A CASCADABLE VLSI DESIGN FOR GENET

Chang J. Wang and Edward P. K. Tsang

Abstract

Constraint Satisfaction Problems (CSPs) are at the heart of many AI applications. The currently existing constraint programming languages and systems are mostly based on heuristic search techniques. The major problems of these techniques are the limited parallelism available in the algorithms and the inadequacy of handling over-constrained CSPs. GENET is a competitive neural network model developed for solving CSPs with large size and tight constraints. It realizes a stochastic heuristic search in a fully parallelized processing fashion.

This paper presents the VLSI design of a cascadable GENET module, which may be configured to suit the structure of the CSPs in application. A realistic estimation of the potential speed-up over the existing programming languages and systems is given, which is based on the result of software simulation of the GENET's behaviour in solving over several thousands of randomly generated CSPs with various tightness of constraints. It is concluded that GENET, when implemented in VLSI technology with a moderate parallelism, a significant speed-up over the conventional approaches is possible.

Keywords:

Constraint Satisfaction, VLSI, neural networks, stochastic search, learning algorithm, parallel architecture.

A CASCADABLE VLSI DESIGN FOR GENET

Chang J. Wang and Edward P. K. Tsang

Department of Computer Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom

INTRODUCTION

This paper presents a VLSI design for a competitive neural network model, known as GENET (Wang and Tsang 1991), for solving Constraint Satisfaction Problems (CSP). The CSP is a mathematical abstraction of the problems in many AI application domains. In essence, a CSP can be defined as a triple (Z, D, C), where Z is a finite set of variables, D is a mapping from every variable to a domain, which is a finite set of arbitrary objects, and C is a set of constraints. Each constraint in C restricts the values that can be simultaneously assigned to a number of variables in Z. If the constraints in C involve up to but no more than n variables it is called an n-ary CSP. The task is to assign one value per variable satisfying all the constraints in C (Mackworth 1977). In addition, associated with the variable assignments might be costs and utilities. This turns CSPs into optimization problems, demanding CSP solvers to find a set of variable assignments that would produce a maximum total utility at a minimal cost. Furthermore, some CSPs might be over-constrained, i.e. not all the constraints in C can be satisfied simultaneously. In this case, the set of assignments to a maximum number of variables without violating any constraints in C, or the set of assignments to all the variables which violates a minimal number of constraints might be sought for.

Problems that may be formalized as the above CSPs are legion in AI applications. For instance, line labelling in vision (Waltz 1975), temporal reasoning (Tsang 1987, Dechter et al 1991), and resource allocation in planning and scheduling (Prosser 1990, Dincbas et al 1988), to name a few, are all well known CSPs.

The conventional approaches for solving CSPs are based on problem reduction and heuristic search (Mackworth 1977, Haralick and Elliott 1980). A few techniques in such categories have been used in commercial constraint programming languages, such as CHIP, Charme and Pecos. Prolog III and a functionally similar language CLP(R) are based on linear programming techniques for handling continuous numerical domains. For symbolic variables, CHIP uses the Forward Checking with Fail First Principle (FC-FFP) heuristic (Haralick and Elliott 1980). CHIP's strategy has worked reasonably well in certain real life

problems (Dincbas et al 1988). However, since CSPs are NP-hard in nature, the size of the search space for solving a CSP is inherently $O(d^N)$, 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. Thus, search techniques, even with good heuristics, inevitably suffer from exponential explosion in computational complexity. Hence, these techniques will not be effective for solving large problems, especially, when timely responses from the CSP solvers are crucial (e.g. in interactive and real time systems). Although speed could be gained in problem reduction and search-based CSP solvers by using parallel processing architecture, as Kasif (1990) points out, problem reduction is inherently sequential, and it is unlikely that a CSP can be solved in logarithmic time by using only a polynomial number of processors.

Swain and Cooper (1988) proposed a *connectionist* approach that can fully parallelize the problem reduction in binary CSPs, implementing an arc-consistency maintenance algorithm which attempts to reduce the size of the variable domains by deleting values which can never be part of any solution (cf. Mackworth 1977). In this approach, a CSP is represented as a network. The nodes are organized in a two-dimensional structure, with each column corresponding to one variable and containing all the nodes that correspond to some value that can be assigned to the variable. An arc in the network represents the compatibility between the connected two nodes. It employs one JK flip-flop for each node, thus $N \times d$ JK flip-flop are required for N variables each with a domain size d; and one JK flip-flop for each arc, thus $N^2 \times d^2$ JK flip-flop are required. Accordingly, enough combinational logic would be required to control setting/resetting the flip-flops.

Initially, all the node flip-flops are set *on* and arc flip-flops are set according to the constraints in *C* - *on* for compatible values and *off* otherwise. Then, the network iterates to delete the nodes, by resetting the node flip-flops, that are constrained by every possible assignment of its neighbour variable. This is known as the relaxation procedure. However, solutions to the CSP are not directly generated by this procedure. The time complexity for the relaxation procedure is $O(Nd)$ in terms of cycles.

Guesgen (Guesgen and Hertzberg 1992) extends Swain and Cooper's approach to facilitate the generation of solutions from the converged network. This approach distributedly encodes the possible solutions in bit strings stored in the remaining nodes. Additional $O(N^3 \times d^3)$ JK flip-flops are required for this purpose. Thereafter, solutions may be found by succession of bit-wise AND operations over N binary strings, selected one from each set of the nodes that represent the values assignable to a single variable. If the result of the AND operations is non-zero, a solution is thus found as the selected nodes. Unfortunately, the number of possible selections of the bit-strings is an exponential function of the *compatible* variable domain size, assuming a uniform domain size for each variable after the network converges. It should be noted that arc-consistency maintenance may not always reduce the variable domain sizes significantly, and the space complexity of $O(N^3 \times d^3)$ will quickly reach the capacity limit of the current VLSI technology.

To summarize, the constraint programming languages and systems implemented on conventional workstations are too slow for solving problems of realistic size, e.g. those that may have hundreds of variables with a domain size of a few tens. The *connectionist* approach based on VLSI technology is not scalable due to its massive internal connections for setting/resetting the J-K flip/flops and, hence, will be severely limited by the chip size currently available. In addition, none of these approaches can handle over-constrained CSPs which are at the heart of many real-life problems. To address these problems, a generic neural network approach, known as GENET, that effectively realizes a stochastic heuristic search, has been proposed to speed up the performance of CSP solvers (Tsang and Wang 1991). In the next section, we shall briefly describe the GENET model. This will be followed by a presentation of a cascadable VLSI design for implementing GENET.

GENET

GENET was inspired by the early attempt to apply neural network technique to solve binary CSPs (Adorf and Johnston 1990), which has led to the discovery of the Heuristic Repair Method (Minton and Johnston 1990). The Heuristic Repair Method is based on a heuristic called the *Min-conflict heuristic*. Initially, variables are picked at random to be assigned values which violates the least number of constraints (the Min-conflict heuristic). Then the program iterates to check the compatibility of the assignments, dividing the variables into two sets: *A*, containing the variables whose assignment violates no constraints, and *B*, containing the variables whose assignment violates some constraints. A variable in *B* is then selected to be *repaired*. It will be assigned a value that violates the least number of constraints (Min-conflict). After the repair, the variables are re-examined and transferred from *A* to *B* or vice versa, according to their status. This procedure iterates until set *B* becomes empty (which is not guaranteed to happen). Then set *A* contains a solution to the CSP.

The Heuristic Repair Method has been demonstrated to be capable of solving the million queens problem in minutes when simulated on a single processor. However, a vital problem with the Heuristic Repair Method is that it can easily be trapped in local minima. Hence, it may only be useful for solving loosely constrained CSPs where many solutions exist and are relatively evenly distributed in the search space.

The GENET approach is based on the principle of Min-conflict, but incorporates a learning algorithm to escape local minima. GENET is a competitive neural network model that has binary neurons (*on/off* state) and symmetric connections. The connectivity is sparse because only constrained neurons are connected with a negative weight, initially set to be *-1*. The neurons in the network are clustered to represent variable domains. For instance**,** if the CSP to be solved has *N* variables and each variable may be assigned with one of *d* values at one time, then the network will consist of *N* clusters, each of *d* neurons, and the connections will be set up according to the constraints in the original CSP. **Figure 1** shows a cluster of neurons, where the input signals are the output states of the connected neurons that represent incompatible (constrained) value assignments.

