A Generic Neural Network Approach For Constraint Satisfaction Problems

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

Department of Computer Science, University of Essex
Wivenhoe Park, Colchester, UK

Abstract

The Constraint Satisfaction Problem (CSP) is a mathematical abstraction of the problems in many AI application domains. In many of such applications timely response by a CSP solver is so crucial that to achieve it, the user may be willing to sacrifice completeness to a certain extent. This paper describes a neural network approach for solving CSPs which aims at providing prompt responses. The effectiveness of this model, which is called GENET, in solving CSPs with binary constraints is demonstrated by a simulator. Although the completeness is not guaranteed, as in the case of most of the existing stochastic search techniques, solutions have been found by the GENET simulator in all of our randomly generated problems tested so far. Since the neural network model lends itself to the VLSI implementation of parallel processing architectures, the limited number of cycles required by GENET to find the solutions for the tested problems gives hope for solving large CSPs in a fraction of the time required by conventional methods.

1 Introduction

Constraint satisfaction is the nature of the problems in many AI application domains. For instance, line labelling in vision [1], subproblems in temporal reasoning [2, 3], and resource allocation in planning and scheduling [4, 5] can all be formulated as Constraint Satisfaction Problems (CSPs). A CSP is defined as a triple (*Z, D, C*), where *Z* is a finite set of variables, *D* is a set of domains, one for each variable, and *C* is a set of constraints. Each constraint in *C* restricts the values that one can assign to a set of variables simultaneously. The task is to assign one value per variable, satisfying all the constraints in *C* [6].

To make discussion easier we shall first define a few terms here. The assignment of a value to a variable is called a *label*. A (possibly empty) set of labels is called a *compound label*. A compound label with *k* labels in it is called a *k-compound label*. A constraint is *n-ary* if it applies to *n* variables. A *binary CSP* is a CSP with unary and binary constraints only.

This research is motivated by the need to produce prompt solutions to CSPs.

There are at least two reasons for such need:

(A) Time is limited

In some applications the user has very limited time to solve the problems. For example, in allocating resources such as manpower and equipments to jobs in a workshop, one could be dealing with a very dynamic situation. For instance, jobs may be added, cancelled or given priority, and resources could become unavailable, etc. If these changes happen frequently one may not be allowed too much time for scheduling and re-scheduling.

Furthermore in many scheduling problems the user has to consider a large number of combinations of constraints. Sometimes not all the constraints can be satisfied and, therefore, the user has to consider relaxing some of them. To enable him/her to evaluate the different situations the real time response of a CSP solver is crucial.

(B) Time determines cost

There are applications where cost is a function of the response time. Imagine sitting in the control room of a container port and allocating storage locations to containers unloaded from container vessels. Each minute's delay in clearing the containers from a busy port could cost hundreds of pounds. Therefore one would like to decide as soon as possible where to store the containers.

Besides, in a catastrophic incident, such as a natural disaster or an accident in a nuclear power station, one of the coordinator's tasks may be to allocate resources, such as rescuing teams and scarce equipments, to different sites. The coordinator's reaction time may directly affect the amount of loss in terms of life and death and the amount of properties and equipments being destroyed.

The majority of existing work in CSP has been focused on problem reduction and heuristic search [6, 7]. A few techniques in such categories have been used in the constraint programming language CHIP. For symbolic variables, CHIP uses the Forward Checking algorithm together with the Fail First Principle (FC-FFP) heuristic [7]. CHIP's strategy has worked reasonably well in certain real life problems [8, 9, 10]. Unfortunately, CSPs are NP-hard in nature. The size of the search space in a CSP is typically $O(d^N)$, where d is the size of each domain in D (assuming, for simplicity, that all domains have the same size) and N is the number of variables in the problem. For CSPs with, say, $N = 10000$ variables and $d = 50$, the search space will be so huge (50^{10000}) that even a good heuristic algorithm may not be able to produce any answer within a tolerable period of time.

Speed could be gained in CSP solvers by using parallel processing architectures. Indeed, Saletore and Kale show that to a certain extent, linear speedup is achievable by using parallel architectures [11]. However, as Kasif points out, problem reduction (which CHIP uses) is inherently sequential, and it is unlikely that a CSP can be solved in logarithmic time by using only a polynomial number of processors [12].

