be retrieved. The larger the set $B(S_f)$ is, the more effective the model is for memory S_i.

The attraction domains of memories stored in the parallel two-state Hopfield network, and variants, have been studied extensively under certain conditions (refer to [MPRV87, KH90]). Unfortunately, this issue is quite complicated, since the attraction domains depend on the following:

(i) The storage mechanism employed: how the weights w_{ij}, w_{ii}, and w_i are derived from the given memories

(ii) The assumptions on the memories: whether they are chosen randomly or are chosen to draw out worst-case behavior

(iii) The number of memories

The attraction domain results cannot be interpreted meaningfully without a detailed discussion of these issues. Since these are less central to the theme of the current paper, we omit their discussion and, hence, detailed presentation of attraction domain results.

3.5 Applications

In his seminal 1982 paper, Hopfield proposed this neural network as a model for *associative memory* [Hop82]. We explain this briefly as follows. Hopfield suggested that a set of memories P could be stored in a network by constructing its edge-weight matrix W and vertex-weight vector w in such a way that the resulting set L_{fp} of fixed points coincides with the set P. Thus each fixed point of the network represents a memory from the set P. To retrieve a memory $S_f \in L_{fp}$ from a cue $S_i \in \Sigma$, the serial operation is used, with the initial state set to S_i. (Recall that, unlike the parallel operation, the serial operation always converges to a fixed point from an arbitrary initial state.)

To construct the matrix W and vector w from a given set of memories $X_1, \ldots, X_p$, $X_i \in \{-1, 1\}^n$ for $i = 1, \ldots, p$, Hopfield [Hop82] proposed the *Hebb rule*, which sets

$$W := \sum_{i=1,\ldots,p} X_i X_i^T,$$

where T denotes transpose. The vector w is set to zero. Unfortunately, the Hebb rule does not work very well in general (L_{fp} can differ greatly from P). In fact, not surprisingly, it is impossible to construct a perfect network (i.e., with $L_{fp} = P$) for an arbitrary pattern set P, no matter how one chooses W and w (see [AMJ85]). A considerable amount of research has concentrated on alternatives to the Hebb rule for storing memories in the Hopfield model. We omit the details here; interested readers may examine, for example, [AMJ85, MPRV87, Ama89, HKP91].

In 1985, Hopfield and Tank [HT85] proposed another application area, *quadratic optimization*, for this neural network. We illustrate the case of combinatorial optimization. To solve a combinatorial optimization problem (usually approximately) in a Hopfield network, one first maps the problem to a quadratic objective function on $-1/1$ variables with symmetric coefficients. One then identifies this objective function with the energy function of the network in such a way that the minima of the energy function correspond to optima of the objective function. One then derives the edge-weight matrix W and the vertex-weight vector w from the energy function, yielding the network. Finally, one operates the network in serial mode from some initial state. Since serial operation always reduces energy monotonically, it eventually settles into a local minimum of the energy function (hence local optimum of the mapped problem).

Here is a sketch of an example. Consider the *maximum clique problem*, the problem of finding the largest fully connected subgraph of a given graph on n vertices. To express this problem in terms of an appropriate objective function, we first choose n variables $x_i \in \{0, 1\}$. We then make the correspondence $U(x) = \{i | x_i = 1\}$ between a vector x and a subset $U(x)$ of the vertex set V of the graph. We write out the objective function

$$F(x) = \sum_{i=1,\ldots,n} x_i \ + \ f(x),$$

where the first term encodes the size of the set $U(x)$ and the second term is chosen so as to enforce the constraint that U be a clique. The resulting $F(x)$ must be expressible as

$$F(x) = \sum_{i,j} a_{ij} x_i x_j + \sum_i a_i x_i,$$

where $a_{ij} = a_{ji}$. We identify F with the energy function E of (2) to derive the weights w_{ij} and w_i. And then we operate the resulting network serially to retrieve a fixed point (local minimum of E) x^* from which we can extract the clique. In [JG] it is shown that there is a one-to-one correspondence between maximal cliques of the graph and fixed points of the network for one such encoding. In [JSG96] an extensive experimental study is conducted on heuristically solving the maximum clique problem by this method.

