N90-29078

# LEARNING IN STOCHASTIC NEURAL NETWORKS FOR CONSTRAINT SATISFACTION PROBLEMS

**Mark D. Johnston**
*Space Telescope Science Institute*[1]
*3700 San Martin Drive, Baltimore, MD 21218 USA*

**Hans-Martin Adorf**
*Space Telescope – European Coordinating Facility*
*European Southern Observatory*
*Karl-Schwarzschild-Str. 2, D-8046, Garching bei München, F.R. Germany*

**Abstract**

We describe a newly-developed "artificial neural network" algorithm for solving constraint satisfaction problems (CSPs) which includes a learning component that can significantly improve the performance of the network from run to run. The network, referred to as the Guarded Discrete Stochastic (GDS) network, is based on the discrete Hopfield network but differs from it primarily in that auxiliary networks (guards) are asymmetrically coupled to the main network to enforce certain types of constraints. Although the presence of asymmetric connections implies that the network may not converge, we find that, for certain classes of problems, the network often quickly converges to find satisficing solutions when they exist. The network can run efficiently on serial machines and can find solutions to very large problems (e.g. $N$-queens for $N$ as large as 1024). One advantage of the network architecture is that network connection strengths need not be instantiated when the network is established: they are needed only when a participating neural element transitions from off to on. We have exploited this feature to devise a learning algorithm, based on consistency techniques for discrete CSPs, that updates the network biases and connection strengths and thus improves the network performance.

## 1 Introduction

Constraint satisfaction problems (CSPs) arise frequently in AI applications and have been investigated by many researchers. Most of the commonly used methods for finding solutions to CSPs are based on backtracking tree search or its variants. A variety of techniques have been utilized to make this type of search more efficient: these include pre-processing the constraints, ordering the instantiation of variables, or making intelligent decisions about how to backtrack when a deadend is encountered (see, e.g., [1,2,3,4]).

A very different approach has been taken by researchers investigating "artificial neural network" or "connectionist" approaches to solving CSPs (e.g. [5,6]). In this type of approach the constraints are encoded in the

[1] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

network topology and connection strengths so that the state of the network when it converges can be interpreted as a solution to the CSP. The network dynamics can be described in terms of an "energy" function which the network minimizes as it runs [7]. A problem with these methods is the tendency of the network to settle into a local minimum of the energy function, representing a solution only to a sub-problem of the CSP. Techniques for escaping from local minima are known [8,9] but tend to be time-consuming and thus greatly limit the size of the problem that can be represented and solved. We have previously described a new network architecture which circumvents some of these problems [10]. Our approach, which we call the Guarded Discrete Stochastic (GDS) network, avoids local minima by coupling the main network to one or more fast-acting auxiliary (guard) networks that enforce additional higher-order constraints. While this has the drawback that the network is no longer guaranteed to converge to any stable configuration, we find that for a variety of problems the network has a high probability of converging with sufficient speed that solutions to very large problems can be found even on serial machines.

In the GDS network, as in other neural network approaches to CSPs, the problem is explicitly encoded in the network when it is constructed. This is in contrast to the use of neural networks on other types of problems where the network goes through a training phase to "learn" the values of the connection strengths and biases that are appropriate to the problem [9,11]. One advantage of the GDS network architecture and update scheme is that the the connections can be treated as "virtual", i.e. the values of the connection strengths are not needed until a participating neuron transitions from off to on. We have found that this can be used as the basis for a learning algorithm that infers additional constraints only from instantiated connections. This can be viewed as the network analog of "learning while searching" as successfully applied to backtracking tree search [12].

In the following (Section 2) we first briefly describe the GDS network architecture and update scheme from [10]. We then describe the learning algorithm and present results for two CSPs that show how learning can significantly improve the network's performance (Section 3). We conclude with a general discussion of the network's behavior during search and why the learning algorithm is effective (Section 4).

