Solving CSP via Neural Network Which Can Update All Neurons Simultaneously

Takahiro Nakano and Masahiro Nagamatu

Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology
2-4 Hibikino, Wakamatu-ku, Kitakyushu 808-0196, Japan
e-mail:nakano-takahiro@edu.brain.kyutech.ac.jp

***Abstract*—The constraint satisfaction problem (CSP) is a combinatorial problem to find a solution which satisfies all given constraints. We proposed a neural network called LPPH to solve the satisfiability problem (SAT). The LPPH is not trapped by any point which is not a solution of the SAT. In this paper, we extend the LPPH for solving the CSP. Our neural network can update all neuron simultaneously. This is an advantage for the VLSI implementation. Experimental results show our proposal is effective.**

I. Introduction

The CSP (Constraint Satisfaction Problem) is a problem to find a variable assignment which satisfies all given constraints. Because the CSP has a well defined abstract formulation, it can represent various problems in the computer science. The decision problem whether the given CSP has a solution or not is an NP complete problem. There are two kinds of methods for solving the CSP, the complete search method and the incomplete search method. The complete search method can determine the inconsistency of the given problem, while the incomplete search method can not. If the given problem has solutions, the incomplete search method can find a solution quickly.

The tree search method [1] and the merge method [2] are the well used complete search methods. The tree search method recursively divides the original problem into partial problems, and finds a solution for each partial problem. The merge method merges constraints of the given problem, and at the end of search, it can find all solutions by unifying all constraints to one constraint. The MCHC (Min-Conflict Hill Climbing) [3] and the GENET [4] belong to the incomplete search method. The MCHC assigns a value to each variable randomly as an initial assignment, and iteratively changes the value of a selected variable to minimize the number of violations of constraints. The MCHC algorithm has the possibility of being trapped by local minima. To escape from local minima, the MCHC reassigns initial values. The GENET also changes value of selected variables so as to minimize the number of violations of constraints. When the GENET is trapped by local minima, it manages to escape from local minima by increasing the weights of constraints which is not satisfied at the time.

We proposed a neural network called LPPH for solving the SAT (SATisfiability problem) [5], [6], [7]. The SAT is a problem to find an assignment of truth values to variables which satisfies the given CNF (Conjunctive Normal Form). To use the LPPH for solving the SAT, at first we convert the SAT to an equivalent continuous valued problem called CONSAT (CONtinuous valued SAT). In this approach each clause has a weight, and if the clause is not satisfied, it increases its weight. The dynamics of the LPPH changes the landscape dynamically, and the trajectory of the LPPH is not trapped by any point which is not the solution of the CONSAT.

In this paper, we extend the LPPH called LPPH-CSP for solving the CSP. Compared with the SAT, the CSP can represent problem more compactly and effectively. In our experiment, we compare the LPPH-CSP with the LPPH and the GENET, and show the effectiveness of the LPPH-CSP. The LPPH-CSP can update all neurons simultaneously. This is an advantage for the VLSI implementation.

II. CSP

The CSP is a combinatorial problem to find a solution which satisfies all given constraints. The CSP is defined by a triple (X,D,C).

- $X=\{X_1,X_2,\ldots,X_n\}$ is a finite set of variables.
- $D=\{D_1,D_2,\ldots,D_n\}$ is a finite set of domains. Each domain D_i is a finite set of values and each variable X_i is assigned a value in D_i.
- $C=\{C_1,C_2,...,C_m\}$ is a finite set of constraints.

A solution of the CSP is a variable assignment to X which satisfies C.

Let x_{ij} be a Boolean variable which represents "variable X_i is assigned the jth value in D_i". x_{ij} is called a VVP (Variable-Value Pair). If x_{ij} is true (x_{ij}=1), variable X_i is assigned the jth value in D_i. If x_{ij} is false (x_{ij}=0), variable X_i is not assigned the jth value in D_i. Constraint C_r consists of a set of VVPs. In this paper, we consider the following types of constraints.

- ALT(n,S) [at-least-n-true constraint]
  S is a finite set of VVPs. The ALT constraint requires that at least n of VVPs in S must be true.
- ALF(n, S) [at-least-n-false constraint]
  The ALF constraint requires that at least n of VVPs in S must be false.

