# SOLVING CONSTRAINT SATISFACTION PROBLEMS USING NEURAL NETWORKS

*Dr. C. J. Wang Dr. E. P. K. Tsang*

Department of Computer Science, University of Essex Wivenhoe Park

Colchester CO4 3SQ

## INTRODUCTION

A constraint satisfaction problem (CSP) is defined as a triple *(Z,D,C)*, where *Z* is a finite set of variables, *D* is the set of domains for the variables, and *C* is a set of constraints. Each constraint in *C* restricts the values that one can assign to a set of variables simultaneously. A constraint is *n*-ary if it applies to *n* variables. The task is to assign one value per variable, satisfying all the constraints in *C* [1]. A *binary constraint* CSP is a CSP with unary and binary constraints only.

Problems in many application domains can be formulated as CSPs, for instance, the N-queens problem [2], line labelling in vision [3], temporal reasoning [4, 5], and scheduling [6], to name a few. The majority of existing work in CSP focuses on problem reduction and heuristic search [1, 2]. The search space in a CSP is $O(d^N)$, where *d* is the domain size (assuming, for simplicity, that all domains have the same size) and *N* is the number of variables in the problem. We define the *tightness* of a CSP to be the number of solutions over the search space.

Constraint programming languages based on heuristic search, such as CHIP [7], have claimed to be efficient in a number of applications [6]. For handling variables with finite domains, these languages use the Forward Checking algorithm coupled with the Fail First Principle (we shall refer to this later as the FC-FFP approach) [2]. One problem of such heuristic search approaches is that they have exponential complexity. For CSPs with, say, 10000 variables and an average domain size of 50, the search space will be so huge that even a heuristic search approach may not produce any answer within a tolerable period of time. Thus, many real life CSPs are computationally intractable by heuristic search.

In principle, neural networks may be able to overcome the above problems. Whilst a heuristic search approach guarantees to find a valid solution if there is any, the probabilistic nature of the neural network convergence procedure may produce a solution much more quickly. Hopfield and Tank's work on the Travelling Salesman Problem [8] is such an example. The attempt to apply neural network techniques to solve CSPs has led to the discovery of an algorithm called *Heuristic Repair Method* [9], which uses the so called *Min-conflict Heuristic*. The Heuristic Repair Method is extracted from the GDS neural network model [10], and it manages to solve the million-queens problem in minutes.

A CSP can be represented as a network structure in which the variable assignments are represented by the activation of nodes and the constraints are represented by connections, possibly with different weights. Hopefully, when the network converges, the set of nodes which are on represent a set of assignments which form a solution. One problem with applying neural network techniques for solving CSP is that the network may settle in local minima -- i.e. a set of assignments which violates a small number of constraints but it does not represent a solution of the problem. The Heuristic Repair Method has only shown its effectiveness in binary CSPs where many solutions exist. In the N-queens problem, the larger N is the more the solutions there would be. Our experiments have shown that the Heuristic Repair Method will fail to solve CSPs which have few solutions or the problems for which there are many local minima. This will be discussed in detail later.

In this paper, we describe GENET, a generic neural network simulator, that can solve general CSPs with finite domains. GENET generates a sparsely connected network for a given CSP with constraints C specified as binary matrices, and simulates the network convergence procedure. In case the network falls into local minima, a heuristic learning rule will be applied to escape from them. The network model lends itself to massively parallel processing. The experimental results of applying GENET to randomly generated, including very tight constrained, CSPs and the real life problem of car sequencing will be reported and an analysis of the effectiveness of GENET will be given.

## NETWORK MODEL

The network model is based on the Interactive Activation model (IA) with modifications to suit the natures of the CSPs as defined at the beginning of this paper. The IA model in its original form can be characterized as *weak constraint satisfaction*, in which the connections represent the coherence, or compatibility, between the connected nodes. This model was developed for associative information retrieval or pattern matching [11, 12]. However, it is not adequate for solving CSPs in general, for which all the constraints are absolute and none of them should be violated at all. For this purpose, the following modifications have been developed.