# 2 The Guarded Discrete Stochastic (GDS) Network

The problem we consider is a general binary CSP involving a set of $N$ variables $X_1, \ldots, X_N$ with domains $D_1, \ldots, D_N$, and an associated set a set of constraints $C_\alpha(X_j, X_k), \alpha = 1, \ldots, M$. A binary constraint is a subset of the Cartesian product $D_j \times D_k$ which specifies combinations of values which are incompatible with each other. A solution is an assignment of values to all of the variables so that no constraints are violated. We are interested here in the problem of finding at least one satisfying assignment (the satisficing problem).

We first consider how to represent this CSP by a Hopfield discrete neural network [7] of which the GDS network is a generalization. Let the output (zero or one) of the neuron labeled $ij$ be denoted by $y_{ij}$, where $i$ refers to the $i^{th}$ variable $X_i$ and $j$ refers to $d_{i,j}$, the $j^{th}$ value in the domain $D_i$. When viewed as a matrix (with a variable column width depending on cardinality of the domains $D_i$), rows are associated with variables and columns are associated with values.

The assignment of $d_{i,j}$ to $X_i$ is represented by $y_{ij} = 1$. The input $x_{ij}$ to neuron $ij$ is the sum of a bias term $b_{ij}$ and a weighted sum of the output of other neurons:

$$x_{ij} = \sum_{mn} W_{ij,mn}\, y_{mn} + b_{ij} \tag{1}$$

$W_{ij,mn}$ is called the connection matrix. In the two-state neuron model the output is related to the input by:

$$y_{ij} = \begin{cases} 1 & x_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases} . \tag{2}$$

In the discrete Hopfield model with no transmission delays, neurons are selected at random and their output is set according to Eqn. (2). When the connections are symmetric ($W_{ij,mn} = W_{mn,ij}$) and there is no self-feedback ($W_{ij,ij} = 0$), then there exists a bounded "energy" function which the network minimizes as it runs. The biases

$b_{ij}$ and connection weights $W_{ij,mn}$ can be chosen so that a solution to the CSP is a minimum of this energy function as follows (see Fig. 1):

$$b_{ij} = \beta \tag{3}$$

$$W_{ij,mn} = \begin{cases} -\omega & \text{if } (d_{i,j}, d_{m,n}) \in C_a(X_i, X_m) \\ -\eta & \text{if } i = m, j \neq n \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $\beta$ and $\omega, \eta > \beta$ are positive constants. The first set of terms in Eqn. (4) implement the constraints $C_a$, i.e. if a pair of assignments is forbidden by any constraint, then there is an inhibitory link between the corresponding neurons. The second set of terms represents the condition that at most one value can be assigned to each variable.

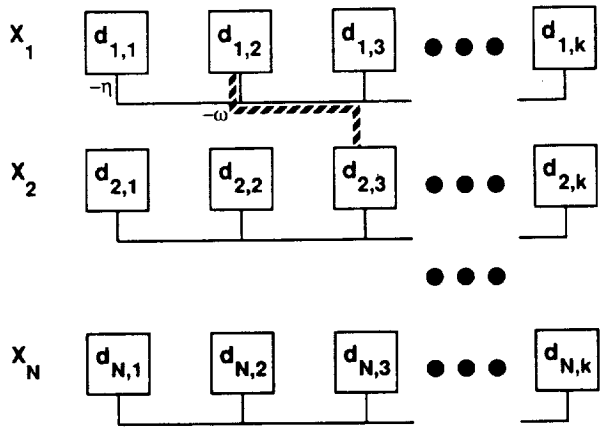


Figure 1. A Hopfield network for a binary CSP: variables are represented by rows, value assignments by neurons on each row (labelled by the domain value they represent). Here it is assumed that each variable $X_i$ can assume one of $k$ values. The network includes a set of symmetric inhibitory links that permit only one value to be assigned to each variable (solid lines) and another set that represents the binary constraints (one example is shown as a heavy dashed line).