The neuron states are updated in convergence cycles. In each convergence cycle, every neuron adds up the weights connected to active neurons. The neurons in the same cluster compete with each other and the one that receives a maximum input, meaning that less constraints would be violated if it is turned on rather than others, will be selected to turn on and the others are turned off. In case of tie situations, preference is given to the *on* neuron, if there is one in the tie, else a random choice will be made. The convergence cycles continues until all the *on* neurons have a nil input, which means that no constraints are violated if the

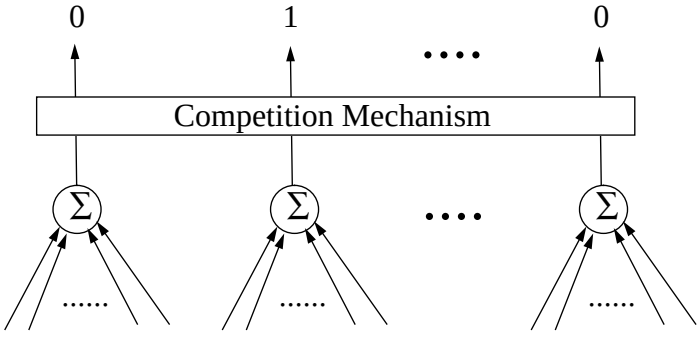

Figure 1 An example of a cluster of neurons in GENET, assuming the second neuron receives the maximum input.

variables are assigned the values indicated by the current network state.

This convergence procedure resembles the Min-Conflict heuristic but is deterministic in identifying local minima. The network is in a local minimum, if a convergence cycle fails to update any neuron's state whilst some of the *on* neurons still receive negative inputs. This means that some constraints are violated by the variable assignments represented by the current network state. In this case, a learning procedure is applied to penalize the connection weights that link two active neurons. The time required by GENET to solve a CSP is measured in terms of number of convergence cycles it takes for the network to settle down with a solution.

Extensive experiments have been carried out over several thousands of randomly generated CSPs with various tightness of constraints and various problem sizes. This approach has also been tried on the car-sequencing problem (Wang and Tsang 1991). Although the completeness is not guaranteed in GENET, our experiments have shown that GENET always finds solutions for solvable problems. More interestingly, when GENET is given insoluble problems, optimal solutions are always found, where optimality is measured by the number of constraints being violated. This is verified by a branch-and-bound program. The result of our experiments can be summarized as follows.

First of all, given a fixed tightness among the constraints, the number of convergence cycles required to solve significantly large problems is limited. For example, with the tightness used in our tests, problems with 200 variables which have uniform domain size of 6, require no more than 200 cycles to be either solved or concluded over-constrained. It should be pointed out that problems of this size have a potential search space of $O(6^{200})$, which is the largest problem we could simulate within a tolerable time on a Solbourne 902/5E with two SPARC processors. If a full parallelism of $N \times d$ is supported by VLSI implementation of GENET and if a convergence cycle time were to take a few hundreds of nanoseconds, such a problem can be solved in terms of tens of microseconds. For reference, a program which employs the FC-FFP heuristic (a strategy integrated in CHIP) would take over 40 minutes CPU time when run on the Solbourne 902/5E. This implies that the VLSI implementation of GENET would provide a potential speed gain in an order of 10^6 to 10^8 over existing CSP languages run on commercial workstations. The potential speed-up of such a high order means that a very high overhead is affordable should full parallelism be not possible. In fact, a parallelism of d, which is highly conceivable with current VLSI technology, will lead to a speed-up of $O(10^3)$ over the sequential heuristic search approach. This will be explained in detail later.

Secondly, in our experiments, the weight values never fall below -50 and the total input to a neuron never fall below -100. If this proves to be the general case, then 8-bit weights and 8-bit accumulation registers should provide sufficient precision in digital implementation. This also gives guidelines for implementations using analog VLSI technology. Below, we discuss VLSI implementation of GENET based on these guidelines.

A CASCADABLE VLSI ARCHITECTURE FOR GENET

The computation in GENET consists of two parts. First, every neuron will have to sum up weighted input signals, and then all the neurons in the same cluster will compete with each other to select the winner. Since the GENET is a binary neural network, the former has become a simple summation of the weights connected to *active* (*on*) neurons in the network. This would significantly simplify the design. To implement GENET, the weight storage, summation mechanism in each neuron, and the competition mechanism in each cluster will have to be considered.

In the current literature on implementing neural networks, various VLSI technologies,

such as analog (e.g. Graf and Jackel 1989), digital (e.g. Yasunaga et al 1990), hybrid (e.g. Morishita et al 1990) and pulse-modulated (e.g. Murray 1989 and Tomberg and Kaski 1990), have been reported for implementing synapse memories, neuronal activation functions, and the multiplication and summation mechanism in a parallelized fashion. Any of these techniques may be borrowed to implement GENET as and when they are commercially mature. For competitive neural networks, Lazzaro et al have designed a sub-threshold competition mechanism (WTA) that has been used in VLSI sensory systems.

A requirement essential to GENET applications is that the competition mechanism should allow the size of each cluster to vary dynamically at run time. This is because the size of a cluster depends on the domain size of the variable it represents, which varies from application to application. Therefore, the size of a cluster as well as the connections between clusters has to be programmable. It would be difficult for sub-threshold WTA to meet these requirements. For these reasons, we investigated the design of a multiple signal comparator (*N*-Comparator) as the competition mechanism. The function of this comparator is to take *n* input signals and identify which of them is the maximum, or a maximum one if there is a tie. The input signals may be analog or digital depending on the technology adopted for implementing other parts of the neurons. However, the output signals are logic *1* or *0*, one for each input. This comparator will replace the upper box of **Figure 1**. The critical requirement of the design is that such comparators should be cascadable via switches so as to vary the number of signals to be compared simultaneously.

Analog N-Comparator

The design of the analog comparator is based on comparing voltage signals. **Figure 2** shows a single *N*-comparator designed in CMOS technology. The input V_1 through to V_n are analog voltage signals to be compared and the output O_1 through to O_n are analog voltage signals that amplify the difference between the maximum input signal and the rest of the input signals. Cascading a few stages of such comparators can amplify this difference so much that the final output signals will become logical signals, indicating clearly which input signal wins the competition.

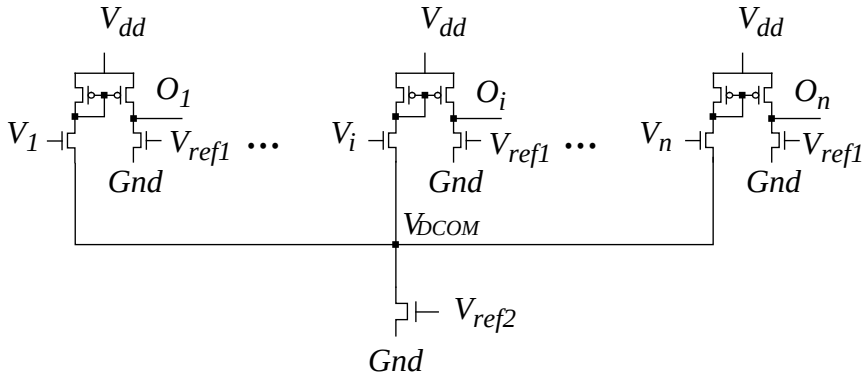

Figure 2 Analog multiple signal comparator

The sensitivity of this *N*-comparator, i.e. the minimum voltage difference among the input signals that may drive their corresponding outputs into logical *on* or *off*, correlates with the number of signals, i.e the value of *n*, due to the common current sink. SPICE simulation shows that for a single stage *4*-comparator, using standard 2μ CMOS technology and V_{dd}*=5V*, the sensitivity is about *0.7V*, but non-linear across the operating range (*0.86V* to *4.25V* in our simulation). In terms of quantized analog signals, this means that four different levels of voltage signals, each with a *±0.22V* safety margin, can be reliably compared. This

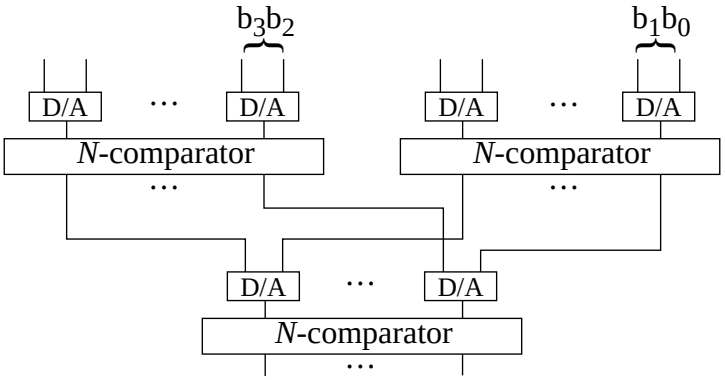

Figure 3 Cascading 2-bit comparators for 4 bit signals

corresponds to a two-bit comparator, with a D/A at each input line.