This general approach is not without its difficulties. For many combinatorial problems, good encodings are not known (see [WP88]). (An encoding is bad, for instance, if some of the network's fixed points do not constitute feasible solutions of the original problem.) This has led to a fair amount of research on characterizing optimization problems that map well to the Hopfield model (see [KA89, JG]). There is also considerable work on using other variants of the Hopfield model on this problem (e.g., the randomized and continuous versions; see Sections 4 and 5).

Notwithstanding the difficulties of finding good encodings and the fact that the network performs only local optimization, this approach is attractive from the point of view of distributed computing and new applications of self-stabilization. Indeed, in theory, every problem in NP may be mapped to the Hopfield network. (And many have been mapped in practice.) Thus a self-stabilizing meta-algorithm exists, in principle, for locally optimal versions of every NP problem.

Here is a partial list of combinatorial problems that have been encoded and approximately solved in Hopfield networks: We have already mentioned the maximum clique problem, an NP-hard problem; see [JSG96, Gro96]. The traveling salesperson problem was studied in [HT85, HKP91], the graph coloring problem in [KA89, Jag96], and the graph bipartitioning problem in [PA88]. Virtually all of the above works simulate the networks on sequential machines. In most cases the neural net algorithms are not entirely competitive with state-of-the-art conventional ones. However, the former algorithms offer the potential benefits of distributed computing and self-stabilization, which deserve further study.

4 A locally randomized version

The processes in the two-state Hopfield network of the previous section employ a deterministic state transition function. In this section, the processes employ a certain randomized state transition function (refer to [HKP91]).

Let

$$h_i = w_i + \sum_{j=1}^{n} w_{ij} s_j(t).$$

Then the state transition function is

$$\mathrm{Prob}(s_i := \pm 1) = g_T(h_i), \tag{3}$$

where

$$g_T(h_i) = \frac{1}{1 + e^{\mp h_i/T}}.$$

Here T is a "temperature" parameter that controls the amount of randomness. When $T \to 0$, (3) reduces to the deterministic version (1), which always decreases energy. When $T \to \infty$, the events $s_i := 1$ and $s_i := -1$ are equally probable; thus energy-increasing state-transitions are also admissible.

The state transition function $g_T(h_i)$ is the same one employed in simulated annealing; its choice is dictated by the fact that this governs the evolution of the network in accordance with the principles of statistical mechanics. Indeed, if T is annealed during the evolution of the randomized Hopfield network, then the network emulates simulated annealing.

In the context of this section, the following definition is useful. Assume that the state set α_i of the process i contains numeric quantities. Let $\langle s_i \rangle$

denote the expected value of the state of the process i. A distributed system is called *1-bounded self-stabilizing on average* if, starting from any initial state $S \in \Sigma$, the system eventually reaches a point after which the system state does not change on average, namely, that the n-tuple $(\langle s_1 \rangle, \ldots, \langle s_n \rangle)$ remains fixed. Note that this definition is rigorous; the question of what time scale over which to measure "average" is an operational one.

This definition is different from that of *probabilistic self-stabilization* introduced in the distributed systems community (see [Sch93]). A probabilistically self-stabilizing distributed system, started from an arbitrary initial state, eventually enters L with probability approaching 1 and once inside L never escapes it. By contrast, a 1-bounded self-stabilizing on average system does not enter any region L at all. One way to interpret this is to say that a fixed region L has been replaced by a certain statistic on it—in our case, the average value of each state. It is the statistic that should converge, not the state.

It is known that the locally randomized Hopfield network is 1-bounded self-stabilizing on average in the serial and parallel cases (see [AK89], [HKP91, p. 33]). Indeed, in essence, this may be surmised from the well-known fact that the simulated annealing algorithm converges in the average sense.

Is it enough to say that a locally randomized Hopfield network merely emulates simulated annealing when T is annealed? It is useful to add the following. The network constitutes a *distributed* implementation of simulated annealing. On the negative side, this implementation works only for (quadratic) cost functions that can be mapped to the energy function E of the previous section.

5 A continuous-state version

In 1984, Hopfield introduced a version of the network that employs continuous-valued process states [Hop84]. The state set of each process is $\alpha_i \in [0, 1]$. The evolution of the network is governed by the following system of n nonlinear coupled differential equations:

$$\frac{ds_i}{dt} = -s_i + g_T(h_i), \tag{4}$$

where $g_T(h_i)$ and h_i were defined in the previous section.