- AMT(n,S) [at-most-n-true constraint]
  The AMT constraint requires that at most n of VVPs in S must be true.
- AMF(n,S) [at-most-n-false constraint]
  The AMF constraint requires that at most n of VVPs in S must be false.

In the ordinary definition of the CSP, only binary constraints are included. The binary constraint requires "at least one of the given two VVPs is false", and is represented by ALF(1,S). By introducing the above four types of constraints, we can represent the given combinatorial problems more compactly. The N-coloring problem is the problem to paint all nodes of the given undirected graph by the given number of colors, not to color the adjacent nodes by the same color. Fig.1 (b) shows a CSP which represents a 3-coloring problem of an undirected graph shown in Fig.1 (a). In Fig.1 (b), ALT constraints and AMT constraints require that each node must be painted by just one color. ALF constraints require that adjacent nodes must not be painted by the same color.

III. GENET

Wand and Tsang [4] proposed a neural network called GENET for solving the CSP. In the GENET algorithm, one variable is iteratively selected and changes its value to minimize the number of violations of all constraints. If the GENET is trapped by a local minimum, it escapes from the local minimum by increasing the weights of constraints which are not satisfied at the time. In this section, we explain the GENET for the CSP which has only binary constraints, however the GENET can be applied to general constraints [8].

Neuron $<i,j>$ corresponds VVP x_{ij}. The input to the neuron $<i,j>$ is calculated by the following equation.

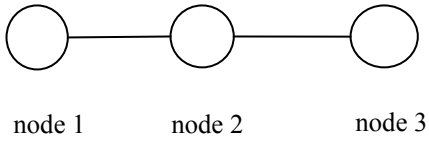
$$I_{<i,j>} = \sum_{<k,l>} W_{<i,j><k,l>} V_{<k,l>}. \qquad (1)$$

$W_{<i,j><k,l>}$ is a weight of connection between neuron $<i,j>$ and $<k,l>$. All weights will be initially set to 0. $V_{<k,l>}$ is the output of neuron $<k,l>$. $V_{<k,l>}$ has a discrete value 0 or 1. If $V_{<k,l>}= 1$, x_{kl} is true, and if $V_{<k,l>}= 0$, x_{kl} is false.

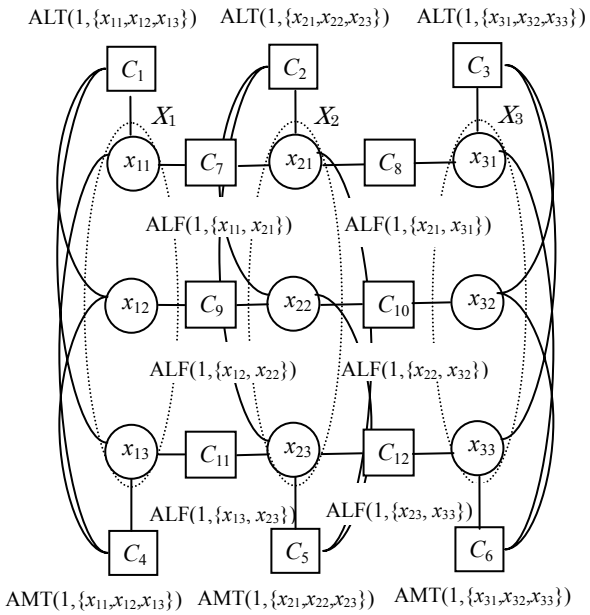
A. GENET Algorithm

The algorithm of the GENET is described as follows:

1. Select arbitrarily one neuron from each variable. Set the output of the selected neuron to 1, and set the outputs of the other neurons to 0.
2. For each variable:
   Calculate the inputs of neurons in the variable, and set the output of a neuron which has the largest input to 1. If tie break occurs, select arbitrarily one of them. Set the outputs of neurons to 0.
3. If none of neurons has changed the output in step2, then
   (a) If all constraints are satisfied, then a solution is found. Terminate the algorithm.
   (b) Otherwise, activate learning.
4. go to step2.