Whilst the sensitivity can be greatly improved by adding more stages to amplify the voltage difference or using bipolar technology, for true analog input signals, it is sufficient to show that, for digital signals, an analog comparator with the above sensitivity can always be cascaded to realize higher precision, as shown in **Figure 3**. This also means that higher precision digital values may be compared in a kind of serial fashion; and the comparison speed only depends on the number of bits in the comparands, rather than the number of the signals being compared simultaneously.

Digital *N*-Comparator

The digital *N*-comparator compares *n* digital values, simultaneously, bit by bit from the *msb* to *lsb*, and the maximum value will be identified by the success of a series of bit-wise comparisons. This can be realized by using a bus structure with *p*-MOSFETs to drive the bus high and a common current sink to shunt the bus low.

Starting from the *msb*, each signal puts a bit onto the corresponding bus line and, after some time delay for the bus to settle down, compares its bit value with that on the bus line. If they are the same, it continues to do the same with the next bit. Otherwise, it will stop putting its lower bits onto the bus and signal a failure in the competition. If a signal succeeds in every bit position, this signal contains a maximum value. **Figure 4** show the logic for a

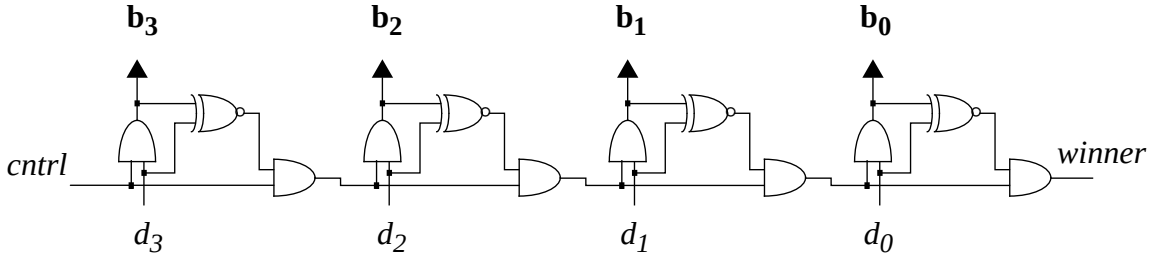

Figure 4 Control logic for one signal in a digital multiple signal comparator

signal of 4-bit value, where $\mathbf{b_3b_2b_1b_0}$ are connected to the bus lines; $d_3d_2d_1d_0$ represent the 4-bit digital input signal; the signal *cntrl* may prohibit the input signal from participating the competition; and the signal *winner*, if true, indicates this input has won the competition.

The advantage of this design is two-fold. Firstly, the value *n* can be large due to the digital bus structure. Secondly, cascading such comparators by connecting their buses together via switches would allow dynamic clustering at run time. The latter would facilitate programmable network configuration. The speed of comparison only depends on the

number of bits in the values being compared, three gates' delay plus a bus charge/discharge per bit, as may be seen in **Figure 4**. SILOS simulation under Cadence™ showed that an 8-bit *63*-Comparator can reliably complete the comparison in well under *40ns*. An added benefit is that the signal present on the bus is in fact the maximum value of the digital numbers being compared. This simplifies the detection of the winning value, which is required to signal the successfully converged network state.

Note. In the above description of the *N*-comparator, both analog and digital, the necessary control logic and status registers are omitted for the sake of simplicity. The omitted control logic includes: a) the circuit that detects the *nil* value of the winning input, indicating that no constraints were violated; b) the circuit that determines which of the winning input signals, in the case of a tie, is going to turn *on* its corresponding output state; and c) the circuit that detects a *no-change* in a convergence cycle, i.e. the selected neuron is already *on* in the previous cycle.

It should also be pointed out that, although the N-comparators as described above are designed to select the maximum value, similar techniques may be applied to design them for selecting the minimum value. It is also possible to compare positive values with negative ones, with a treatment of the sign. Since all the weights in GENET are negative, the N-comparator will actually be comparing the magnitude of negative values. This can be realized either by designing the comparator to select the minimum value, equally applicable to analog as well as digital design; or selecting the maximum 1's complement of the values, in the case of a digital design; or selecting the maximum voltage on reversely charged capacitors, in the case of an analog design. Therefore, we will ignore this problem in our discussions.

GENET Module

Having discussed the competition mechanism, we can now describe the building block of GENET architecture - the GENET module. The GENET module is based on the structure of a cluster of neurons. The architecture of the GENET module is shown inside the box in **Figure 5**. It consists of *n* neurons connected to an *N*-comparator. Each neuron has its own weight memory and a summation mechanism, the ellipses labelled $\sum$. The output signals of the *N*-comparator are latched into state registers, boxes labelled *S*, and fed back to the comparator for its control logic to determine the signal *no_change*. The module outputs the code of the active neuron in the cluster, and takes as input the codes of the active neurons in other clusters. The signal *no_violation* indicates the summation of the input to the winning neuron is nil, meaning no constraints have been violated. The signal *no_change* indicates that, in the current convergence cycle, the neuron selected to turn on was already on in the previous cycle. These two signals indicate the status of the cluster after a convergence cycle, e.g. whether this cluster may conform a part of a solution or a potential local minimum. If it is the latter, a learning cycle may be triggered, if necessary. The GENET module can be cascaded, by connecting the *link_control* signals, to extend the number of neurons per cluster without reducing the performance significantly.

A complete GENET network can be realized by organizing the GENET modules to suit the structure of the CSP to be solved. **Figure 6** shows an example of the overall architecture of GENET in application, where each row represents a cluster and a number of GENET modules may be cascaded for a large cluster. A chain of such GENET modules may be implemented on the same chip and dynamic configuration of the cluster size at run time is simply to program the switches. This is perfectly feasible with the digital comparator, but there might be some technical difficulty with the analog one.

For a realistic measurement of the GENET performance, let's assume that the weights