In practice, a network cannot be evolved according to (4). And distributed systems, as studied in computer science, work in discrete time steps. For these two reasons, the following discrete-time (but still continuous-state) version of (4), an Euler approximation, is more natural to study:

$$s_i(t+1) := s_i(t) + \gamma_i(-s_i(t) + g_T(h_i)), \tag{5}$$

where γ_i is the Euler step size taken by the process i.

To discuss stability results for continuous networks, the following definition, cast in terms of distributed systems, is useful. For this definition, the state space Σ must be a metric space. A distributed system is called *asymptotically 1-bounded self-stabilizing* to a set $L_{fp} \subseteq \Sigma$ if, starting from any initial system state $S_i \in \Sigma$, the network converges in infinite time to some state $S_f \in \Sigma$, that is,

$$\lim_{t \to \infty} S(t) = S_f.$$

In other words, for any $\epsilon > 0$ there exists a δ such that the system enters an ϵ-neighborhood of S_f within the time δ.

Hopfield showed that, for a slightly different system of equations than (4) but with the same set of fixed points, the continuous-time network is asymptotically 1-bounded self-stabilizing, in both serial and parallel operations.[5]

The proof works also for the system of equations (4). Note that, in parallel operation, the *continuous*-state continuous-time Hopfield model is 1-bounded self-stabilizing, whereas the *two*-state model is 2-bounded self-stabilizing. This difference is significant because both the associative memory and the optimization applications of the Hopfield model rely on convergence to fixed points only.

Hopfield proved this result by using the energy function

$$H(S) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}s_is_j - \sum_i w_is_i + \sum_i \int_0^{s_i} g_T^{-1}(s)ds,$$

which differs from $E(S)$ only in the last term. The function $H(S)$ is bounded from below, and Hopfield showed that $dH/dt \leq 0$ during the evolution of the network, according to the variant of (4) mentioned above, with equality holding if and only if the network state is at a fixed point.

As noted earlier, in practice, the continuous-valued Hopfield network is usually evolved via (5). What happens in this case? When the step sizes

[5]See [Hop84, HKP91]. The system of equations is not explicitly described here because it adds little new information but requires considerable explanation.

$\gamma_i \to 0$ for all i, (5) behaves like (4), preserving asymptotic 1-bounded self-stabilization. When the step sizes $\gamma_i = 1$ for all i, Marcus and Westervelt [MW89] showed that the network is in general no longer 1-bounded self-stabilizing in parallel mode—it converges instead to fixed points or to 2-cycles, in analogous fashion to the two-state parallel network.

Herz and Marcus [HM93] generalized these results to what they called the distributed update mode in which, at time t, an arbitrary set of neurons is chosen to update its states. (This so-called distributed mode is still at the coarse-grained level, unlike our analysis in Section 6.) Recently, Wang et al. [WJBG96] studied the general case of (5). They showed that serial evolution of (5) is asymptotically 1-bounded self-stabilizing over all values of $\gamma_i \in [0, 1]$. They obtained a sufficient (in many cases also necessary) condition for a network evolved in parallel via (5) to be asymptotically 1-bounded self-stabilizing. The condition requires that a certain matrix derived from the weight matrix W, the Euler step-sizes γ_i, and the temperature T be positive-definite. Special cases of this condition include the Goles et al. [GFSP85] condition for the two-state case and the Marcus and Westervelt [MW89] condition when $\gamma_i = 1$ for all i. In an expanded version of this paper, they also obtained self-stabilization results for the distributed dynamics of Herz and Marcus applied to (5) for arbitrary $\gamma_i \in [0, 1]$.

Finally, a version of the continuous-time update (4) has been studied that, in a sense, is more asynchronous. Wang, Li, and Blum [LWB94] permitted each neuron s_i to have its own local time (there was no global time) at which it updated itself. They also incorporated transmission time-delays in their update algorithm. In principle, their approach captures the main ingredients of a fine-grained implementation. However, (i) the analysis is only for the continuous-time case and (ii) it imposes strong conditions to assure convergence.

The continuous Hopfield network is applicable to the discrete associative memory and discrete optimization applications of the two-state network (refer to [HKP91]), as a continuous algorithm with the usual caveats. The continuous Hopfield network is also applicable to a continuous version of the associative memory problem, as well as to continuous optimization problems (see [HKP91]). All applications, once again, rely on convergence to fixed points only. In an ideal continuous-time implementation, this property is preserved even during parallel evolution of the network, which turns out to be an advantage, in one sense, over the two-state network. An ideal implementation is, however, not realizable. In practice, the network must be evolved in discrete time. Our results, which provide a sufficient condition for asymptotic 1-bounded self-stabilization of the parallel discrete-time network, are therefore potentially useful in the context of these applications. We have noted [WJBG96], for example, that a certain family of networks commonly used in associative memory applications meets our sufficient condition and therefore converges to fixed points under parallel operation.