Neural Network (NN) techniques give hope to a higher degree of parallelism in solving CSPs. Although NNs normally do not guarantee to find solutions even when some exist (as they could be trapped in local minima or local maxima), their

stochastic hill-climbing nature may also lead them to producing solutions much more quickly (if local minima are avoided). Hopfield and Tank's work on the Travelling Salesman Problem is one such example [13]. The attempt to apply neural network techniques to solve CSPs has led to the discovery of the Heuristic Repair Method, which is based on a heuristic called Min-conflict [14, 15]. The Heuristic Repair Method can solve the million-queens problem in minutes.

Other work in applying *connectionist* approaches to CSPs can be found in [16, 17, 18 and 19]. Swain and Cooper [16] propose to use a connectionist approach to problem reduction (through maintaining arc-consistency [16]), and Cooper [17] applies this technique to graph matching. Guesgen [18] extends Swain and Cooper's approach to facilitate the generation of solutions from the converged network. However, generating solutions from the converged network is far from trivial. Collin et al [19] point out that even for relatively simple constraint graphs, it is difficult to find a general parallel computational model which guarantees completeness. Besides, all these approaches require massively parallel architectures.

The major problem with neural network approaches in general is the danger of the network settling in local minima, which prevents the problem solver from finding solutions when some exist. Our analysis and experiments have shown that the Heuristic Repair Method is only effective for CSPs for which there exist a large number of solutions, such as the N-queens problem with large N [20].

In this paper, we describe a generic neural network approach for solving CSPs with binary constraints. The model that we propose is called GENET. The effectiveness of GENET is demonstrated by a simulator, which can generate connected networks dynamically for binary constraint problems, and simulate 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 high degrees of parallelism. The experimental results of applying the GENET simulator to randomly generated CSPs have been very favorable, which indicates that massive parallelism is not necessary to achieve a few orders of magnitude speedup over sequential heuristic search method.

2 NETWORK MODEL

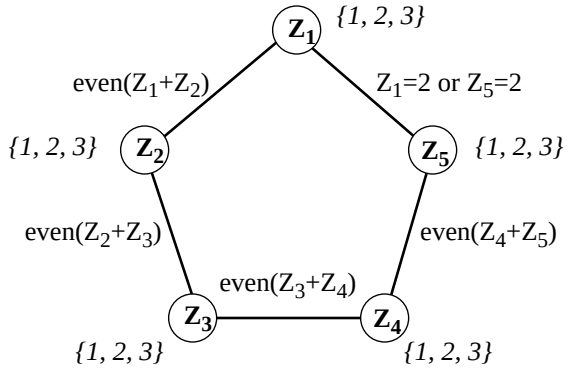
A binary constraint problem is represented in GENET in the following way. One node in the NN is used to represent one label. Each node has a binary output - either it is *on* (*active*) or it is *off* (*inactive*). If a node is on, it means the corresponding assignment is being made. The set of all nodes which represents the labels for the same variable forms a cluster. The connections among the nodes are constructed according to the constraints *C*, with an inhibitory connection between incompatible nodes and no connections between compatible nodes. The weight of all the connections are initialized to *-1*. Figure 1 shows a CSP and its network in GENET.

By convention, we shall denote the state of node *i* as s_i, which is either *1* for *on* or *0* for *off*. The weight of the connection 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 its connected nodes' states. Only one node in each cluster will be allowed to turn on. Therefore every state of the network represents an assignment of

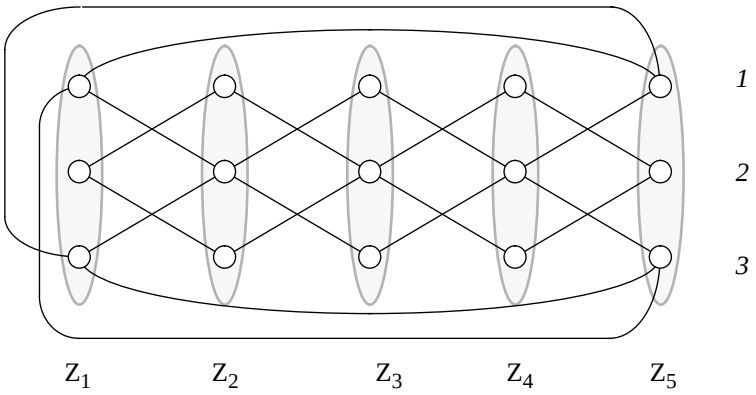
one value per variable. Any network state in which no two ON nodes are connected represents a solution to the CSP.