1. The nodes in the network are grouped into clusters with each cluster representing a variable in *Z*, and the nodes in each cluster represent the values that can be assigned to the variable.
2. Only inhibitory connections are allowed. The inhibitory connections represent the constraints that do not allow the connected nodes to be active (i.e. turned on) simultaneously.
3. The nodes in the same cluster compete with each other in convergence cycles. The node that receives the maximum input will be turned on and the others turned off. This is to ensure that only one value is

assigned to a variable at any time and that this value violates a minimum number of constraints.

By convention, the state of node *i* is denoted as $s_i$, which is either *1* for active or *0* for inactive. The connection weight between nodes *i* and *j* is denoted as $w_{ij}$, which is always a negative integer and initially given the value *-1*. The input to a node is the weighted sum of all the nodes' states connected to it. To illustrate how a network can be constructed for a CSP, let's take a simple example as follows.

EXAMPLE: A TIGHT BINARY CSP

Suppose there are five variables, $Z_1$ to $Z_5$, and all the variables have the domain $d = \{1, 2, 3\}$. Let $V_i$ denote the value taken by the variable $Z_i$. The binary constraints between $V_i$ and $V_{i+1}$, for $i=1$ to *4*, is that the sum of $V_i$ and $V_{i+1}$ must be even. The binary constraint on $V_1$ and $V_5$ together is that: $V_1 = 2$ *OR* $V_5 = 2$

The network will be constructed as in Figure 1, where nodes in column *i* form a cluster representing the variable $Z_i$, nodes in row *j* represent the *j*th value that can be assigned to a variable, and all the connections shown are negative.

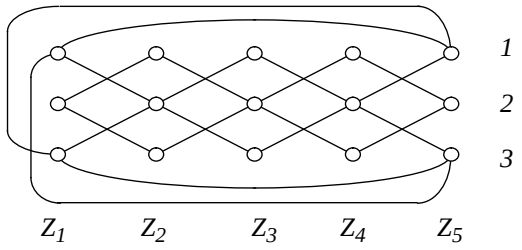


Figure 1. The network structure for the Example

For non-binary CSPs in which the values are not simple ground terms and constraints are imposed on the attributes of the value structures, a multilayer network structure will be required. The number of layers required depends on the complexity of the value structure. We will show how this can be realized later when solving the car sequencing problem.

NETWORK CONVERGENCE

Initially, one node in each cluster is randomly selected, which means randomly assigning a value to each variable. Then, in each convergence cycle, every node calculates its input and the node in each cluster that has the maximum input will be selected to turn on and the others will be turned off. Since there exist only negative connections (representing the constraints in the problem), the winner in each cluster represents a value assigned to the corresponding variable which would violate the fewest constraints. This effectively resembles the Min-Conflict Heuristic [9]. After a number of cycles, the network will settle in a stable state. In a stable state, if all the active nodes have zero input, a valid solution has been found. Otherwise, the network is in a local minimum.

When updating the network state, care has to be taken in selecting the winning node in a cluster if there is more than one node that has the maximum input. In this case, if none of them is on, one will be randomly selected to turn on. If one of them is already on, it will remain on. This is to avoid chaotic or cyclic wandering of the network states. We have considered and experimented to break ties randomly, as it is done in the Heuristic Repair Method, but to find it ineffective in problems which have few solutions.

Take the above example for instance. The Heuristic Repair Method will fail in most runs. This is because only one out of the 243 possible network states represents a solution. Moreover, there are 88 local minima and about another 63 states will lead to a local minimum. Therefore, the chances are that the network will fall into local minima with a probability of 151/243, or approximately 62%. This simple example shows the inadequacy of the Heuristic Repair Method and the pitfall of using neural networks to solve CSPs in general.