6 Convergence under fine-grained atomicity

The various convergence results of the previous sections rely on the assumption that the operation given by (1) is atomic. In this section we examine what happens to convergence under finer-grained atomicity.

Each process's code at a fine level is as follows:

```
Process $i$
loop
1.       Read $s_1(t)$
2.       Read $s_2(t)$
         ...
$n$.      Read $s_n(t)$
$n+1$.    Compute next state $s_i(t+1)$
forever
```

What can we say about convergence when individual instructions of different processes can get interleaved in time? We now examine this issue in the context of the two-state Hopfield model.

One natural question to ask in this context is: Will convergence to fixed points continue to hold, like it did under coarse-grained atomicity, for any fair schedule? The answer is No. Consider the schedule in which, in phase I, processes are activated in cyclic order (e.g., 1 through n), and each of them executes its steps 1 through n. Once this is finished, in phase II, each process executes step $n+1$ (i.e., computes its next state). This schedule simulates parallel operation at the coarse level, which is known to converge to 2-cycles. (Examples illustrating 2-cycle behavior are easy to construct.)

On the other hand, there are at least some fair, deterministic schedules under which convergence to fixed points is assured. Here is one: In phase I, activate processes cyclically as before, but this time make each of them execute steps 1 through $n + 1$ (instead of 1 through n). This schedule simulates fair sequential operation at the coarse level, which is convergent to fixed points.

The deterministic schedule used above essentially reverts back to coarse-grained atomicity. A more realistic situation would be one in which processes activate themselves autonomously at arbitrary times. Fortunately, by a simple modification of each process's program, we can accomodate autonomous operation of the distributed system while preserving convergence to a fixed point (although in infinite time instead of finite time as in the coarse-grained case). Each process's modified program is the following:

```
Process $i$
0.       With probability $p$
0.'           sleep for $\delta$ time-units;
0.”           goto 0;
1.       Read $s_1$
2.       Read $s_2$
         ...
$n$.       Read $s_n$
$n+1$.   Compute new state $s_i$
```

Process i activates itself autonomously at an arbitrary time (there is no global clock) and begins executing from step 0. The pauses are all before step 1, because it suffices to pause only before any of the states of the other processes are read. We assume, furthermore, that the total time any process takes to execute its steps 1 through $n + 1$ (including communication time to read the states of other processors) is bounded by some function $\Delta(n)$ of n. This is quite a mild assumption.

The parameter δ controls the extent of the sleep time, and its setting is governed by the communication delays. The parameter p probabilistically allows different processes to sleep for different amounts of time, breaking any pathological synchronies. More precisely, the combination of these parameters allows one to show that the event E_v used in the proof below occurs asymptotically with probability 1. The boundedness assumption $\Delta(n)$ ensures that p and δ can be chosen appropriately (see proof below).

We are now ready to show that this distributed system almost surely stabilizes to a fixed point in infinite time.

Definition 1 (Adapted from [Sch93]) *A distributed system is called* 1-bounded probabilistically self-stabilizing *if, started from an arbitrary initial state $S \in \Sigma$, the system enters a fixed point with probability approaching* 1 *as time t approaches infinity. Furthermore, once the system enters a fixed point, it remains there.*

Proposition 1 *The distributed system described above is 1-bounded probabilistically self-stabilizing.*

Proof Consider the sequence

$$E_v(t_0) \equiv (1, 0, \ldots, 0, 2, 0, \ldots, 0, \ldots, n, 0, \ldots, 0)^{2^{n+1}}$$

denoting the event that, starting at some time t_0, processes activate themselves in order 1 through n cyclically 2^{n+1} times. In this event, at a time t when a process activates itself, no other process activates itself. Furthermore, there is a gap (indicated by the 0's) of at least $\Delta(n)$ time-units between consecutive activations. Thus, in a $\Delta(n)$ time-interval, only one process is active.