(a) An undirected graph



(b) A CSP which represents a 3-Coloring Problem

Fig.1 An example of 3-Coloring Problem

B. Learning

The GENET may be stuck in a local minimum, i.e. the state in which the GENET can not change its all outputs, though it has not found a solution of the CSP yet. For this reason, the GENET adjusts the weight of connection between neurons by the following equation.

$$W^{t+1}_{<i,j><k,l>} = W^{t}_{<i,j><k.l>} - V_{<i,j>} V_{<k,l>}. \qquad (2)$$

$W^{t}_{<i,j><k,l>}$ is the weight of connection between neuron $<i,j>$ and $<k,l>$ at time t. The cost of each violated constraint increases by learning, and the GENET can escape from the local minimum.

IV. LPPH-CSP

If the GENET updates all variables simultaneously, "this may cause the network to oscillate between a small number of states indefinitely [8]". Therefore, the GENET must update all variables sequentially. In this section, we propose methods

called LPPH-CSP(1) and LPPH-CSP(2) which can update all variables simultaneously.

Let VVP x_{ij} represent the degree of certainty that the variable X_i is assigned the jth value of D_i. x_{ij} has the continuous value between 0 and 1. The dynamics of the LPPH-CSP is defined as follows:

$$\begin{aligned} \frac{dx_{ij}}{dt} &= x_{ij}\left(1-x_{ij}\right)\sum_{r=1}^{m} w_r s_{rij}\left(\boldsymbol{x}\right), \\ \frac{dw_r}{dt} &= -\alpha w_r + h_r\left(\boldsymbol{x}\right), \end{aligned} \tag{3}$$

where $s_{rij}(\boldsymbol{x})$ represents a force put on x_{ij} for satisfying constraint C_r. w_r is the weight of constraint C_r. $h_r(\boldsymbol{x})$ represents the degree of unsatisfaction of constraint C_r. α is called the attenuation coefficient. In the LPPH-CSP, each variable changes its value so as to satisfy all constraints, and weight w_r increases, if constraint C_r is not satisfied. The LPPH-CSP searches a solution of the CSP by numerically solving the above dynamics. This dynamics is an extension of the dynamics of the LPPH for the SAT [5].

The important question is how to define functions $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ for the constraints introduced in 2. Now we will consider two methods to define these functions.

A. LPPH-CSP(1)

In the first method, the function $s_{rij}(\boldsymbol{x})$ is defined for each constraint C_r as follows:

$$s_{rij}\left(\boldsymbol{x}\right) = -\frac{\partial h_r\left(\boldsymbol{x}\right)}{\partial x_{ij}}. \tag{4}$$

The function $h_r(\boldsymbol{x})$ is defined as follows for ALT(1,S), ALF(1,S), AMT(1,S) and AMF(1,S).

- C_r=ALT(1,S)

For this type of constraint $h_r(\boldsymbol{x})$ is defined as follows:

$$h_r\left(\boldsymbol{x}\right) = \prod_{x_{ij}\in S}\left(1-x_{ij}\right). \tag{5}$$

From (4) we have

$$s_{rij}\left(\boldsymbol{x}\right) = \prod_{\substack{x_{kl}\in S \\ (k,l)\neq(i,j)}}\left(1-x_{kl}\right).$$

If any VVP in C_r other than x_{ij} has value 1, $s_{rij}(\boldsymbol{x}) = 0$, namely no force is put on x_{ij}. Otherwise, some force is put on x_{ij} so as to increase the value of x_{ij} .

- C_r=ALF(1, S)

For this type of constraint $h_r(\boldsymbol{x})$ is defined as follows:

$$h_r\left(\boldsymbol{x}\right) = \prod_{x_{ij}\in S} x_{ij}. \tag{6}$$

From (4) we have

$$s_{rij}\left(\boldsymbol{x}\right) = -\prod_{\substack{x_{kl}\in S \\ (k,l)\neq(i,j)}} x_{kl}.$$

If any VVP in C_r other than x_{ij} has value 0, $s_{rij}(\boldsymbol{x}) = 0$, namely no force is put on x_{ij}. Otherwise, some force is put on x_{ij} so as to decrease the value of x_{ij} .