ESCAPING LOCAL MINIMA

When the network settles in a local minimum, there are some active nodes that have negative input, indicating that some constraints are violated. This happens because the state update of a node is a local decision based on the principle that the activated node in a cluster should violate a minimal number of constraints. Of course, this does not necessarily lead to a globally optimal decision (one which finds a solution). In the case of local minima, the state update rule would fail to make alternative choices. It would appear that introducing randomness or noise in the state update rule, in the manner of simulated annealing as suggested in the literature [13], might help in escaping local minima. However, this will degrade the overall performance so drastically that this approach will not be effective for solving real life problems.

In order to overcome these problems, we propose a learning rule that heuristically updates the connection weights to help make alternative selections of active nodes to escape local minima. The change of weight for the connection between every pair of nodes *i* and *j*, $\Delta w_{ij}$, is defined as follows:

$$\Delta w_{ij} = - s_i \times s_j$$

To show that this heuristic learning rule is effective to escape local minima, let's consider the case in which the network is in a local minimum. Since the network is in a local minimum, there must exist at least two active nodes connected by a negative weight. Let one such pair of nodes be *i* and *j*. By stipulation, nodes *i* and *j* must have the maximum input in their own clusters. However, their inputs will be reduced by one after every learning cycle, as long as the network state does not change. Clearly, after sufficient (normally one or a few) learning cycles, either *i* or *j* will not win the competition in their own clusters. Hence, the state of the network will eventually find its way out of the local minima. This learning rule is effectively developing a weighting of the constraints that guides the network state trajectory towards solutions, if there exists any. When this learning rule is applied to solve the above example problem, the network always converges to the solution,

with an average of 23 convergence cycles over thousands of runs. As we shall report later, our extensive experiments show that this learning rule is so effective that for all the tested CSPs that are solvable, the network always finds a solution.

## THE CAR-SCHEDULING PROBLEM

The purpose of this experiment is to show how GENET can be applied to non-binary highly complex problems. For clarity, we will describe only a simplified version of our experiments. The *car sequencing problem* is a highly constrained problem appearing in GM production lines and considered intractable by earlier researchers [14]. Cars to be manufactured can be classified into different types (models), with each type requiring different options (e.g. sun roof, radio, etc.) to be installed. Production requirements specify the number of cars of each type to be manufactured. The problem is to position the cars to be manufactured on a conveyor belt (for production), satisfying a set of *capacity constraints*. The capacity constraints of a particular work area *w* limits the frequency of cars which require work to be done in *w* arriving in any sub-sequence. For example, because of limitation in work force, no more than 3 out of any 4 consecutive cars on the conveyor belt should require air-conditioning to be fitted.

For example, the production line may be able to produce three types of cars, and the cars may have up to three optional accessories such as air-conditioning, sun-roof, and stereo tape player, etc. The options for the type of cars are shown as in Table 1, where 1's mean the option is required by that type of car. Table 1 also shows capacity constraints on the work areas which install the three options. A constraint of *m/n* indicates at most *m* out of any *n* consecutive cars may have this option. This problem is complicated because not only the capacity constraints will have to be satisfied, but also the production quota will have to be met. In the above problem, for instance, the task is to schedule 10 cars of Type 1 and 20 cars each of Types 2 and 3 so that none of the three work areas (for installing the three options) is overloaded. In an industrial environment the constraints are normally high, and the options are many, and therefore the scheduling problem is highly difficult.

| | Car Types | | | capacity |
|---|---|---|---|---|
| | Type 1 | Type 2 | Type 3 | constraints |
| Option 1 | 1 | 1 | 0 | 2/3 |
| Option 2 | 1 | 0 | 1 | 3/4 |
| Option 3 | 0 | 1 | 1 | 2/3 |
| required: | 10 | 20 | 20 | |

Table 1. Options, Requirements and Constraints

## NETWORK DESIGN FOR CAR-SEQUENCING

Remember that a CSP is defined as a triple of (*Z, D, C*). In this problem, *Z* is the set of positions in a sequence of cars to be scheduled on to the production line. The domains for the variables are the car types. The constraints are the capacity constraints (which are *n*-ary constraints for capacity constraint *m/n*) and production requirements (which are *50*-ary constraints). The network is constructed as follows.