3 Network Convergence and Escaping Local Minima

The network convergence procedure is as follows. Initially one node in each cluster is randomly selected to turn on. Then, in each convergence cycle, every node calculates its input. In each cluster, the node that receives the maximum input will be turned 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 which, when being assigned to the corresponding variable, violates the fewest



(a) A CSP with 5 variables, Z_1, Z_2, Z_3, Z_4 and Z_5, all with the same domain {1, 2, 3}



(b) The network corresponding to (a) generated by GENET

Figure 1. A CSP and its corresponding network generated by GENET

constraints. In tie situations, if one of the nodes in the tie was on in the previous cycle, it will stay on. If all the nodes in the tie were off in the previous cycle, a random choice is made to break the tie. This is to avoid chaotic or cyclic wandering of the network states. We have experimented in breaking ties randomly, as is done in the Heuristic Repair Method [15]. But we find this strategy ineffective in problems which have few solutions.

If and when the network settles in a stable state, GENET checks if that state represents a solution. As stated before, a state in which all the active nodes have zero input represents a solution. Otherwise, it represents a local minimum.

When the network settles in a local minimum, the state update rule fails to make alternative choices for variable assignments based on the local information received at each cluster of nodes. It would appear that introducing randomness or noise in the state update rule, in a manner of the simulated annealing as suggested in the literature [21], might help in escaping local minima. However, this will degrade the overall performance drastically, which could make this approach not effective enough for solving any real life problems. Instead, we have adopted a heuristic learning rule which updates the connection weights as follows:

$$w_{ij}(t+1) = w_{ij}(t) - s_i(t) \times s_j(t)$$

where t is a discrete time instance measured in terms of cycles. This learning rule effectively does two things: (a) it continuously reduces the value of the current state until it ceases to be a local minimum, and (b) it reduces the possibility of any violated constraint being violated again. Hopefully, after sufficient learning cycles, the connection weights in the network will lead the network states to a solution. The network convergence algorithm is shown in pseudo codes in Figure 2.

The states of all the nodes are revised in parallel asynchronously in the model. The outer REPEAT loop terminates when a solution is found, or some resource is exhausted, which could mean that the maximum number of cycles have been reached, or the time limit has been exceeded. The resources available vary from application to application.

4 Experiments and Evaluation of GENET

In order to test the effectiveness of GENET for solving CSPs, a GENET simulator is constructed. Thousands of tests have been performed on randomly generated and specially designed CSPs to evaluate the performance of GENET. Random problems are generated with the following 5 parameters varied:

- N the number of variables;
- D the maximum size of domains;
- d the average size of individual domains ($d \leq D$) - the introduction of d in addition to D is to allow us to test problems which may have different domain sizes;
- p_1 the percentage of pairwise variables being constrained - in the generated problem, there are $p_1 \times \frac{N \times (N-1)}{2}$ constraints; and

p_2 the percentage of compound labels satisfying each binary constraint; e.g. if there exists a binary constraint between variables *X* and *Y*, and the domain sizes of *X* and *Y* are d_1 and d_2, then p_2 of the $d_1 \times d_2$ combinations of labels for *X* and *Y* are compatible with each other.