If the network update algorithm Eqn. (2) is applied to this problem it is quickly found that, while the network sometimes converges to an assignment for all $N$ variables, it frequently comes to rest in a stable state with $n < N$ neurons active: these are local minima of the energy function. The GDS network introduces a way to escape local minima that is especially well-suited for discrete networks: the asymmetric coupling of the main network to an auxiliary network (Fig. 2). The auxiliary network, which we call a guard network, is designed to enforce an additional important condition of the problem, namely that when a solution is found, every variable must have an assigned value. When this condition is enforced, states with $n < N$ neurons active are no longer stable, and so the network continues to evolve.

The guard network consists of an additional $N$ neurons, one for each variable which must have an assigned value. A guard neuron with bias $b_i^g$, input $x_i^g$, and output $y_i^g$ is connected to each neuron on the row $i$ that it guards. The input to the guard is $x_i^g = -\theta \sum_j y_{ij}$, while the contribution by the guard to the input of neuron $ij$ is $\phi\, y_i^g$. If we choose the guard bias to be $b_i^g = \gamma > 0$ and choose $\theta > \gamma$ and $\phi > 0$ sufficiently large, then the guard on row $i$ will fire only when *no* neurons on row $i$ are firing. When the guard fires, a large positive value $\phi$ is added to the input of each neuron on the row: if $\phi$ is chosen to be large enough to overcome the

effect of any number of inhibitory links, then any neurons on the row can transition from off to on, thereby reducing the energy of the network. Thus local minima due to the absence of any firing neurons on a row are eliminated. The price paid for this desirable feature is that the symmetry of the connection matrix for the combined network is lost, and thus convergence to a stable state is no longer guaranteed. In practice some stopping criterion must be specified, which may be problem-dependent.

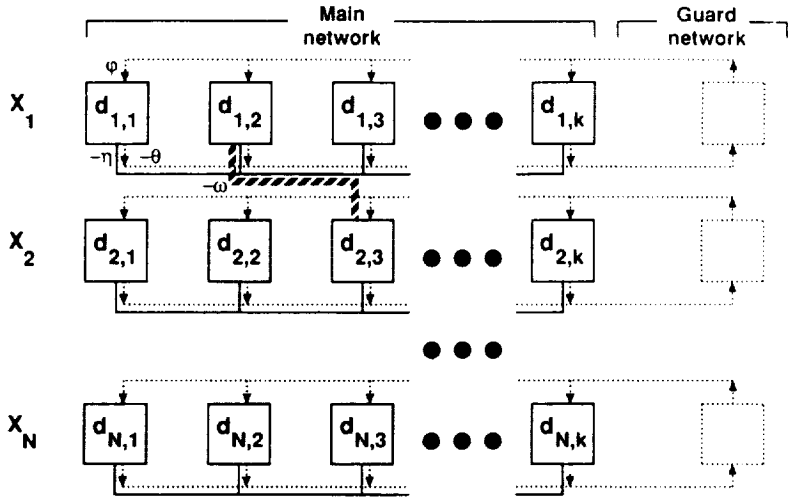


Figure 2. The GDS network: the network of Fig. 1 is coupled asymmetrically to a guard network to enforce the condition that each variable must have an assigned value (dotted lines).

We have found that it is most effective to update the guard network synchronously with transitions on the main board, i.e. each guard's output is always maintained consistent with its input according to Eqn. (2). This essentially treats the guards as a separate network which runs on a faster timescale than the main network. We have also found that random selection of which neuron in the main network to examine next is much less effective than selecting at random one *set* of neurons which are monitored by one guard neuron, then changing the state of the neuron in the set whose output is "most inconsistent" with its input (if any). That is, we select the neuron with the maximum value of either

$$x_{ij} \text{ if } y_{ij} = 0 \text{ and } x_{ij} \geq 0, \text{ or } |x_{ij}| \text{ if } y_{ij} = 1 \text{ and } x_{ij} < 0, \tag{5}$$

with ties broken arbitrarily.

The initial state of the network is an important consideration. If for a particular CSP there is some heuristic which can identify variable assignments which are "likely" to be part of a solution, then these can be used to specify the initial network state. If, as is often the case, no such assignments are known, then it is appropriate to start the network with all neurons in the off state $y_{ij} = 0$. In either case, the initial state will usually have nearly all neurons in the off state. This leads to the observation that the connections $W_{ij,mn}$ need not be pre-computed and stored, but may be calculated only when neuron $mn$ first transitions from off to on. (If the connections can be computed efficiently enough then it may not even be effective to store them at all). This can permit a large reduction in storage requirements: even though the number of possible connections may be large, only a small fraction may be instantiated during any given set of runs of the network.