The first layer of the network consists of *N* clusters of nodes, with each cluster representing one variable, i.e. a position in the car sequence. Each cluster has three nodes representing the three possible values, i.e. the three car types. For convenience, we call the nodes in this layer *variable-node*s. If a variable represents position *j* in the car sequence, then we shall call the cluster which corresponds to this variable the *j*th cluster.

The second layer also has *N* clusters of nodes. Each cluster corresponds to one variable, and each node represents one option which might be required. For convenience, we shall call the nodes in the second layer *option-nodes*. The clusters in the second layer are assumed to be ordered in the same way as those in the first layer. The connections between the first and the second layers simply map the car types into options. For example, if the variable-node in the *i*th cluster of the first layer which represents car type 1 is on, then the nodes in the *i*th cluster of the second layer which represent options 1 and 2 will be on (because car type 1 requires options 1 and 2).

The final layer is constructed according to the capacity constraints. We shall call the nodes in the final layer *constraint-nodes*. If the capacity constraint of option *k* is $m/n$, then there is one constraint-node connected to the option-nodes of every *n* consecutive clusters. If a constraint-node X receives active inputs from more than *m* option-nodes, it is turned on, which signals the violation of this constraint. X will then transmit a negative signal to the

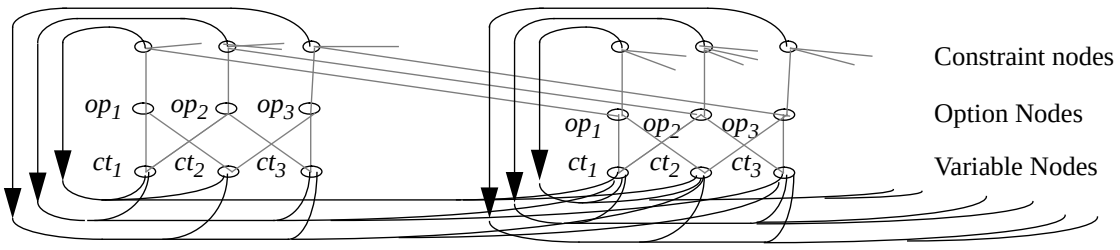


Figure 2. A template of the network for Car-Scheduling Problem, where $op_i$ stands for option *i*, $ct_i$: for car type *i*, dotted lines are mapping connections (*+1*), and solid lines are constraints (*-1*). For simplicity, the requirement-nodes are not shown.

corresponding variable-nodes to reduce their chance of being switched on.

The production requirements constraints are implemented similarly. One node, call it *requirement-node*, is used for every car type and it is connected to all the variable-nodes that represent the value of that type. If the number of cars in any type exceeds the requirement, the corresponding requirement-node will be turned on and it will send a negative signal back to those variable-nodes. Figure 2 shows a template of the network structure. Our experiments show that the network always converges to a valid solution if the problem is solvable, and heuristic learning is not even required when the number of cars is below 20.

## RANDOMLY GENERATED CSPS

In order to test the effectiveness of GENET for solving CSPs in general, we have performed thousands of experiments on different types of randomly generated CSPs which were generated by varying the following 5 parameters:

- $N$ the number of variables;
- $D$ the maximum size of domains;
- $d$ the average size of individual domains ($d \le D$);
- $p_1$ the percentage of constraints between the variables, i.e. in the generated problem, there are $p_1 \times N(N-1)/2$ constraints; and
- $p_2$ the percentage of the value compatibility between every two constrained variables, i.e. $p_2 \times d_i \times d_j$ combinations of the value assignment of the variables $V_i$ and $V_j$ are legal.