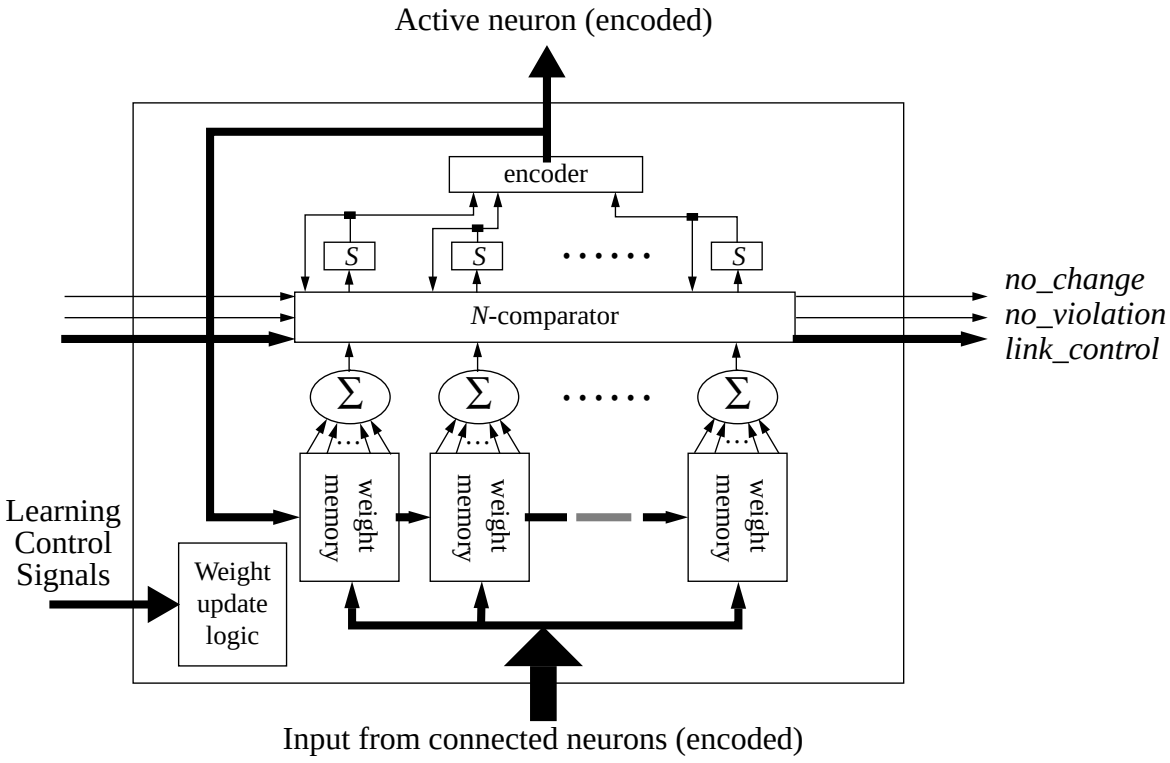

Figure 5 Architecture of a GENET module

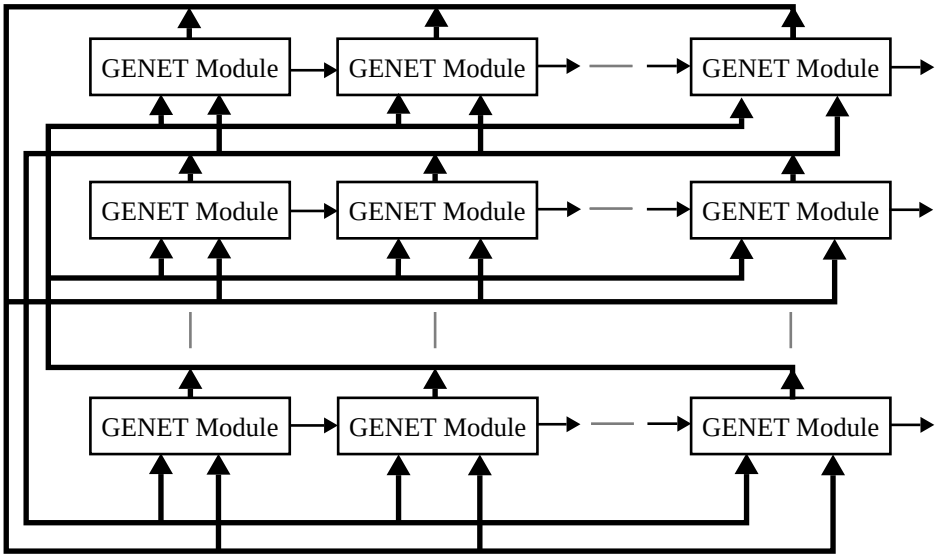

Figure 6 Overall architecture of GENET

are 8 bits and stored in off-chip fast SRAM, e.g. the 16K×4 bit INTEL C51C98-25 with 25ns access time. It means that the real memory access time can be made about 40ns, when address decoding overhead is included. Further, assume the digital comparator is integrated with a bit serial adder for the purpose of summing up the input signal to each neuron. Hence, only one input line is needed for one signal being compared. Take a CSP that has 200 variables for example, in a convergence cycle, each neuron will sum up 200 weights, because there is only one neuron in each cluster turned on. To sum up 200 weights, with each of 8 bits sequenced in, will need 200×8×40ns=64μs. To be more pragmatic, let's assume the 200 input signals are multiplexed via 20 lines. This means one convergence cycle would take 640μs. The competition time is ignorable in this case.

According to our simulation, the above problem can be solved in no more than 200 cycles, i.e. less then 200×640μs=128ms, while the FC-FFP heuristic program will take over 40 minutes of CPU time to solve the same problem. This means that the GENET approach may give a potential speed-up in the order of 10^5 against the heuristic search program. This estimation assumes a *full* parallelism of $N \times d$ neurons. In fact, even a parallelism of d would give a potential speed-up of the order of 10^3. This estimation overlooks other overheads that may occur in practice, such as updating weights in a learning phase, which may be twice the time of a convergence cycle. However, we may still conclude that the GENET, when implemented in VLSI technology with a moderate parallelism, e.g. tens of GENET modules each of tens of neurons which is highly feasible with the current VLSI technology, will out-perform the existing CSP solvers significantly.

CONCLUSIONS

Real life Constraint Satisfaction Problems are typically large and complex, and can not be satisfactorily solved by the current commercially available constraint programming languages and systems within a tolerable period of time. The competitive neural network model GENET has been developed to tackle these problems. The GENET approach provides an effective parallel algorithm for solving CSPs. When implemented in VLSI technology with a realistic, moderate parallelism, a significant speed-up over the existing constraint programming languages and systems is possible.

Acknowledgment

The authors would like to acknowledge the contribution of Kate W. C. Sin who investigated GENET behaviour with over-constrained CSPs, Nordin B. Salleh who simulated the analog design using SPICE simulator, and Hai Cheong Wong who simulated the digital design using SILOS simulator under Cadence™ EDGE®.

References

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

Dechter, R., Meiri, I. & Pearl, J., "Temporal constraint networks", Artificial Intelligence, 49, pp. 61-95, 1991.

Dincbas, M., Simonis, H. & Van Hentenryck, P., "Solving car sequencing problem in constraint logic programming", Proceedings, European Conference on AI, pp. 290-295, 1988.

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.

Graf, H. P. and Jackel, L.D., "Analog Electronic Neural Network Circuits", IEEE Circuits and Devices Magazine, pp. 44-55, July 1989.

Guesgen, H.W. & Hertzberg J., "A Perspective of Constraint-based Reasoning", Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992

Haralick, R.M. and Elliott, G.L., "Increasing tree search efficiency for constraint satisfaction problems", Artificial Intelligence 14, pp. 263-313, 1980.

Kasif, S., "On the parallel complexity of discrete relaxation in constraint satisfaction networks", Artificial Intelligence (45), pp. 275-286, 1990.

Lazzaro, J., Ryckebusch, S., Mahowald, M. A., and Mead, C. A., "Winner-Take-All Networks of O(N) Complexity", in *Advances in Neural Information Processing Systems I*, Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 1989, 703-711

Mackworth, A.K., "Consistency in networks or relations", Artificial Intelligence 8(1), pp. 99-118, 1977.

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), pp.17-24, 1990.

Morishita, T., Tamura, Y. and Otsuki, T., "A BiCMOS Analog Neural Network with Dynamically Updated Weights", IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, pp. 142-143, Feb. 1990.

Murray, A. F., "Pulse Arithmetic in VLSI Neural Networks", IEEE Micro Mag., pp. 64-74, Dec. 1989.

Prosser, P., "Distributed asynchronous scheduling", PhD Thesis, Department of Computer Science, University of Strathclyde, November 1990.

Swain, M.J. & Cooper, P.R., "Parallel hardware for constraint satisfaction", Proc. AAAI, pp. 682-686, 1988.

Tomberg, J. E. and Kaski, K. K. K., "Pulse-Density Modulation Technique in VLSI Implementations of Neural Network Algorithms", IEEE J. of Solid-State Circuits, vol. 25, no. 5, pp. 1277-1286, Oct. 1990.

Tsang, E.P.K., "The consistent labelling problem in temporal reasoning", Proc. AAAI Conference, Seattle, pp. 251-255, July 1987.

Tsang, E. P. K., & Wang, C. J., "A generic neural network approach for constraint satisfaction problems", Proc. NCM'91 Applications of Neural Networks, to be published in Series in Neural Networks by Springer Verlag, 1992.

Waltz, D.L., "Understanding line drawings of scenes with shadows", in WINSTON, P.H. (ed.) The Psychology of Computer Vision, McGraw-Hill, New York, pp. 19-91, 1975.

Wang, C. J., & Tsang, E. T. K., "Solving constraint satisfaction problems using neural networks", Proceedings, IEE Second International Conference on Artificial Neural Networks, pp. 295-299, 1991.

Yasunaga, M., Masuda, N., Yagyu, M., Asai, M., Yamada, M., and Masaki, A., "Design, Fabrication and Evaluation of a 5-Inch Wafer Scale Neural Network LSI Composed of 576 Digital Neurons", Proc. Int. Joint Conf. on Neural Networks, Vol. II, pp. 527-535, June 1990.