- C_r=AMT(1, S)

For this type of constraint $h_r(\boldsymbol{x})$ is defined as follows:

$$h_r\left(\boldsymbol{x}\right) = \frac{\sum_{x_{ij}\in S}\sum_{\substack{x_{kl}\in S_r \\ (k,l)\neq(i,j)}} x_{ij}x_{kl}}{2_{n_r}C_2}, \tag{7}$$

where n_r is the number of VVPs in C_r. From (4) we have

$$s_{rij}\left(\boldsymbol{x}\right) = -\frac{\sum_{\substack{x_{kl}\in S_r \\ (k,l)\neq(i,j)}} x_{kl}}{{}_{n_r}C_2}.$$

If the sum of VVPs in C_r other than x_{ij} is equal to 0, $s_{rij}(\boldsymbol{x}) = 0$, namely no force is put on x_{ij}. Otherwise, some force is put on x_{ij} so as to decrease the value of x_{ij}.

- C_r=AMF(1, S)

For this type of constraint $h_r(\boldsymbol{x})$ is defined as follows:

$$h_r\left(\boldsymbol{x}\right) = \frac{\sum_{x_{ij}\in S}\sum_{\substack{x_{kl}\in S_r \\ (k,l)\neq(i,j)}}\left(1-x_{ij}\right)\left(1-x_{kl}\right)}{2_{n_r}C_2}. \tag{8}$$

From (4) we have

$$s_{rij}\left(\boldsymbol{x}\right) = \frac{\sum_{\substack{x_{kl}\in S_r \\ (k,l)\neq(i,j)}}\left(1-x_{kl}\right)}{{}_{n_r}C_2}.$$

If the sum of VVPs in C_r other than x_{ij} is equal to 0, $s_{rij}(\boldsymbol{x}) = 0$, namely no force is put on x_{ij}. Otherwise, some force is put on x_{ij} so as to decrease the value of x_{ij}.

In these definitions, we explained only about the case of n = 1 for ALT(n,S), ALF(n,S), AMT(n,S) and AMF(n,S). We can also define $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ for constraints with n > 1. However in these cases, the number of terms in the above functions becomes large. For example, the number of terms in $h_r(\boldsymbol{x})$ for ALT(n,S) becomes ${}_nC_2$.

B. LPPH-CSP(2)

- C_r=ALT(n, S)

For this type of constraint $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ are defined as follows:

$$\begin{aligned} h_r\left(\boldsymbol{x}\right) &= 1-\mathrm{NMax}\left(n,S\right), \\ s_{rij}\left(\boldsymbol{x}\right) &= \begin{cases} 1-\mathrm{NMax}\left(n+1,S\right), \\ \text{if } \ x_{ij} \geq \mathrm{NMax}\left(n,S\right), \\ h_r\left(\boldsymbol{x}\right), \\ \text{otherwise}, \end{cases} \end{aligned} \tag{9}$$

where $\mathrm{NMax}(n,S) = n$th maximum value in S.

- C_r=ALF(n, S)

For this type of constraint $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ are defined as follows:

$$h_r(\boldsymbol{x}) = \mathrm{NMin}(n, S),$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} -\mathrm{NMin}(n+1, S), \\ \text{if} \quad x_{ij} \le h_r(x), \\ -h_r(\boldsymbol{x}), \\ \text{otherwise}, \end{cases} \tag{10}$$

where $\mathrm{NMin}(n, S) = n\text{th}$ minimum value in S.

- C_r=AMT(n, S)

For this type of constraint $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ are defined as follows:

$$h_r(\boldsymbol{x}) = \mathrm{NMax}(n+1, S),$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} -\mathrm{NMax}(n, S), \\ \text{if} \quad x_{ij} \le h_r(x), \\ -h_r(\boldsymbol{x}), \\ \text{otherwise}. \end{cases} \tag{11}$$

- C_r=AMF(n, S)

For this type of constraint $h_r(\boldsymbol{x})$ and $s_{rij}(\boldsymbol{x})$ are defined as follows:

$$h_r(\boldsymbol{x}) = 1 - \mathrm{NMin}(n+1, S),$$

$$s_{rij}(\boldsymbol{x}) = \begin{cases} 1\text{-}\mathrm{NMin}(n, S), \\ \text{if} \quad x_{ij} \ge \mathrm{NMin}(n+1, S), \\ h_r(\boldsymbol{x}), \\ \text{otherwise}. \end{cases} \tag{12}$$

V. Experiment

A. Comparison of proposed methods

We compared the LPPH-CSP(1), the LPPH-CSP(2) and the LPPH. For the LPPH, the given CSP is converted to the SAT at first, and the LPPH is applied to the SAT. Table 1 shows the average CPU time for the N-Queen Problems. The average is calculated by changing initial points 30 times. For each try, the CPU time is limited to 300 sec. Table 1 shows only LPPH-CSP(2) can find a solution efficiently.