The probability of two random assignments being compatible, $p_c$, can be obtained as follows:

$$p_c = 1 - p_1 + p_1 \times p_2$$

| No. of variables | average cycles | max.cycles |
|---|---|---|
| 10 | 1.480 | 2.00 |
| 20 | 1.950 | 2.00 |
| 30 | 2.000 | 2.00 |
| 40 | 2.020 | 3.00 |
| 50 | 2.120 | 5.00 |
| 60 | 2.560 | 8.00 |
| 70 | 2.920 | 7.00 |
| 80 | 5.050 | 24.00 |
| 90 | 6.180 | 28.00 |
| 100 | 7.370 | 21.00 |
| 110 | 10.640 | 26.00 |
| 120 | 14.080 | 38.00 |
| 130 | 18.720 | 51.00 |
| 140 | 26.970 | 94.00 |
| 150 | 35.440 | 102.00 |
| 160 | 58.240 | 223.00 |
| 170 | 107.500 | 337.00 |

Table 2. Average and maximum number of cycles taken by GENET to solve randomly generated problems, with 100 runs per each case

We have tested on problems with varying number of variables, varying domain sizes, and varying degrees of tightness (by varying $p_1$ and $p_2$).

The results of GENET is checked against programs which perform complete search (using the FC-FFP approach). GENET is given a limit in the number of convergence cycles. If this limit is exceeded before a solution is found, GENET is instructed to report a failure. For all the problems tested, GENET is found to be capable of converging on solutions in solvable problems, and reporting failure in insoluble problems.

## ANALYSIS OF EFFECTIVENESS

In the simulator, the state of one node is changed at a time. However, in a hardware implementation, the nodes could change their states in parallel. The performance of the hardware implementation should be measured by the time it takes to find solutions, which can be estimated by the number of cycles in the simulator. Table 2 shows the number of cycles that it takes to solve CSPs which have $D = d = 6$, $p_1 = 10\%$ and $p_2 = 85\%$. Our test under these parameters is limited to 170 variables because the exhaustive search program fails to terminate in over 24 hour for problems with 180 variables or more. It should be mentioned that GENET terminates faster than the exhaustive search program in problems with 160 variables or more.

As can be seen from Table 2, the number of cycles taken by GENET to find solutions grows exponentially with the number of variables $N$. Statistical analysis shows that:

$$no\_of\_cycles = e^{0.026 \times N - 0.306}$$

with the correlation R = 0.975, which suggests a good fitting. This is not surprising, as the CSP is an NP-hard problem. However, we should note that the absolute number of cycles required to find a solution is bounded for the following reasons. The number of cycles that is required to find a solution is influenced by the number of times that learning takes place, which is in turn influenced by the number of local minima in the search space. Tracing in the simulator reveals that the tighter the problem is, the more times the learning takes place. The probable number of solutions in percentage of the total search space, $S_p$, can be expected to be:

$$S_p = \prod_{i=1}^{N-1} p_c^i = p_c^{\frac{N(N-1)}{2}}$$

Clearly, as $S_p$ decreases much more quickly than $d^N$ increases, the tightness of a problem grows super exponentially as $N$ grows (all other parameters being kept unchanged). When N grows to over 200, problems are normally insoluble. Therefore, the number of cycles taken by GENET is bounded.

It is important to note that the absolute number of cycles is of the order of hundreds. When $N = 200$, the expected number of cycles is 133.00. For an analog computer that takes $10^{-8}$ to $10^{-6}$ seconds to process one cycle (as a rough estimation), a problem of size $O(6^{200})$ can be solved in terms of $10^{-6}$ to $10^{-4}$ seconds.

## DISCUSSION

The number of nodes required by GENET for binary constraint problems is $N \times d$, where $N$ is the number of variables and $d$ is the domain size (assuming all domains have the same size). When $k$-ary constraints, where $k>2$, are considered, the number of nodes required is $O(N^k d)$ in the worst case.

Recently, Guesgen proposes a NN approach for solving CSPs [15]. The number of nodes required for binary constraint problems in this method is $O(N^2d^2)$, or $O(N^3d^3)$ if the operation of each node is to be simplified. When $k$-ary constraints, where $k>2$, are being considered, the number of nodes will be $O(N^k d^k)$. The set up and the operations involved in each node are significantly more complex than that in GENET.