A CASCADABLE VLSI DESIGN FOR GENET

Chang J. Wang and Edward P. K. Tsang

Abstract

Constraint Satisfaction Problems (CSPs) are at the heart of many AI applications. The currently existing constraint programming languages and systems are mostly based on heuristic search techniques. The major problems of these techniques are the limited parallelism available in the algorithms and the inadequacy of handling over-constrained CSPs. GENET is a competitive neural network model developed for solving CSPs with large size and tight constraints. It realizes a stochastic heuristic search in a fully parallelized processing fashion.

This paper presents the VLSI design of a cascadable GENET module, which may be configured to suit the structure of the CSPs in application. A realistic estimation of the potential speed-up over the existing programming languages and systems is given, which is based on the result of software simulation of the GENET's behaviour in solving over several thousands of randomly generated CSPs with various tightness of constraints. It is concluded that GENET, when implemented in VLSI technology with a moderate parallelism, a significant speed-up over the conventional approaches is possible.

Keywords:

Constraint Satisfaction, VLSI, neural networks, stochastic search, learning algorithm, parallel architecture.

A CASCADABLE VLSI DESIGN FOR GENET

Chang J. Wang and Edward P. K. Tsang

Department of Computer Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom

INTRODUCTION

This paper presents a VLSI design for a competitive neural network model, known as GENET (Wang and Tsang 1991), for solving Constraint Satisfaction Problems (CSP). The CSP is a mathematical abstraction of the problems in many AI application domains. In essence, a CSP can be defined as a triple (Z, D, C), where Z is a finite set of variables, D is a mapping from every variable to a domain, which is a finite set of arbitrary objects, and C is a set of constraints. Each constraint in C restricts the values that can be simultaneously assigned to a number of variables in Z. If the constraints in C involve up to but no more than n variables it is called an n-ary CSP. The task is to assign one value per variable satisfying all the constraints in C (Mackworth 1977). In addition, associated with the variable assignments might be costs and utilities. This turns CSPs into optimization problems, demanding CSP solvers to find a set of variable assignments that would produce a maximum total utility at a minimal cost. Furthermore, some CSPs might be over-constrained, i.e. not all the constraints in C can be satisfied simultaneously. In this case, the set of assignments to a maximum number of variables without violating any constraints in C, or the set of assignments to all the variables which violates a minimal number of constraints might be sought for.

Problems that may be formalized as the above CSPs are legion in AI applications. For instance, line labelling in vision (Waltz 1975), temporal reasoning (Tsang 1987, Dechter et al 1991), and resource allocation in planning and scheduling (Prosser 1990, Dincbas et al 1988), to name a few, are all well known CSPs.

The conventional approaches for solving CSPs are based on problem reduction and heuristic search (Mackworth 1977, Haralick and Elliott 1980). A few techniques in such categories have been used in commercial constraint programming languages, such as CHIP, Charme and Pecos. Prolog III and a functionally similar language CLP(R) are based on linear programming techniques for handling continuous numerical domains. For symbolic variables, CHIP uses the Forward Checking with Fail First Principle (FC-FFP) heuristic (Haralick and Elliott 1980). CHIP's strategy has worked reasonably well in certain real life

problems (Dincbas et al 1988). However, since CSPs are NP-hard in nature, the size of the search space for solving a CSP is inherently $O(d^N)$, 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. Thus, search techniques, even with good heuristics, inevitably suffer from exponential explosion in computational complexity. Hence, these techniques will not be effective for solving large problems, especially, when timely responses from the CSP solvers are crucial (e.g. in interactive and real time systems). Although speed could be gained in problem reduction and search-based CSP solvers by using parallel processing architecture, as Kasif (1990) points out, problem reduction is inherently sequential, and it is unlikely that a CSP can be solved in logarithmic time by using only a polynomial number of processors.

Swain and Cooper (1988) proposed a *connectionist* approach that can fully parallelize the problem reduction in binary CSPs, implementing an arc-consistency maintenance algorithm which attempts to reduce the size of the variable domains by deleting values which can never be part of any solution (cf. Mackworth 1977). In this approach, a CSP is represented as a network. The nodes are organized in a two-dimensional structure, with each column corresponding to one variable and containing all the nodes that correspond to some value that can be assigned to the variable. An arc in the network represents the compatibility between the connected two nodes. It employs one JK flip-flop for each node, thus $N \times d$ JK flip-flop are required for N variables each with a domain size d; and one JK flip-flop for each arc, thus $N^2 \times d^2$ JK flip-flop are required. Accordingly, enough combinational logic would be required to control setting/resetting the flip-flops.

Initially, all the node flip-flops are set *on* and arc flip-flops are set according to the constraints in *C* - *on* for compatible values and *off* otherwise. Then, the network iterates to delete the nodes, by resetting the node flip-flops, that are constrained by every possible assignment of its neighbour variable. This is known as the relaxation procedure. However, solutions to the CSP are not directly generated by this procedure. The time complexity for the relaxation procedure is $O(Nd)$ in terms of cycles.

Guesgen (Guesgen and Hertzberg 1992) extends Swain and Cooper's approach to facilitate the generation of solutions from the converged network. This approach distributedly encodes the possible solutions in bit strings stored in the remaining nodes. Additional $O(N^3 \times d^3)$ JK flip-flops are required for this purpose. Thereafter, solutions may be found by succession of bit-wise AND operations over N binary strings, selected one from each set of the nodes that represent the values assignable to a single variable. If the result of the AND operations is non-zero, a solution is thus found as the selected nodes. Unfortunately, the number of possible selections of the bit-strings is an exponential function of the *compatible* variable domain size, assuming a uniform domain size for each variable after the network converges. It should be noted that arc-consistency maintenance may not always reduce the variable domain sizes significantly, and the space complexity of $O(N^3 \times d^3)$ will quickly reach the capacity limit of the current VLSI technology.

To summarize, the constraint programming languages and systems implemented on conventional workstations are too slow for solving problems of realistic size, e.g. those that may have hundreds of variables with a domain size of a few tens. The *connectionist* approach based on VLSI technology is not scalable due to its massive internal connections for setting/resetting the J-K flip/flops and, hence, will be severely limited by the chip size currently available. In addition, none of these approaches can handle over-constrained CSPs which are at the heart of many real-life problems. To address these problems, a generic neural network approach, known as GENET, that effectively realizes a stochastic heuristic search, has been proposed to speed up the performance of CSP solvers (Tsang and Wang 1991). In the next section, we shall briefly describe the GENET model. This will be followed by a presentation of a cascadable VLSI design for implementing GENET.

GENET

GENET was inspired by the early attempt to apply neural network technique to solve binary CSPs (Adorf and Johnston 1990), which has led to the discovery of the Heuristic Repair Method (Minton and Johnston 1990). The Heuristic Repair Method is based on a heuristic called the *Min-conflict heuristic*. Initially, variables are picked at random to be assigned values which violates the least number of constraints (the Min-conflict heuristic). Then the program iterates to check the compatibility of the assignments, dividing the variables into two sets: *A*, containing the variables whose assignment violates no constraints, and *B*, containing the variables whose assignment violates some constraints. A variable in *B* is then selected to be *repaired*. It will be assigned a value that violates the least number of constraints (Min-conflict). After the repair, the variables are re-examined and transferred from *A* to *B* or vice versa, according to their status. This procedure iterates until set *B* becomes empty (which is not guaranteed to happen). Then set *A* contains a solution to the CSP.

The Heuristic Repair Method has been demonstrated to be capable of solving the million queens problem in minutes when simulated on a single processor. However, a vital problem with the Heuristic Repair Method is that it can easily be trapped in local minima. Hence, it may only be useful for solving loosely constrained CSPs where many solutions exist and are relatively evenly distributed in the search space.

The GENET approach is based on the principle of Min-conflict, but incorporates a learning algorithm to escape local minima. GENET is a competitive neural network model that has binary neurons (*on/off* state) and symmetric connections. The connectivity is sparse because only constrained neurons are connected with a negative weight, initially set to be *-1*. The neurons in the network are clustered to represent variable domains. For instance**,** if the CSP to be solved has *N* variables and each variable may be assigned with one of *d* values at one time, then the network will consist of *N* clusters, each of *d* neurons, and the connections will be set up according to the constraints in the original CSP. **Figure 1** shows a cluster of neurons, where the input signals are the output states of the connected neurons that represent incompatible (constrained) value assignments.