The probability for any two randomly assigned variables being compatible, p_c, can be obtained as follows:

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

The probability of an *N*-compound label violating no constraint is defined as the *tightness* of a problem. With *N* variables:

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

```
PROCEDURE GENET
BEGIN
  One arbitrary node per cluster is switched ON;
  REPEAT /* network convergence: */
    REPEAT
      Modified := False;
      FOR each cluster DO IN PARALLEL
        BEGIN
          On_node := node which is at present ON;
          Label_Set := the nodes with the maximum input;
          IF NOT (On_node in Label_Set) THEN
            BEGIN
              Modified := True;
              On_node := OFF;
              Switch an arbitrary node in Label_Set to ON;
            END
        END
    UNTIL (NOT Modified); /* the network has converged */
    /* learning: */
    IF (sum of input to all ON nodes < 0) /* in local minma */
      THEN FOR connection c between nodes x and y DO IN PARALLEL
      IF (both x and y are ON)
        THEN decrease the weight of connection c by 1;
  UNTIL (input to all ON nodes are 0) OR (any resource exhausted)
END
```

Figure 2. GENET convergence procedure

A problem is *tight* if its tightness is small. The GENET simulator is tested on problems with varying number of variables, domain sizes and degrees of tightness (by varying p_1 and p_2). Tests have been focused on tight problems to see how likely GENET is to miss solutions. 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. When GENET reports a failure, the problem is passed to a program which performs a complete search to check if GENET has missed any solution. For all the problems tested, GENET is capable of converging on solutions in solvable problems, and reporting failures in insoluble problems.

The time required by GENET to find solutions is measured by the number of convergence cycles required by the simulator. For CSPs in which $N = 170$, $D = d = 6$, $p_1 = 10\%$ and $p_2 = 85\%$, GENET takes just over 100 cycles to terminate. For a fully parallelized (with $d \times N$ parallelism) VLSI implementation one cycle may take a few hundred nano seconds. Thus, a problem of the aforementioned size can be solved in terms of tens to hundreds of micro seconds. Compared with a sequential heuristic search which would take over 20 hours to solve the same problem, the speedup is in the order of 10^8. The speedup of such a high order implies that a very high overhead could be affordable if the full parallelism is not supported. In fact, a parallelism of d would give a respectable speedup, as shown below.

Take the above problem for example. Suppose we have only d parallel processors. This would allow us to process one cluster at each processing cycle. To allow for more adverse situations, suppose each processor has only one input port. This would mean that one network convergence cycle had to be completed in $N(N-1)$ processing cycles (N clusters to be processed in order and each cluster needs $N-1$ cycles for adding up the active input from all other clusters). Since $N = 170$, the speedup will be degraded down to the order of 10^4. Considering this 4 orders of magnitude speedup being achievable at a moderate parallelism (d) and with a relatively simple and low cost processor (one input port), we may conclude that the GENET approach indeed fulfills our initial research objectives.

Another interesting finding in our experiments is that the weights in the network never fall below -50 and the total input to a node never fall below -100. The range of these values could be further reduced if floating points instead of integers were used for the weights. This finding is important for the analogue VLSI implementation of the GENET model, as shown below.

5 VLSI Implementation

A full discussion of a VLSI implementation of the GENET model is beyond the scope of this paper. In this section, we will simply show the feasibility of such implementation using analogue techniques.

The computation in each GENET convergence cycle can be divided into two parts: (a) the summation of input signals to a node, and (b) the competition within a cluster. The summation can be realized by a linear operational amplifier and the competition can be realized by an analogue channel comparator [22].

Figure 3 shows a schematic diagram for the design of one cluster of nodes, where

the vertical box represents the channel comparator with one channel receiving the total input signal of each node in the cluster. One single-bit register is associated to a channel, as shown to the right hand side of the vertical box (labeled D), to store the state of the corresponding node. The channel that receives the highest signal level will be switched on, and consequently its register will be set. All other channels will be switched off and so are their registers.

The state of the registers are also fed back to the channel comparator so that if the channel with the highest signal level already has its associated register set, a signal of *no change* is generated for this cluster. Thus, local minima can be detected. The comparator also checks the highest signal level against a *null* reference signal. If they match, a signal of *no violation* is generated. Henceforth, the solution states can be detected. The tie situation is solved by the natural deviation in the circuitry parameters and the feedback from the state registers.