Table 1 CPU time(sec) for N-Queen problems

problem	LPPH	LPPH-CSP(1)	LPPH-CSP(2)
10-Queen	0.08	0.04	0.01
20-Queen	3.43	6.52	0.04
30-Queen	20.78	300	0.10
40-Queen	64.72	300	0.30
50-Queen	115.05	300	0.44

B. Attenuation coefficient

From experiments of the LPPH for the SAT, we know that the CPU time depends the value of attenuation coefficient. Fig.2, Fig.3, and Fig.4 show how the value of attenuation coefficient α influences the CPU time of the LPPH-CSP(2). Horizontal axes indicate the value of attenuation coefficient α, and vertical axes indicate the average CPU time. Fig.2, Fig.3, and Fig.4 show results for the Car Sequencing Problems, Random CSPs and the N-Queen Problems, respectively. For the Car Sequencing Problems, Benchmark problems of CSPLib(http://4c.ucc.ie/~tw/csplib/) are used. Random CSPs are generated by randomly adding binary constraints. For Random CSP, the number of variables is 20, the number of values for each variables is 10, and the number of clauses is 3800. From these experimental results, we can see the value about 0.1 is the best for the attenuation coefficient α for these problems.

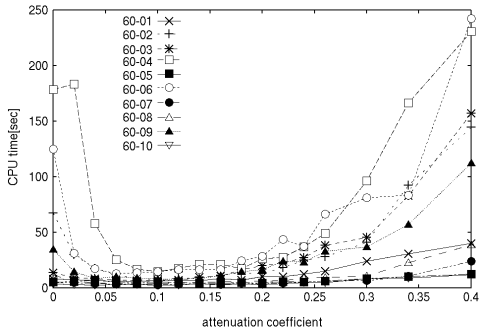


Fig.2 CPU time vs. attenuation coefficient for Car Sequencing Problems

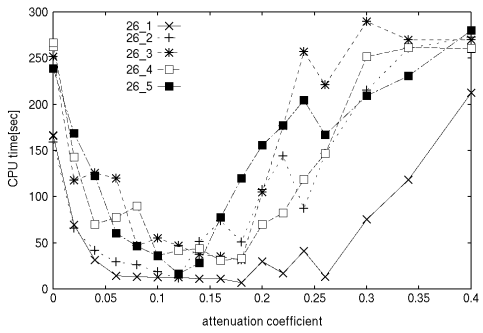


Fig.3 CPU time vs. attenuation coefficient for Rondom CSPs

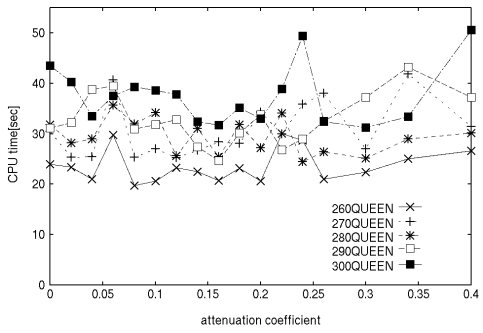


Fig.4 CPU time vs. attenuation coefficient for N-Queen Problems

C. Comparison with GENET

To investigate the efficiency of the LPPH-CSP(2), we compared the GENET and the LPPH-CSP(2). Fig.5, Fig.6, and Fig.7 show the results for the Car Sequencing Problems, Random CSPs and the N-Queen Problems, respectively. Random CSPs are generated by randomly adding AMT constraints. For Random CSP, the number of variables is 100, the number of values for each variables is 40, and the number of clauses is 1200. For the N-Queen Problems, the LPPH-CSP(2) is faster than the GENET. However, for the Car Sequencing Problems and Random CSPs, the GENET is faster than the LPPH-CSP(2).