Although the FC-FFP approach and other search algorithms can be parallelized, they cannot provide satisfactory solutions even with a polynomial number of processors [16]. The GENET approach, however, requires no more processors than the number of nodes required in the problem, as discussed above.

## SUMMARY AND FUTURE WORK

This is a report of on-going research. We have presented a general framework for applying neural network techniques to CSPs. The network model is developed from the Interactive Activation Model.

We have proposed to use structured multilayer recurrent neural networks to represent non-binary CSPs, and a learning algorithm to propagate constraints effectively through the network so as to escape local minima. To justify our approach, we have looked at, apart from a large number of randomly constructed CSPs, some specially designed problems for which the Heuristic Repair Method has failed to produce solutions, and the car sequencing problem which is highly constrained non-binary CSP. In all our tests so far, the simulator GENET has succeeded in finding solutions when one exists and reporting failure when the problem is insoluble, although the proof of completeness has not yet been theoretically developed. In any case, we argue that this approach gives hope to solving real life CSPs to the scale that would be intractable by conventional methods.

Our next task is to investigate the properties of our model more thoroughly and improve GENET's efficiency. Our long term objective is to design hardware, based on our formalism, for solving CSPs very efficiently.

## ACKNOWLEDGEMENT

The authors are grateful to Dr John Ford for his help in analysing the experimental results and Jenny Emby for her help in improving the presentation.

## REFERENCES

1. Mackworth, A.K., "Consistency in networks or relations", Artificial Intelligence 8(1), 1977, 99-118
2. Haralick, R.M. & Elliott, G.L., "Increasing tree search efficiency for constraint satisfaction problems", Artificial Intelligence 14(1980), 263-313
3. Waltz, D.L., "Understanding line drawings of scenes with shadows", in WINSTON, P.H. (ed.) The Psychology of Computer Vision, McGraw-Hill, New York, 1975, 19-91
4. Tsang, E.P.K., "The consistent labelling problem in temporal reasoning", Proc. AAAI Conference, Seattle, July, 1987, 251-255
5. Dechter, R., Meiri, I. & Pearl, J., "Temporal constraint networks", Artificial Intelligence, 49, 1991, 61-95
6. Dincbas, M., Simonis, H. & Van Hentenryck, P., "Solving car sequencing problem in constraint logic programming", Proceedings, European Conference on AI, 1988, 290-295
7. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A. & Graf, T., "Applications of CHIP to industrial and engineering problems", First International Conference on Industrial and Engineering Applications of AI and Expert Systems, June, 1988
8. Hopfield, J. J., and Tank, D.W., "'Neural' Computation of Decisions in Optimization Problems", Biol. Cybern. 52, 141-152
9. Minton, S., Johnston, M.D., Philips, A. B. & Laird, P., "Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method", American Association for Artificial Intelligence (AAAI), 1990, 17-24
10. Adorf, H.M. & Johnston, M.D., "A discrete stochastic neural network algorithm for constraint satisfaction problems", Proceedings, International Joint Conference on Neural Networks, 1990
11. McClelland, J. L., & Rumelhart, D. E., "An interactive activation model of context effects in letter perception: Part 1. An account of basic findings", Psychological Review, 88, 375-407
12. Rumelhart, D. E., & McClelland, J. L., "An interactive activation model of context effects in letter perception: Part 2. The contextual enhancement effect and some tests and extensions of the model", Psychological Review, 89, 60-94
13. Davis, L. (ed.), "Genetic algorithms and simulated annealing", Research notes in AI, Pitman/Morgan Kaufmann, 1987
14. Parrello, B.D., Kabat, W.C. & Wos, L., "Job-shop scheduling using automated reasoning: a case study of the car sequencing problem", Journal of Automatic Reasoning, 2(1), 1986, 1-42
15. Guesgen, H.W., "Connectionist networks for constraint satisfaction", AAAI Spring Symposium on Constraint-based Reasoning, March, 1991, 182-190
16. Kasif, S., "On the parallel complexity of discrete relaxation in constraint satisfaction networks", Artificial Intelligence (45) 1990, 275-286