An example of the GDS network's performance is provided by the well-studied $N$-queens problem of placing $N$ queens on an $N \times N$ chessboard, one on each row, so that no queen threatens another. This can be represented as a binary CSP with $N$ variables representing the chessboard rows and $N$ values representing the columns in

which the queens are placed. The connections $W_{ij,mn}$ encode the constraints that no two queens can threaten each other along columns or diagonals. Row threats are automatically disallowed since variables can only have one assigned value.

$N$-queens has been used as a model problem in several studies of improvements to regular backtracking search: see especially Stone and Stone [13] who conducted an investigation of backtracking and most-constrained search for $N$ up to 96. They suggest that backtracking has exponential, and most-constrained search has polynomial time complexity over the range of $N$ they studied, but they note that they were unable to find solutions in a "reasonable amount of time" for $N = 97$. A continuous neural network representation of the 8-queens problem was investigated in [5].

Solutions to the $N$-queens problem are easily found by the GDS network. In Fig. 3 is plotted the median number of neuron transitions ($0 \rightarrow 1$ and $1 \rightarrow 0$) required for the network to converge to a solution (estimated from a large number of runs) versus linear board-size $N$. The result is linear in $N$ for large $N$ as shown by the straight line fit. Note that a minimum of $N$ $0 \rightarrow 1$ transitions is required to proceed directly from the "empty board" initial state ($y_{ij} = 0$ for all $ij$) to a solution with no "wandering". The surprising result is that only a proportionately small number of *excess* transitions beyond this minimum is required to find solutions: empirically this excess is found to be about $0.16N$. To check that this behavior holds for very large $N$ we have run the network with $N$ as large as 1024. This corresponds to a main network containing $N^2 \approx 10^6$ neurons, with $> 10^9$ potentially non-zero connections. A solution to the $N = 1024$ problem was found to require only 1196 transitions and required a wall-clock time of less than 12 minutes on a 16Mb TI Explorer II workstation.

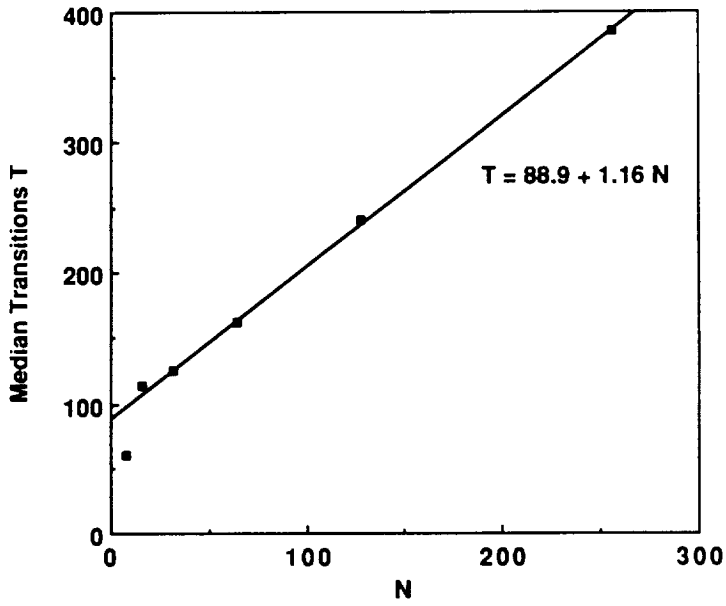


Figure 3. N-queens: Median number of transitions to convergence vs. linear board-size $N$.