The proof hinges on showing the following to be true:

1. If event $E_v(t_0)$ occurs starting at some time t_0, then a fixed point is eventually entered

2. In the limit as t goes to infinity, the event $E(t_0)$ occurs with probability approaching 1 starting at some $t_0 \leq t$

We first prove item 1. If event $E(t_0)$ occurs, then it simulates serial operation of the two-state Hopfield network (at the coarse-grained level) for its duration. The duration of each inactivity gap, coupled with our boundedness assumption about each process's activity duration, ensures this. The remaining question is whether coarse-grained serial operation is simulated for a sufficiently long duration that a fixed point is necessarily entered. Since a two-state Hopfield model has 2^n states, there can be at most 2^n state changes during coarse-grained serial operation before a fixed point is entered. (Otherwise, a cycle would be entered, contradicting the coarse-grained convergence result.)

It is easy to see that if the cyclic serial operation of the event $E(t_0)$ runs for 2^{n+1} coarse-level steps, then there must necessarily have been 2^n state-changes. (If in any n contiguous activations there are no state changes, then we must be at a fixed point.)

We now prove item 2. It is easy to see that, for any particular t_0, the probability of event $E(t_0)$ is lower-bounded by some positive function that depends only on n, p, and δ. (It is not difficult to estimate this probability more explicitly, but this is not needed.) Therefore, for fixed positive values of n, p, δ, the probability that the event $E(t_0)$ occurs for some t_0 between 1 and infinity tends to 1.

Finally, it is easily seen that once the system enters a fixed point it remains trapped there forever. (No process changes state no matter what the subsequent order of interleaved operations.) ■

A few remarks are in order.

- Like in [Sch93], though entry into L is probabilistic (with probability approaching 1 as time t approaches infinity), escape from L is impossible.

- In [GHR90, Sch93] there is an investigation of the robustness of self-stabilization across granularity of implementation. Some analogous remarks in the setting of Hopfield networks are in order. On the one hand, the two-state Hopfield model is fragile in going from a coarse- to a fine-grained implementation, since convergence is not preserved for every fair schedule. On the other hand, it is also moderately robust, since there is a reasonable probabilistic schedule under which convergence to a fixed point is assured, almost surely, in infinite time.

- The proof does not appear to be extendible to the continuous-state Hopfield network. The proof seems to rely crucially on the fact that for the two-state Hopfield model, convergence is assured within 2^n state changes.

- The proof is quite loose. It is based on an event that has a very low (though nonzero) probability of occurrence. Perhaps tightening this argument will lead to a stronger result. Preliminary computational experiments we conducted on fine-grained implementations of random Hopfield networks of up to a 100 neurons revealed that convergence to fixed points was quite rapid.

7 Perturbation analysis

In many situations, the guarantee that a system when perturbed out of its legal region will *eventually* enter it again is insufficient. If the perturbation is small, one desires the system to return to its legal region quickly. The concept of self-stabilization does not deal with this issue. This has given rise to the notion of systems that not only self-stabilize but also have the *fault-containment* property (refer to [GG, GGP96, GGHP96, GGP97]). Fault-containment, roughly speaking, refers to a self-stabilizing system that, when a few processes get faulty, restabilizes after only a few state changes.

A similar issue may be studied in the setting of serial Hopfield networks. Let $S^* \in \{-1,1\}^n$ be a fixed point of a Hopfield network instance operated in serial mode. Perturb S^* by flipping any k of its bits. We want to bound the number of state changes to reconvergence.

As in earlier sections, the analysis hinges on energy function arguments. We obtain an upper bound on the energy increase when k bits of a fixed point are flipped. Together with lower bounds on the global energy minimum and on the amount of energy change in any state change, this allows us to place an upper bound on the number of state changes to reconvergence.

Assume that $|w_{ij}| \leq W_{\max}$ for all i, j and that $|w_i| \leq w_{\max}$ for all i. Denote by S^*_k the vector obtained by flipping any k bits of a fixed point S^*. It may be seen that

$$E(S^*_k) \leq E(S^*) + 2k(nW_{\max} + w_{\max}),$$

since flipping any one bit can increase the energy by at most $2(nW_{\max}+w_{\max})$ because there are n neurons and because of the assumed bounds $W_{\max}$ and $w_{\max}$ on the magnitudes of the weights. Now we write for the fixed point S^*