Clearly the sensitivity of the channel comparator depends on the minimum interval amongst the levels of the signals being compared, and the range of signal

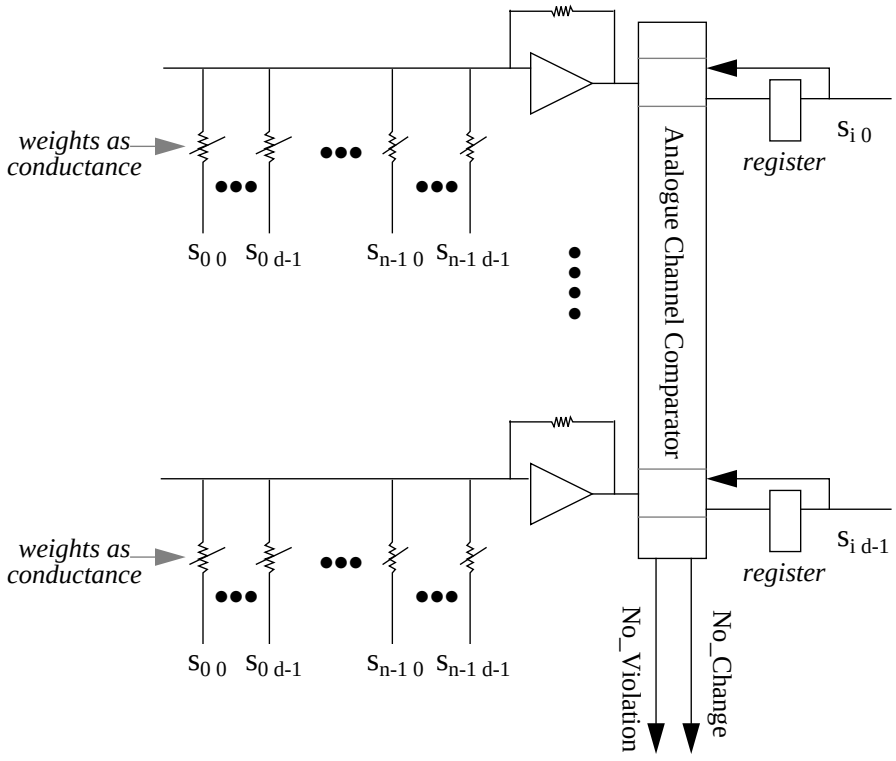


Figure 3. The design of one cluster of nodes, where s_{ij} represents the state of the *j*th node in cluster *i*

levels is limited by the VLSI technology. However this is not a problem with the GENET model because, as mentioned before, the weights and total input to a node are well limited. Firstly, a fine tuning of the weights' granularity can help reduce the total range of the signal levels. Secondly, the saturation at the lower end of the signal levels does not affect the competition. Lastly, should it be necessary, a uniform signal level shift can be applied to solve the problem.

It should be noted that Figure 3 only shows the design principle. The technical detail of a cascadable VLSI architecture for GENET is described in [22], which proposes an even simpler design that does not require modifiable resistors for learning purposes. The weights are stored off chip, the size of the channel comparator is expandable by cascading such neuro-chips vertically, and the number of clusters processed concurrently is expandable by cascading such neuro-chips horizontally.

6 Future Work

Research in GENET is on-going. Our current research focus is in the following directions. Firstly, we are evaluating GENET's performance in solving partial constraint satisfaction problems (PCSPs) [23]. One instance of the PCSPs which is useful for scheduling is to minimize the number of constraints violated when the problem is over-constrained. The PCSP is difficult because one can not use constraints to prune off as much of the search space in it as in the standard CSP (because the optimal solution may violate some constraints).

The second major direction that we are investigating is to handle general constraints. For non-binary problems and problems in which the values are not simple ground terms, a multilayer structured network will be required. The number of layers required depends on the complexity of the value structure. We have performed preliminary tests in applying GENET to networks constructed (by the experimenters) for instances of the car-sequencing problem [8, 24]. Like all the other tests so far, GENET finds solutions in those networks. Our approach to the car-sequencing problem is outlined in [20].

Our long term research direction is to fabricate VLSI neuro-chips for constructing CSP solvers for real life applications.