The expected time complexity of the GDS network on the $N$-queens problem is $O(N^2)$, since the expected number of transitions to convergence is (empirically) $O(N)$ and each transition requires adding a connection weight to the inputs of $O(N)$ inhibited neurons (and the overhead associated with each transition is also $O(N)$). The space complexity of the network is $O(N^2)$, even though the number of non-zero connections is $O(N^3)$. Further results of the GDS network on $N$-queens and other CSPs is provided in [10].

# 3 The GDS Learning Algorithm

It has long been known that pre-processing constraints in CSPs can lead to dramatic improvements in the effectiveness of backtracking search [1,2]. These techniques, known as consistency methods, are based on the deduction of additional constraints from those explicitly provided. These additional constraints can be exploited in backtracking search to avoid repetitively exploring sets of assignments that cannot be part of any solution. While these techniques have generally been applied before search begins, Dechter [12] has shown how they can be applied during the search process to provide a kind of "learning while searching". An analogous learning process can be defined for the GDS network by exploiting the fact that network connections need not be instantiated until they are needed, i.e. when a neuron participating in a constraint transitions from off to on. Learning can be based on instantiated connections only, leading to changes in the network biases and connection strengths that improve the performance of the network from one run to the next.

The GDS learning algorithm we have developed is independent of the problem represented by the network. It operates as a separate module which analyzes the results of one or more "training" runs to update the network bias values, connection strengths, or both. Training consists of the following series of steps which can be repeated as often as desired:

1. Starting with all neurons off ($y_{ij} = 0$ for all $ij$), run the network for a fixed number of transitions $T_{train}$ and record the connections for each neuron $ij$ which transitions from 0 to 1. Denote the set of all neurons which have transitioned from 0 to 1, since the network was initialized, as ON.
2. Reset all neurons to their off state.
3. For each neuron $ij$ in ON with $x_{ij} \geq 0$, set its state to on ($y_{ij} = 1$) and turn off all others. Update the inputs of any other neurons $mn$ based on the recorded connections $W_{mn,ij} \neq 0$. If $mn$ is in ON and $x_{mn} \geq 0$ and $x_{mp} < 0$ for $p \neq n$, then set $y_{mn} = 1$ and update inputs again. Repeat until no further changes occur.
    - update biases: if there is any row $m$ such that $x_{mn} < 0$ when $b_{mn} \geq 0$ for all $n$, then set the bias of $ij$ to some value $b_{ij} < -\phi$ (effectively removing $ij$ from the network, i.e. deleting $d_{ij}$ from $D_i$).
    - update connections: if there is any $mn$ such that $x_{mn} < 0$ when $b_{mn} \geq 0$, then record the connection coefficient $W_{mn,ij} = -\omega$. This represents an induced constraint between $X_i$ and $X_m$ indicating that $d_{ij}$ and $d_{mn}$ are incompatible assignments and cannot be part of any solution.

Updating only the biases corresponds to a partial arc-consistency algorithm where only instantiated connections are considered. Updating the connection weights corresponds to partial path-consistency, i.e. the recording of additional induced constraints. These two update schemes correspond in Dechter's nomenclature to "first-order" and "second-order" learning, respectively. Note that updating the connections as described above does not correspond to full path-consistency even on the set of instantiated connections, since additional constraints could possibly be induced by those discovered during a training step. Thus the computational effort expended in training is much less than that required to perform full arc- or path-consistency [14,15].

We have compared the results of applying this learning algorithm to the results obtained from running the network on only those constraints provided explicitly in the definition of the problem. Two CSPs have been used in this investigation: the random CSP used by Dechter and Pearl in their study of Advised Backtracking [4], and the Zebra problem used by Dechter in her investigation of learning in backtracking search [12].