The neuron states are updated in convergence cycles. In each convergence cycle, every neuron adds up the weights connected to active neurons. The neurons in the same cluster compete with each other and the one that receives a maximum input, meaning that less constraints would be violated if it is turned on rather than others, will be selected to turn on and the others are turned off. In case of tie situations, preference is given to the *on* neuron, if there is one in the tie, else a random choice will be made. The convergence cycles continues until all the *on* neurons have a nil input, which means that no constraints are violated if the

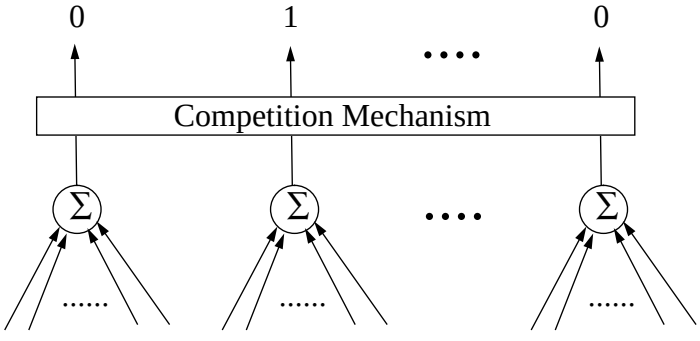

Figure 1 An example of a cluster of neurons in GENET, assuming the second neuron receives the maximum input.

variables are assigned the values indicated by the current network state.

This convergence procedure resembles the Min-Conflict heuristic but is deterministic in identifying local minima. The network is in a local minimum, if a convergence cycle fails to update any neuron's state whilst some of the *on* neurons still receive negative inputs. This means that some constraints are violated by the variable assignments represented by the current network state. In this case, a learning procedure is applied to penalize the connection weights that link two active neurons. The time required by GENET to solve a CSP is measured in terms of number of convergence cycles it takes for the network to settle down with a solution.

Extensive experiments have been carried out over several thousands of randomly generated CSPs with various tightness of constraints and various problem sizes. This approach has also been tried on the car-sequencing problem (Wang and Tsang 1991). Although the completeness is not guaranteed in GENET, our experiments have shown that GENET always finds solutions for solvable problems. More interestingly, when GENET is given insoluble problems, optimal solutions are always found, where optimality is measured by the number of constraints being violated. This is verified by a branch-and-bound program. The result of our experiments can be summarized as follows.

First of all, given a fixed tightness among the constraints, the number of convergence cycles required to solve significantly large problems is limited. For example, with the tightness used in our tests, problems with 200 variables which have uniform domain size of 6, require no more than 200 cycles to be either solved or concluded over-constrained. It should be pointed out that problems of this size have a potential search space of $O(6^{200})$, which is the largest problem we could simulate within a tolerable time on a Solbourne 902/5E with two SPARC processors. If a full parallelism of $N \times d$ is supported by VLSI implementation of GENET and if a convergence cycle time were to take a few hundreds of nanoseconds, such a problem can be solved in terms of tens of microseconds. For reference, a program which employs the FC-FFP heuristic (a strategy integrated in CHIP) would take over 40 minutes CPU time when run on the Solbourne 902/5E. This implies that the VLSI implementation of GENET would provide a potential speed gain in an order of 10^6 to 10^8 over existing CSP languages run on commercial workstations. The potential speed-up of such a high order means that a very high overhead is affordable should full parallelism be not possible. In fact, a parallelism of d, which is highly conceivable with current VLSI technology, will lead to a speed-up of $O(10^3)$ over the sequential heuristic search approach. This will be explained in detail later.

Secondly, in our experiments, the weight values never fall below -50 and the total input to a neuron never fall below -100. If this proves to be the general case, then 8-bit weights and 8-bit accumulation registers should provide sufficient precision in digital implementation. This also gives guidelines for implementations using analog VLSI technology. Below, we discuss VLSI implementation of GENET based on these guidelines.

A CASCADABLE VLSI ARCHITECTURE FOR GENET

The computation in GENET consists of two parts. First, every neuron will have to sum up weighted input signals, and then all the neurons in the same cluster will compete with each other to select the winner. Since the GENET is a binary neural network, the former has become a simple summation of the weights connected to *active* (*on*) neurons in the network. This would significantly simplify the design. To implement GENET, the weight storage, summation mechanism in each neuron, and the competition mechanism in each cluster will have to be considered.

In the current literature on implementing neural networks, various VLSI technologies,

such as analog (e.g. Graf and Jackel 1989), digital (e.g. Yasunaga et al 1990), hybrid (e.g. Morishita et al 1990) and pulse-modulated (e.g. Murray 1989 and Tomberg and Kaski 1990), have been reported for implementing synapse memories, neuronal activation functions, and the multiplication and summation mechanism in a parallelized fashion. Any of these techniques may be borrowed to implement GENET as and when they are commercially mature. For competitive neural networks, Lazzaro et al have designed a sub-threshold competition mechanism (WTA) that has been used in VLSI sensory systems.

A requirement essential to GENET applications is that the competition mechanism should allow the size of each cluster to vary dynamically at run time. This is because the size of a cluster depends on the domain size of the variable it represents, which varies from application to application. Therefore, the size of a cluster as well as the connections between clusters has to be programmable. It would be difficult for sub-threshold WTA to meet these requirements. For these reasons, we investigated the design of a multiple signal comparator (*N*-Comparator) as the competition mechanism. The function of this comparator is to take *n* input signals and identify which of them is the maximum, or a maximum one if there is a tie. The input signals may be analog or digital depending on the technology adopted for implementing other parts of the neurons. However, the output signals are logic *1* or *0*, one for each input. This comparator will replace the upper box of **Figure 1**. The critical requirement of the design is that such comparators should be cascadable via switches so as to vary the number of signals to be compared simultaneously.

Analog N-Comparator

The design of the analog comparator is based on comparing voltage signals. **Figure 2** shows a single *N*-comparator designed in CMOS technology. The input V_1 through to V_n are analog voltage signals to be compared and the output O_1 through to O_n are analog voltage signals that amplify the difference between the maximum input signal and the rest of the input signals. Cascading a few stages of such comparators can amplify this difference so much that the final output signals will become logical signals, indicating clearly which input signal wins the competition.

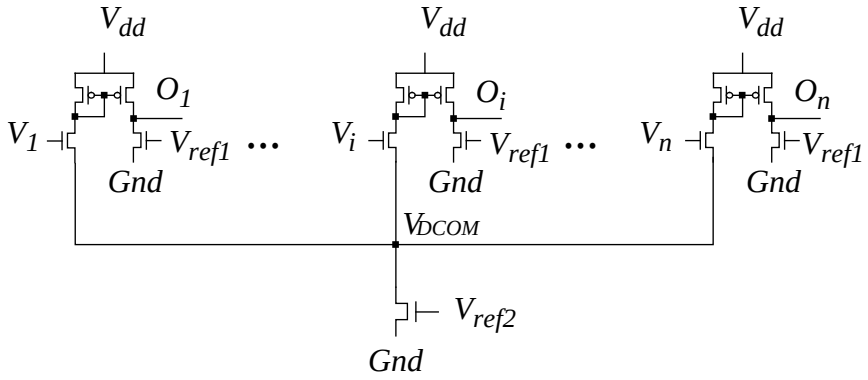

Figure 2 Analog multiple signal comparator

The sensitivity of this *N*-comparator, i.e. the minimum voltage difference among the input signals that may drive their corresponding outputs into logical *on* or *off*, correlates with the number of signals, i.e the value of *n*, due to the common current sink. SPICE simulation shows that for a single stage *4*-comparator, using standard 2μ CMOS technology and V_{dd}*=5V*, the sensitivity is about *0.7V*, but non-linear across the operating range (*0.86V* to *4.25V* in our simulation). In terms of quantized analog signals, this means that four different levels of voltage signals, each with a *±0.22V* safety margin, can be reliably compared. This

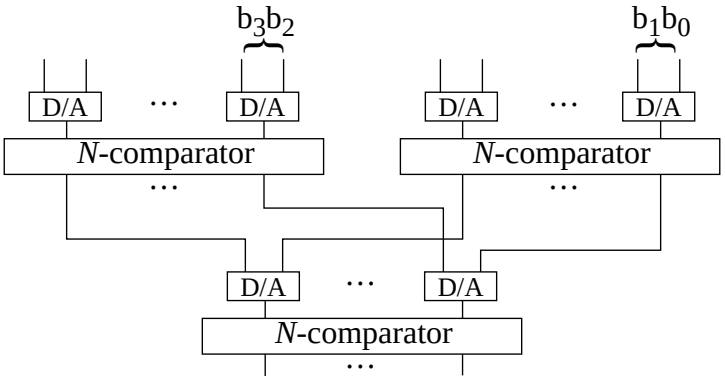

Figure 3 Cascading 2-bit comparators for 4 bit signals

corresponds to a two-bit comparator, with a D/A at each input line.