VI. CONCLUSION

In this paper, we proposed two neural networks called LPPH-CSP(1) and LPPH-CSP(2). The LPPH-CSP(1) is a naive extension of the LPPH for the SAT, and the LPPH-CSP(2) is defined by using NMAX or NMIN functions. From experimental results, the LPPH-CSP(2) is extremely faster than LPPH-CSP(1) and the LPPH. Experiments in which the LPPH-CSP is compared with the GENET show that for some problems the LPPH-CSP is faster than GENET, while for other problems the GENET is faster than the LPPH-CSP. However, generally speaking, we can say the LPPH-CSP(2) is as efficient as the GENET. The GENET must update all variables sequentially, while the LPPH-CSP can update all variables simultaneously. We think this is an advantage of the LPPH-CSP(2) for the VLSI implementation.

This study will be followed by applying the LPPH-CSP(2) for extended problems of the CSP such as Max-CSP. We also need to incorporate other new types of constraints which are useful to represent practical problems into the LPPH-CSP(2).

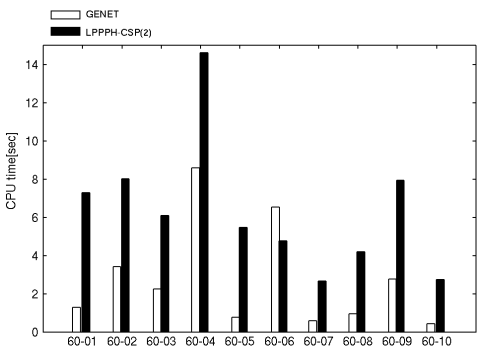


Fig.5 Comparison with GENET for Car Sequencing Problems

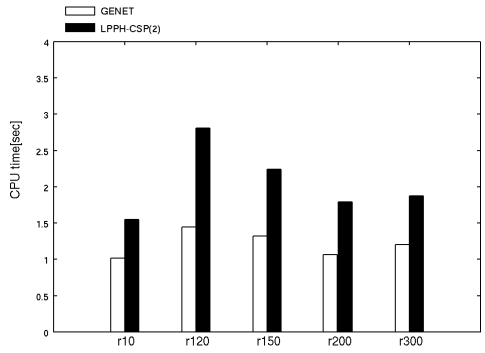


Fig.6 Comparison with GENET for Random CSPs

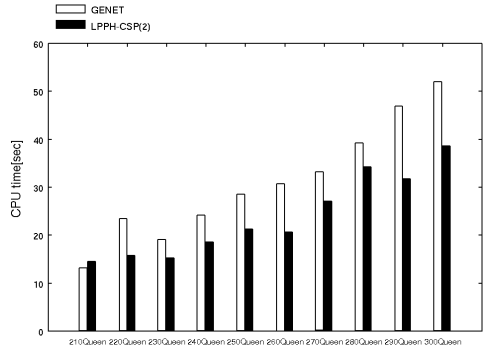


Fig.7 Comparison with GENET for N-Queen Problems

REFERENCES

[1] R. M. Haralick and G. L. Elliot, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", *Artificial Intelligence*, Vol.14, 263-313. 1980.

[2] S. Nishihara, "Consistent Labeling Problems with Applications", *Jornal of IPSHJ*, Vol.31, No.4, 500-507, 1990, in Japanese.

[3] S. Minton, P. Laird, M. D. Johnston and A. B. Philips, "Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems", *Artificial Intelligence*, Vol.58, 161-205, 1992.

[4] C. J. Wang, and E. Tsang, "Solving Constraint Satisfaction Problems Using Neural Networks", *In Proceedings of the IEE 2nd Conference on Artificial Neural Networks*, 295-299, 1991.

[5] M. Nagamatu and T. Yanaru, "On the Stability of Lagrange Programming Neural Networks for Satisfiability Problems of Propositional Calculus", *Neurocomputing*, 13, 119-133, 1995.

[6] M. Nagamatu and T. Yanaru, "Lagrangian Method for Satisfiabiability Problems of Propositional Calculus" , *The Second NewZealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, November*, 20-23, 1995.

[7] M. Nagamatu, T. Nakano, N. Hamada, T. Kido and T. Akahosi, "Extensions of Lagrange Programming Neural Network for Satisfiability Problem and its Several Variations", *In Proceedings of 9th International Conference on Neural Information Processing*, 1781-1785, 2002.

[8] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu, "GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement", *National Conference on Artificial*