## 3.1 The Dechter-Pearl Problem

This problem is one of a family of random CSPs [16] specified by four parameters: the number of variables $N$, the number of values $k$ each variable can assume, the probability $p_1$ of having a constraint between any pair of variables, and the probability $p_2$ that a constraint allows a given pair of values. The behavior of the GDS

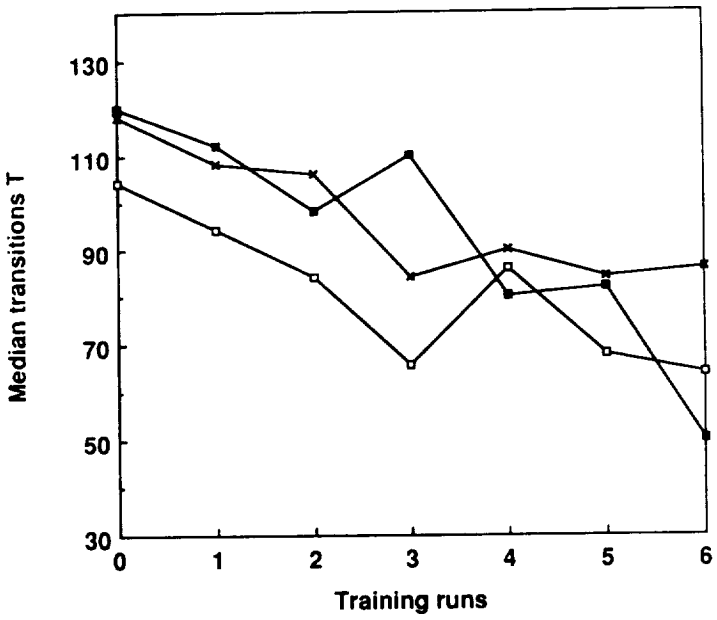


Figure 4. Dechter-Pearl CSP: median number of transitions to convergence vs. number of first-order training runs for three randomly-generated problem instances (filled squares, open squares, and crosses).

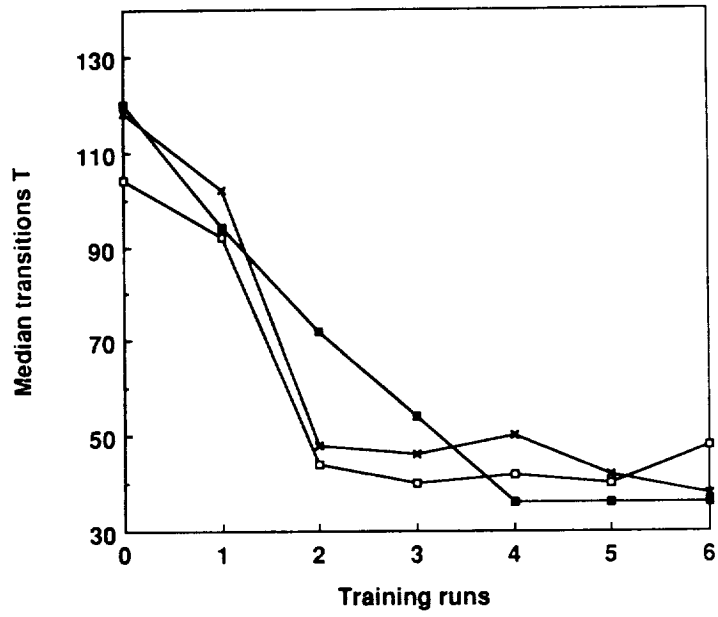


Figure 5. Dechter-Pearl CSP: median number of transitions to convergence vs. number of second-order training runs for the same problem instances as Fig. 4.

network on this problem for $k = 5$, $p_1 = 0.5$, and $p_2 = 0.6$ was reported in [10] for a range of $N$ between 30 and 120. The median number of transitions $T$ required for the network to converge was found to be linear in $N$: $T \cong 35 + 2.5N$.

Here we consider the case $N = 30$ and investigate the effectiveness of the bias and connection learning algorithms on the performance of the network. Three randomly-generated problem instances were generated and subjected

to a variable number of training sessions ranging from one to six. One set of runs consisted of bias updates only (first-order learning); the other consisted of both bias and connection updates (second-order learning). Each training run was arbitrarily limited to $T_{train} = N$ transitions. Each series of training runs was started from the explicit constraints only, i.e. there is no correlation of the results as the number of training runs increases.