$$E(S^*) \leq E_{\min} + f(n, W, w),$$

where $E_{\min}$ is the global energy minimum of the network; f is some function that upper-bounds the gap between the minimum and maximum energy values of local energy minima and whose value is determined by the network instance $\langle n, W, w\rangle$; W is the matrix of the edge weights; and w the vector of the vertex weights. Therefore

$$E(S^*_k) \leq E_{\min} + 2k(nW_{\max} + w_{\max}) + f(n, W, w).$$

Assume that ΔE, the minimum amount of energy decrease in any one state-changing step, is no less than a positive function $c(n, W, w)$. Then the number of state changes to reconvergence is less than or equal to

$$\frac{2k(nW_{\max} + w_{\max}) \; + \; f(n, W, w)}{c(n, W, w)}. \tag{6}$$

We now apply (6) to a particular class of networks. In applications of two-state Hopfield nets to combinatorial optimization, many encodings have been found in which $|w_{ij}| \in \{0, 1\}$ and $w_{\max}$ and $c(n, W, w)$ are constants (see [KA89, JG]). Furthermore, in many cases, for instance, in an encoding of the maximum clique problem into a Hopfield network (refer to [JG]), one can show that $f(n, W, w) = O(n)$. In such cases, from (6), the number of state changes to reconvergence is thus $O(kn)$.

8 Discussion

This paper describes a self-stabilizing neural network model developed in the early 1980s. The model fits naturally within the usual framework of distributed systems. This paper surveys a large body of research on this model and its variants over the past fifteen years. The survey focuses on the important variants, their self-stabilization results, and novel applications of self-stabilization arising from this neural network setting. Two new results at the intersection of neural networks and distributed systems are also obtained. The first is a probabilistic convergence result for a fine-grained distributed implementation of this network. The second one deals with perturbation analysis. These results may be seen as some indication of the possibility of further research at the intersection of these two fields.

It is hoped that this paper spurs cross-fertilization of ideas between the distributed systems and neural network communities in the context of self-stabilization. Perhaps the variants of self-stabilization that naturally arise in the Hopfield network setting (e.g., k-bounded self-stabilization, k-bounded self-stabilization on average, asymptotic 1-bounded self-stabilization) are of interest in the distributed systems setting. We suggest that the notion of attraction domains, which plays an important role in Hopfield networks (e.g., in associative memory applications), may be found useful also in the distributed systems setting. Perhaps the associative memory and optimization applications of self-stabilizing neural networks can be extended to the more general setting of self-stabilizing distributed systems. For some preliminary work in this direction, in the context of optimization, see [CDK], which shows that a certain kind of self-stabilizing distributed constraint satisfaction is impossible in a Hopfield-type neural network model but is achievable in a more sophisticated self-stabilizing distributed system.

It is useful to note that the Hopfield network is the best candidate in the field of neural networks for the study of self-stabilization issues. It is second only to the backpropagation network in its significance within the field. The self-stabilization property applies neither to the backpropagation network nor to virtually any other network except the Hopfield network. Indeed the Hopfield network was designed, in effect, with the self-stabilization property in mind.

We close this paper with a few suggestions for further work at the intersection of the two fields.

- Proposition 1 establishes only probabilistic convergence in the asymptotic limit. The proof employs a loose argument. Perhaps the result can be strengthened to establish rapid probabilistic convergence.

- It remains necessary to obtain impossibility results for certain problems on the Hopfield model. Some preliminary work along these lines is in [CDK].

- Additional research may extend the associative memory and combinatorial optimization applications of the Hopfield model to the more general setting of self-stabilizing distributing systems.

Acknowledgments

The author thanks Rina Dechter for introducing him to this topic and Ajoy Datta for reviving his interest in it. The author thanks an anonymous reviewer for helpful comments, especially for suggesting self-stabilization analysis under a fine-grained implementation, and another anonymous reviewer for pointing out that not every fair deterministic schedule preserves convergence to the network's fixed-point property (Section 3) under the fine-grained implementation (Section 6).

References

[AK89] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* Wiley, New York, 1989.

[Ama89] S. Amari. Characteristics of sparsely encoded associative memory. *Neural Networks*, 2(6):451–457, 1989.

[AMJ85] Y. S. Abu-Mostafa and J. S. Jacques. Information capacity of the Hopfield model. *IEEE Transactions on Information Theory*, 31(4):461–464, July 1985.

[CDK] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. Unpublished manuscript.