Whilst the sensitivity can be greatly improved by adding more stages to amplify the voltage difference or using bipolar technology, for true analog input signals, it is sufficient to show that, for digital signals, an analog comparator with the above sensitivity can always be cascaded to realize higher precision, as shown in **Figure 3**. This also means that higher precision digital values may be compared in a kind of serial fashion; and the comparison speed only depends on the number of bits in the comparands, rather than the number of the signals being compared simultaneously.

Digital *N*-Comparator

The digital *N*-comparator compares *n* digital values, simultaneously, bit by bit from the *msb* to *lsb*, and the maximum value will be identified by the success of a series of bit-wise comparisons. This can be realized by using a bus structure with *p*-MOSFETs to drive the bus high and a common current sink to shunt the bus low.

Starting from the *msb*, each signal puts a bit onto the corresponding bus line and, after some time delay for the bus to settle down, compares its bit value with that on the bus line. If they are the same, it continues to do the same with the next bit. Otherwise, it will stop putting its lower bits onto the bus and signal a failure in the competition. If a signal succeeds in every bit position, this signal contains a maximum value. **Figure 4** show the logic for a

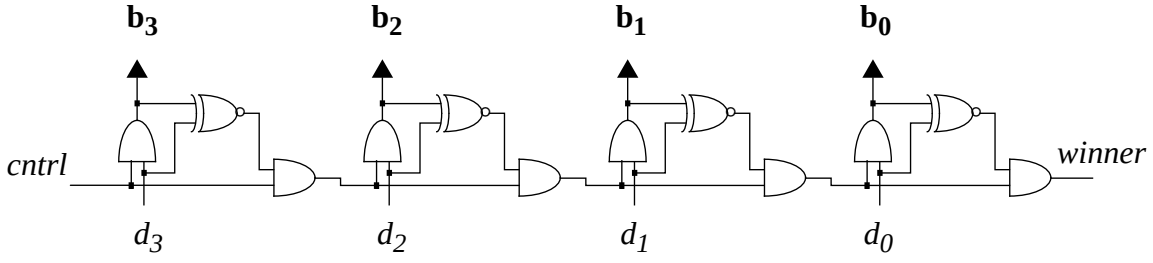

Figure 4 Control logic for one signal in a digital multiple signal comparator

signal of 4-bit value, where $\mathbf{b_3b_2b_1b_0}$ are connected to the bus lines; $d_3d_2d_1d_0$ represent the 4-bit digital input signal; the signal *cntrl* may prohibit the input signal from participating the competition; and the signal *winner*, if true, indicates this input has won the competition.

The advantage of this design is two-fold. Firstly, the value *n* can be large due to the digital bus structure. Secondly, cascading such comparators by connecting their buses together via switches would allow dynamic clustering at run time. The latter would facilitate programmable network configuration. The speed of comparison only depends on the

number of bits in the values being compared, three gates' delay plus a bus charge/discharge per bit, as may be seen in **Figure 4**. SILOS simulation under Cadence™ showed that an 8-bit *63*-Comparator can reliably complete the comparison in well under *40ns*. An added benefit is that the signal present on the bus is in fact the maximum value of the digital numbers being compared. This simplifies the detection of the winning value, which is required to signal the successfully converged network state.

Note. In the above description of the *N*-comparator, both analog and digital, the necessary control logic and status registers are omitted for the sake of simplicity. The omitted control logic includes: a) the circuit that detects the *nil* value of the winning input, indicating that no constraints were violated; b) the circuit that determines which of the winning input signals, in the case of a tie, is going to turn *on* its corresponding output state; and c) the circuit that detects a *no-change* in a convergence cycle, i.e. the selected neuron is already *on* in the previous cycle.

It should also be pointed out that, although the N-comparators as described above are designed to select the maximum value, similar techniques may be applied to design them for selecting the minimum value. It is also possible to compare positive values with negative ones, with a treatment of the sign. Since all the weights in GENET are negative, the N-comparator will actually be comparing the magnitude of negative values. This can be realized either by designing the comparator to select the minimum value, equally applicable to analog as well as digital design; or selecting the maximum 1's complement of the values, in the case of a digital design; or selecting the maximum voltage on reversely charged capacitors, in the case of an analog design. Therefore, we will ignore this problem in our discussions.

GENET Module

Having discussed the competition mechanism, we can now describe the building block of GENET architecture - the GENET module. The GENET module is based on the structure of a cluster of neurons. The architecture of the GENET module is shown inside the box in **Figure 5**. It consists of *n* neurons connected to an *N*-comparator. Each neuron has its own weight memory and a summation mechanism, the ellipses labelled $\sum$. The output signals of the *N*-comparator are latched into state registers, boxes labelled *S*, and fed back to the comparator for its control logic to determine the signal *no_change*. The module outputs the code of the active neuron in the cluster, and takes as input the codes of the active neurons in other clusters. The signal *no_violation* indicates the summation of the input to the winning neuron is nil, meaning no constraints have been violated. The signal *no_change* indicates that, in the current convergence cycle, the neuron selected to turn on was already on in the previous cycle. These two signals indicate the status of the cluster after a convergence cycle, e.g. whether this cluster may conform a part of a solution or a potential local minimum. If it is the latter, a learning cycle may be triggered, if necessary. The GENET module can be cascaded, by connecting the *link_control* signals, to extend the number of neurons per cluster without reducing the performance significantly.

A complete GENET network can be realized by organizing the GENET modules to suit the structure of the CSP to be solved. **Figure 6** shows an example of the overall architecture of GENET in application, where each row represents a cluster and a number of GENET modules may be cascaded for a large cluster. A chain of such GENET modules may be implemented on the same chip and dynamic configuration of the cluster size at run time is simply to program the switches. This is perfectly feasible with the digital comparator, but there might be some technical difficulty with the analog one.

For a realistic measurement of the GENET performance, let's assume that the weights