7 Concluding Summary

We have pointed out that there are applications in which timely response by CSP solvers is so crucial that a limited degree of sacrifice in completeness by the CSP solver can be justified. For such applications, we have proposed a neuro-network based model called GENET whose aim is to produce solutions fast enough for real time applications. A simulator has been built to demonstrate the effectiveness of GENET. Although completeness in GENET is not guaranteed, the GENET simulator has not yet failed in finding solutions for thousands of randomly generated CSPs which vary significantly in their size and tightness. We have shown the feasibility of building hardware for GENET. The fact that relatively few cycles are needed by the GENET simulator to find solutions gives hope for solving CSPs in a fraction of the time required by conventional techniques.

Acknowledgment

The authors have benefited from discussions with Dr J. Ford, Dr J. Reynold and Kar-Lik Wong in this research. Jenny Emby has greatly improved the readability of this paper.

References

[1] 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

[2] Tsang, E.P.K., “The consistent labelling problem in temporal reasoning”, Proc. AAAI Conference, Seattle, July, 1987, 251-255

[3] Dechter, R., Meiri, I. & Pearl, J., “Temporal constraint networks”, Artificial Intelligence, 49, 1991, 61-95

[4] Stefik, M., “Planning with Constraints (MOLGEN: part 1)”, Artificial Intelligence 16, 1981, 111-140

[5] Prosser, P., “Distributed asynchronous scheduling”, PhD Thesis, Department of Computer Science, University of Strathclyde, November, 1990

[6] Mackworth, A.K., “Consistency in networks or relations”, Artificial Intelligence 8(1), 1977, 99-118

[7] Haralick, R.M. & Elliott, G.L., “Increasing tree search efficiency for constraint satisfaction problems”, Artificial Intelligence 14(1980), 263-313

[8] Dincbas, M., Simonis, H. & Van Hentenryck, P., “Solving car sequencing problem in constraint logic programming”, Proceedings, European Conference on AI, 1988, 290-295

[9] 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

[10] Perrett, M., “Using Constraint Logic Programming Techniques in Container Port Planning”, ICL Technical Journal, May 1991

[11] Saletore, V.A. & Kale, L.V., ”Consistent linear speedups to a first solution in parallel state-space search”, Proc. AAAI, 1990, 227-233

[12] Kasif, S., “On the parallel complexity of discrete relaxation in constraint satisfaction networks”, Artificial Intelligence (45) 1990, 275-286

[13] Hopfield, J. J., and Tank, D.W., “‘Neural’ Computation of Decisions in Optimization Problems”, Biol. Cybern. (52) 1985, 141-152

[14] Adorf, H.M. & Johnston, M.D., “A discrete stochastic neural network algorithm for constraint satisfaction problems”, Proceedings, International Joint Conference on Neural Networks, 1990

[15] Minton, S., Johnston, M.D., Philips, A. B. & Laird, P., “Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method”, Proc. AAAI, 1990, 17-24

[16] Swain, M.J. & Cooper, P.R., “Parallel hardware for constraint satisfaction”, Proc. AAAI, 1988, 682-686

[17] Cooper, P., ”Structure recognition by connectionist relaxation: formal analysis”, Proc. Canadian AI Conference (CSCSI), 1988, 148-155

[18] Guesgen, H.W., “Connectionist networks for constraint satisfaction”, AAAI Spring Symposium on Constraint-based Reasoning, March, 1991, 182-190

[19] Collin Z., Dechter, R. & Katz, S., “On the Feasibility of distributed constraint satisfaction”, Proc. International Joint Conference on AI, 1991, 318-324

[20] Wang, C. J., & Tsang, E. T. K., “Solving constraint satisfaction problems using neural networks”, Proc. IEE Second International Conference on Artificial Neural Networks, 1991

[21] Davis, L. (ed.), “Genetic algorithms and simulated annealing”, Research notes in AI, Pitman/Morgan Kaufmann, 1987

[22] Wang, C.J., “A cascadable parallel architecture for GENET”, Research Notes, Department of Computer Science, University of Essex, forthcoming.

[23] Freuder, E.C., “Partial constraint satisfaction”, Proc., 11th International Joint Conference on AI, 1989, 278-283

[24] 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