[Dij74] E. W. Dijkstra. Self-stabilization in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[Flo92] P. Floreen. *Computational complexity problems in neural associative memories.* PhD thesis, University of Helsinki, Finland, 1992.

[FSGW83] F. Fogelman-Soulie, E. Goles, and G. Weisbuch. Transient length in sequential iteration of threshold functions. *Discrete Applied Mathematics*, 6:95–98, 1983.

[GE88] A. Goshtasby and R. W. Ehrich. Contextual word recognition using probabilistic relaxation labeling. *Pattern Recognition*, 21(5):455–462, 1988.

[GFSP85] E. Goles, F. Fogelman-Soulie, and D. Pellegrin. Decreasing energy functions as a tool for studying threshold networks. *Discrete Applied Mathematics*, 12:261–277, 1985.

[GG] S. Ghosh and A. Gupta. An exercise in fault-containment: Leader election on a ring. *Information Processing Letters.* To appear.

[GGHP96] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 45–54. ACM, Los Alamitos, CA, 1996.

[GGP96] S. Ghosh, A. Gupta, and S. V. Pemmaraju. A fault-containing self-stabilizing spanning tree algorithm. *Journal of Computing and Information*, 2(1):322–338, 1996.

[GGP97] S. Ghosh, A. Gupta, and S. V. Pemmaraju. Fault-containing network protocols. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, pages 431–437. ACM, Los Alamitos, CA, 1997.

[GHR90] M. Gouda, R. Howell, and L. Rosier. The instability of self-stabilization. *Acta Informatica*, 27:697–724, 1990.

[Gro96] T. Grossman. Applying the INN model to the MaxClique problem. In D. S. Johnson and M. A. Trick, editors, *DIMACS Series: Second DIMACS Challenge*, pages 125–145. American Mathematical Society, Providence, RI : Baltimore, MD, 1996.

[HKP91] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Reading, MA, 1991.

[HM93] A. V. M. Herz and C. M. Marcus. Distributed dynamics in neural networks. *Physical Review E*, 47(3):2155–2161, 1993.

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

[Hop84] J. J. Hopfield. Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81:3088–3092, 1984.

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

[Jag96] A. Jagota. An adaptive, multiple restarts neural network algorithm for graph coloring. *European Journal of Operational Research*, 93:257–270, 1996.

[JG] A. Jagota and M. Garzon. On the mappings of optimization problems to neural networks. In *World Congress on Neural Networks*, volume 2, pages 391–398. IEEE, New York.

[JSG96] A. Jagota, L. Sanchis, and R. Ganesan. Approximating maximum clique using neural network and related heuristics. In D. S. Johnson and M. A. Trick, editors, *DIMACS Series: Second DIMACS Challenge*, pages 169–204. American Mathematical Society, Providence, RI : Baltimore, MD, 1996.

[KA89] J. H. M. Korst and E. H. L. Aarts. Combinatorial optimization on a Boltzmann machine. *Journal of Parallel and Distributed Computing*, 6:331–357, 1989.

[KH90] Y. Kamp and M. Hasler. *Recursive Neural Networks for Associative Memory.* Wiley, New York, 1990.

[LWB94] Q. Li, X. Wang, and E. K. Blum. Asynchronous dynamics of continuous-time neural networks. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 493–500. Morgan Kaufman, San Mateo, CA, 1994.

[MPRV87] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461–482, 1987.

[MW89] C. M. Marcus and R. M. Westervelt. Dynamics of iterated-map neural networks. *Physical Review A*, 40(1):501–504, 1989.

[PA88] C. Peterson and J. R. Anderson. Neural networks and NP-complete optimization problems: A performance study on the graph bisection problem. *Complex Systems*, 2(1):59–89, 1988.

[Par94] I. Parberry. *Circuit Complexity and Neural Networks.* MIT Press, Cambridge, 1994.

[Sch93] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[WJBG96] X. Wang, A. Jagota, F. Botelho, and M. Garzon. Absence of cycles in symmetric neural networks. In D. S. Touretzky, M. Mozer, and M. Hasselmo, editors, *Proceedings of the Ninth Neural Information Processing Systems Conference*, pages 372–378. MIT Press, Cambridge, 1996.

[WP88] G. V. Wilson and G. S. Pawley. On the stability of the travelling salesman problem algorithm of Hopfield and tank. *Biological Cybernetics*, 58:63–70, 1988.