The results of first-order learning are plotted in Fig. 4 which shows the median number of transitions required for the network to converge to a solution vs. the number of training runs. Note that a minimum of $N = 30$ transitions is required to proceed directly from the initial network state $y_{ij} = 0$ to a solution. It can be seen that there is an approximately steady decrease in the median number of transitions, from about 115 with no training to an average of about 65 with six training runs.

Second-order learning (Fig. 5) shows a more significant performance improvement with the first few training runs, but little further improvement with additional training. After only three training steps the median number of transitions has decreased from 115 to an average of about 45.

## 3.2 The Zebra Problem

This significantly harder problem was described by Dechter (see Appendix II of [12]) and was used in her study of learning during backtracking search. The problem consists of $N = 25$ variables, each with 5 possible values. The GDS network without learning converges to a solution only about 10% of the time when limited (arbitrarily) to $9N$ transitions. Although first-order learning makes only a marginal difference in the performance of the network, second-order learning shows a dramatic improvement. Fig. 6 shows the probability of convergence in

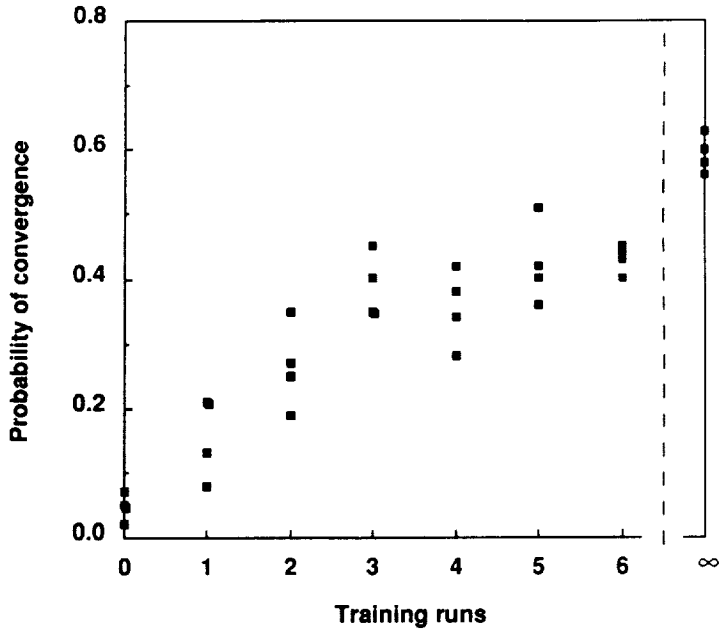


Figure 6. The Zebra Problem: probability of convergence in $9N$ transitions vs. number of training runs, with "∞" representing a fully path-consistent version of the problem.

$9N$ transitions vs. number of second-order training runs. The results for "∞" are for a fully path-consistent version of the problem and represents the best that can be achieved by increasing the amount of training. Even a small number of training runs can clearly improve the network performance by a significant margin.

# 4 Discussion

The behavior of the GDS network can be likened to a stochastic backtracking algorithm which implements a number of "heuristics" to expedite search. Stochastic, in contrast to regular backtracking, means that the order of instantiation of variables is not pre-determined: backtracking makes a systematic exploration of the search tree, while the network stochastically probes the tree in directions that tend to minimize the network energy. Since the network permits temporary inconsistencies in variable assignments at any point until it converges, it can make "lateral jumps" in the search tree to escape from sets of assignments that cannot be consistently extended. These jumps appear to be useful in discovering consistent assignments, although their effectiveness depends on the detailed structure of the search tree (as evidenced by the results on 3-Colorability in [10]).

The heuristics intrinsic to the network come into play when a partial instantiation cannot be consistently extended. This corresponds to encountering a deadend during backtracking search. These network heuristics cannot be strictly isolated (since extending partial assignments and backing out of deadends are simultaneous competing processes), but they can be loosely compared to those developed to improve the behavior of backtracking algorithms:

- backjumping: when the network encounters a deadend it will randomly select an uninstantiated variable and assign it a value which is certain to be inconsistent with one or more previously made assignments. This entire set of inconsistent assignments is at once subject to revision: at least one will eventually be retracted. This corresponds closely to the backjumping or "go back to cause of failure" heuristic which is known to improve the performance of regular backtracking, but is somewhat more general in that any variable with no permitted assignments can be considered the "failure", and any variable with which it is inconsistent can be considered the "cause".
- value selection: when the network extends a partial instantiation by assigning a value to an unassigned variable, any value not forbidden by some constraint is equally likely to be chosen. However, at a deadend, values are selected which are least inhibited by any current assignments (since the neuron input is proportional to the number of constraints that forbid the assignment). This represents a kind of value selection heuristic which undoes a *minimal* set of previous assignments in order to escape from the deadend. Only value assignments that participate in such minimal sets will be made by the network update algorithm Eqn. (5).

The GDS network is a general constraint satisfaction search method, encoding no domain knowledge other than value and value-pair inhibitions. Nevertheless the convergence of the network on some classes of problems is remarkably fast. This, along with the ease with which the network can be set up for new problems, makes it an attractive approach for some classes of large CSPs. We have shown here that the performance of the network can be significantly improved by adding a "learning" module that analyzes the results of one or more training runs and updates the initial values of the neuron biases and connection strengths. The learning algorithm is independent of the problem represented by the network. In terms of the network heuristics discussed above, the effectiveness of this type of learning is due to the resulting increase in the sizes of minimal sets of inconsistent variables. As a result, inconsistent sets are encountered after fewer transitions, and longer and more relevant "jumps" in the search space are made possible. For some types of problems this dramatically improves the speed of convergence of the network.

# References


[1] Montanari, U.: 1974, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing", *Information Sciences* **7**, 95–132

[2] Mackworth, A.K.: 1977, "Consistency in Networks of Relations", *Artif. Intell.* **8**, 99–118

[3] Freuder, E.C.: 1982, "A Sufficient Condition of Backtrack-free Search", *J. ACM* **29**, 24–32

[4] Dechter, R., Pearl, J.: 1988, "Network-based Heuristics for Constraint-Satisfaction Problems", *Artif. Intell.* **34**, 1–38

[5] Tagliarini, G.A., Page, E.W.: 1987 "Solving Constraint Satisfaction Problems with Neural Networks", *Proc. IEEE First Internat. Conf. on Neural Networks, San Diego* **3**, 741–747

[6] Dahl, E.D.: 1987, "Neural Network Algorithm for an NP-Complete Problem: Map and Graph Coloring", *Proc. IEEE First Internat. Conf. on Neural Networks, San Diego* **3**, 113–120

[7] Hopfield, J.J.: 1982, "Neural networks and physical systems with emergent collective computational abilities", *Proc. Nat. Acad. Sci.* **79**, 2554–2558

[8] Kirkpatrick, S., Gelatt, C., Vecchi, M.: 1983, "Optimization by simulated annealing", *Science* **22**, 671–680

[9] Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: 1985, "A Learning Algorithm for Boltzmann Machines", *Cognitive Science* **9**, 147–169

[10] Adorf, H.-M., and Johnston, M.D.: 1988, "A Discrete Stochastic "Neural Network" Algorithm for Constraint Satisfaction Problems", submitted.

[11] Hinton, G.E.: 1987, "Connectionist Learning Procedures", CMU Tech. Report CMU-CS-87-115 (Ver. 2), Department of Computer Science, December 1987.

[12] Dechter, R.: 1986, "Learning While Searching in Constraint Satisfaction Problems", *Proc. AAAI-86, Fifth Nat. Conf. Artif. Intell., Philadelphia* **1**, 178–183

[13] Stone, H.A., Stone, J.M.: 1987, "Efficient search techniques — An empirical study of the $N$-Queens Problem", *IBM J. Res. Devel.* **31**, 464–474

[14] Mackworth, A.K., Freuder E.C.: 1985, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", *Artif. Intell.* **25**, 65–74

[15] Mohr, R., Henderson, T.C.: 1986, "Arc and Path Consistency Revisited", *Artif. Intell.* **28**, 225–233

[16] Haralick, R.M., Elliott, G.L.: 1980, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", *Artif. Intell.* **14**, 263–313