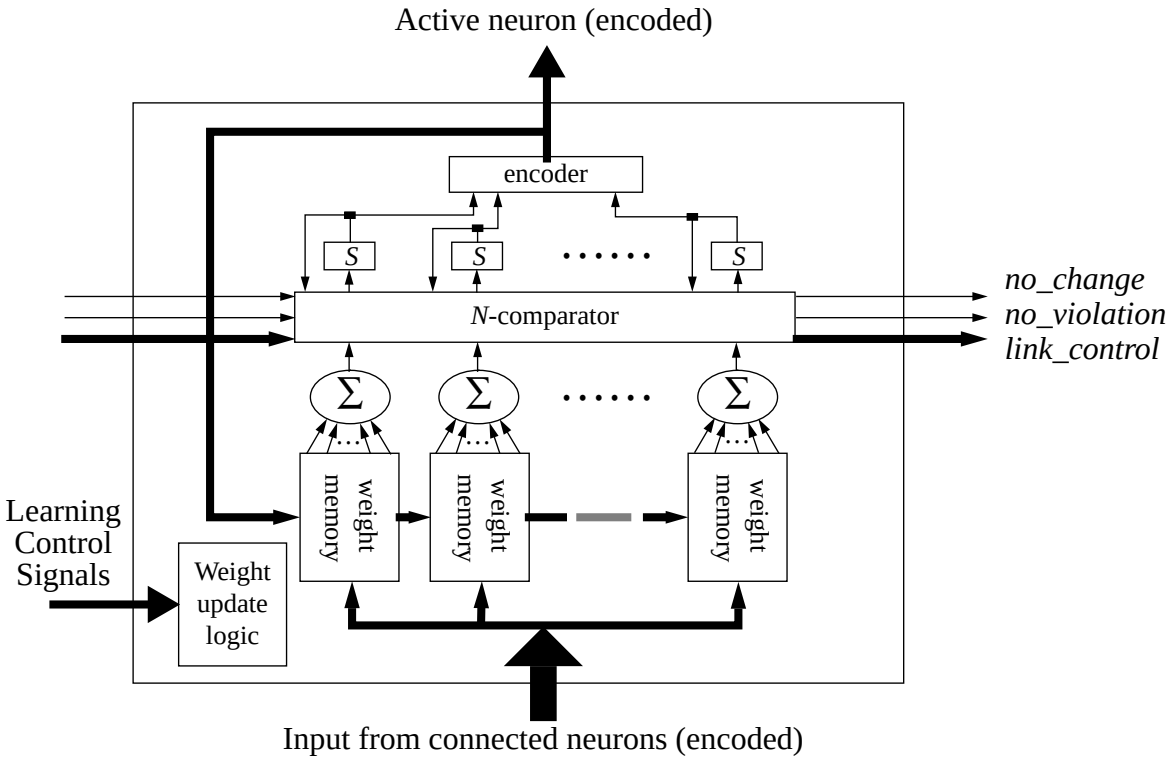

Figure 5 Architecture of a GENET module

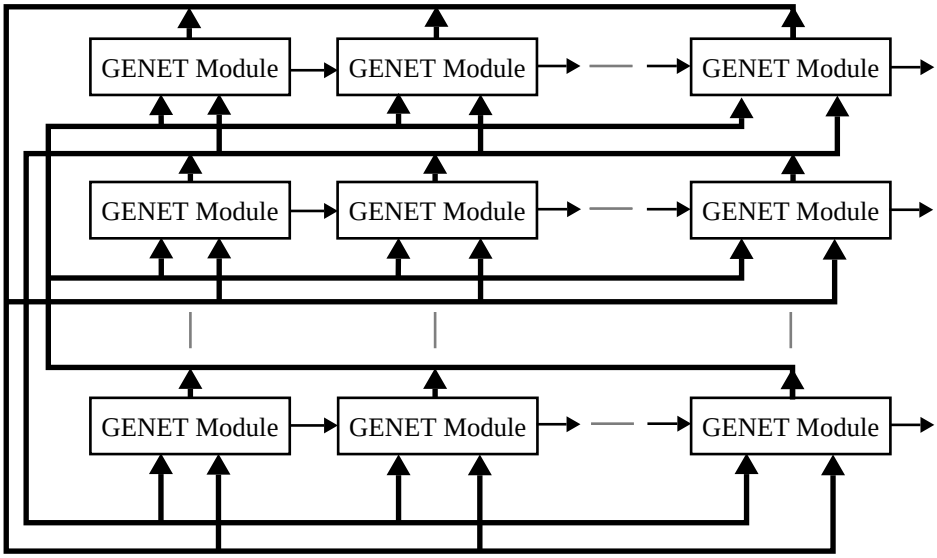

Figure 6 Overall architecture of GENET

are 8 bits and stored in off-chip fast SRAM, e.g. the 16K×4 bit INTEL C51C98-25 with 25ns access time. It means that the real memory access time can be made about 40ns, when address decoding overhead is included. Further, assume the digital comparator is integrated with a bit serial adder for the purpose of summing up the input signal to each neuron. Hence, only one input line is needed for one signal being compared. Take a CSP that has 200 variables for example, in a convergence cycle, each neuron will sum up 200 weights, because there is only one neuron in each cluster turned on. To sum up 200 weights, with each of 8 bits sequenced in, will need 200×8×40ns=64μs. To be more pragmatic, let's assume the 200 input signals are multiplexed via 20 lines. This means one convergence cycle would take 640μs. The competition time is ignorable in this case.

According to our simulation, the above problem can be solved in no more than 200 cycles, i.e. less then 200×640μs=128ms, while the FC-FFP heuristic program will take over 40 minutes of CPU time to solve the same problem. This means that the GENET approach may give a potential speed-up in the order of 10^5 against the heuristic search program. This estimation assumes a *full* parallelism of $N \times d$ neurons. In fact, even a parallelism of d would give a potential speed-up of the order of 10^3. This estimation overlooks other overheads that may occur in practice, such as updating weights in a learning phase, which may be twice the time of a convergence cycle. However, we may still conclude that the GENET, when implemented in VLSI technology with a moderate parallelism, e.g. tens of GENET modules each of tens of neurons which is highly feasible with the current VLSI technology, will out-perform the existing CSP solvers significantly.

CONCLUSIONS

Real life Constraint Satisfaction Problems are typically large and complex, and can not be satisfactorily solved by the current commercially available constraint programming languages and systems within a tolerable period of time. The competitive neural network model GENET has been developed to tackle these problems. The GENET approach provides an effective parallel algorithm for solving CSPs. When implemented in VLSI technology with a realistic, moderate parallelism, a significant speed-up over the existing constraint programming languages and systems is possible.

Acknowledgment

The authors would like to acknowledge the contribution of Kate W. C. Sin who investigated GENET behaviour with over-constrained CSPs, Nordin B. Salleh who simulated the analog design using SPICE simulator, and Hai Cheong Wong who simulated the digital design using SILOS simulator under Cadence™ EDGE®.

References

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

Dechter, R., Meiri, I. & Pearl, J., "Temporal constraint networks", Artificial Intelligence, 49, pp. 61-95, 1991.

Dincbas, M., Simonis, H. & Van Hentenryck, P., "Solving car sequencing problem in constraint logic programming", Proceedings, European Conference on AI, pp. 290-295, 1988.

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.

Graf, H. P. and Jackel, L.D., "Analog Electronic Neural Network Circuits", IEEE Circuits and Devices Magazine, pp. 44-55, July 1989.

Guesgen, H.W. & Hertzberg J., "A Perspective of Constraint-based Reasoning", Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992

Haralick, R.M. and Elliott, G.L., "Increasing tree search efficiency for constraint satisfaction problems", Artificial Intelligence 14, pp. 263-313, 1980.

Kasif, S., "On the parallel complexity of discrete relaxation in constraint satisfaction networks", Artificial Intelligence (45), pp. 275-286, 1990.

Lazzaro, J., Ryckebusch, S., Mahowald, M. A., and Mead, C. A., "Winner-Take-All Networks of O(N) Complexity", in *Advances in Neural Information Processing Systems I*, Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 1989, 703-711

Mackworth, A.K., "Consistency in networks or relations", Artificial Intelligence 8(1), pp. 99-118, 1977.

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), pp.17-24, 1990.

Morishita, T., Tamura, Y. and Otsuki, T., "A BiCMOS Analog Neural Network with Dynamically Updated Weights", IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, pp. 142-143, Feb. 1990.

Murray, A. F., "Pulse Arithmetic in VLSI Neural Networks", IEEE Micro Mag., pp. 64-74, Dec. 1989.

Prosser, P., "Distributed asynchronous scheduling", PhD Thesis, Department of Computer Science, University of Strathclyde, November 1990.

Swain, M.J. & Cooper, P.R., "Parallel hardware for constraint satisfaction", Proc. AAAI, pp. 682-686, 1988.

Tomberg, J. E. and Kaski, K. K. K., "Pulse-Density Modulation Technique in VLSI Implementations of Neural Network Algorithms", IEEE J. of Solid-State Circuits, vol. 25, no. 5, pp. 1277-1286, Oct. 1990.

Tsang, E.P.K., "The consistent labelling problem in temporal reasoning", Proc. AAAI Conference, Seattle, pp. 251-255, July 1987.

Tsang, E. P. K., & Wang, C. J., "A generic neural network approach for constraint satisfaction problems", Proc. NCM'91 Applications of Neural Networks, to be published in Series in Neural Networks by Springer Verlag, 1992.

Waltz, D.L., "Understanding line drawings of scenes with shadows", in WINSTON, P.H. (ed.) The Psychology of Computer Vision, McGraw-Hill, New York, pp. 19-91, 1975.

Wang, C. J., & Tsang, E. T. K., "Solving constraint satisfaction problems using neural networks", Proceedings, IEE Second International Conference on Artificial Neural Networks, pp. 295-299, 1991.

Yasunaga, M., Masuda, N., Yagyu, M., Asai, M., Yamada, M., and Masaki, A., "Design, Fabrication and Evaluation of a 5-Inch Wafer Scale Neural Network LSI Composed of 576 Digital Neurons", Proc. Int. Joint Conf. on Neural Networks, Vol. II, pp. 527-535, June 1990.