NAIS: A Calibrated Immune Inspired Algorithm to solve Binary Constraint Satisfaction Problems*

Marcos Zuñiga[1], María-Cristina Riff[2] and Elizabeth Montero[2]

[1] Projet ORION, INRIA Sophia-Antipolis
Nice, France
e-mail: Marcos.Zuniga@sophia.inria.fr
[2] Department of Computer Science, Universidad Técnica Federico Santa María,
Valparaíso, Chile,
e-mail:María-Cristina.Riff@inf.utfsm.cl, Elizabeth.Montero@inf.utfsm.cl

Abstract We propose in this paper an artificial immune system to solve CSPs. The algorithm has been designed following the framework proposed by de Castro and Timmis. We have calibrated our algorithm using Relevance Estimation and Value Calibration (REVAC), that is a new technique, recently introduced to find the parameter values for evolutionary algorithms. The tests were carried out using random generated binary constraint satisfaction problems on the transition phase where are the hardest problems. The algorithm shown to be able to find quickly good quality solutions.

1 Introduction

Constraint satisfaction problems (CSPs) involve finding values for problem variables subject to constraints on which combinations are acceptable. Over the few years, many algorithms and heuristics were developed to find a solution of CSPs. Following these trends from the constraint research community in the bio-inspired computation community, some approaches have also been proposed to tackle CSP with success such that evolutionary algorithms [4], [8], [10], [12], [11], [14], ants algorithms [13]. Given that recent publications indicate that artificial immune systems offer advantages in solving complex problems [1], [3], our goal here is to propose an efficient immune inspired algorithm which can solve CSPs. Immune artificial systems as well as evolutionary algorithms are very sensitive to the values of their parameters. Garret in [18] proposed a parameter-free clonal selection using adaptive changes.
In this paper, we focalize our attention in a new method proposed for tuning, that is a method that uses statistical properties to determine the best set of parameter values for an evoltuionary algorithm.
The contributions of this paper are:

* Supported by Fondecyt Project 1060377

– An immune inspired algorithm which can solve hard CSPs,
– A new application of the tuning method Relevance Estimation and Value Calibration (REVAC) proposed for evolutionary algorithms, [15],

The paper is structured as follows. In the next section, we define the Constraint Satisfaction Problem. In section 3 we introduce our new approach NAIS. The results of tests and a comparison with other incomplete method are given in section 4. In our summary, we give some conclusions and future works.

2 Binary Constraint Satisfaction Problems

For simplicity we restrict our attention here to binary CSPs, where the constraints involve two variables. Binary constraints are binary relations. If a variable i has a domain of potential values D_i and a variable j has a domain of potential values D_j, the constraint on i and j, R_{ij}, is a subset of the Cartesian product of D_i and D_j. A pair of values (a, b) is called consistent, if (a, b) satisfies the constraint R_{ij} between i and j. The variables i and j are the *relevant* variables for R_{ij}. The constraint network is composed of the variables, the domains and the constraints. Thus, the problem is [9], [12] given a set of variables, a domain of possible values for each variable, and a conjunction of constraints, to find a consistent assignment of values to the variables so that all the constraints are satisfied simultaneously. CSP's are, in general, NP-complete problems and some are NP-hard [7]. Thus, a general algorithm designed to solve any CSP will necessarily require exponential time in problem size in the worst case.

3 NAIS: Network Artificial Immune System

We called our algoritm NAIS which stands for Network Artificial Immune System. The algorithm uses three immune components: antigen, antibody and B-cells. Basically, the antigen represents the information for each variable given by the constraint graph. This information is related to the number of connections of each variable, that is the number of constraints where each variable is a relevant one. Thus, it is a fixed information and not depends on the state of the search of the algorithm. On the contrary, the antibody strongly depends on the state of the search of the algorithm. It has two kinds of information: the variable values and the constraints violated under this instantiation. Finally, a B-cell has all the antibody information required by the algorithm to its evolution.

3.1 Immune Components for CSP

The immune components in our approach are defined as follows:

Definition 1. *(Antigen)*
For a CSP and its constraint graph we define the antigen Ag of the n-tuple of variables $(Ag_1, \ldots, Ag_n)$*, such that the* Ag_i *value is the number of constraints where* X_i *is a relevant variable,* $\forall i, i = 1, \ldots, n$.

The algorithm needs to know for each pre-solution its variable values and the constraints satisfied under this instantiation. For this reason, the antibody has two segments: a structural and a conflicting segment.

Definition 2. *(Structural antibody)*
A structural antibody Ab_s is a mapping from a n-tuple of variables $(X_1, \ldots, X_n) \rightarrow D_1 \times \ldots \times D_n$, such that it assigns a value from its domain to each variable in V. Remark: The structural segment corresponds to an instantiation $\mathbf{I}$ *of the CSP.*

Definition 3. *(Conflicting antibody)*
For a CSP and its constraint graph we define the conflicting antibody Ab_c of the n-tuple of variables $(Ab_{c_1}, \ldots, Ab_{c_n})$, such that the Ab_{c_i} value is the number of violated constraints where X_i is a relevant variable, $\forall i, i = 1, \ldots, n$.

A solution consists of a structural antibody which does not violate any constraint, that is, whose conflicting antibody complements the antigen.
Before defining the B-cell we need to introduce the idea of affinity in the context of our problem.

3.2 Affinity measure

In our approach we are interested in two kinds of affinity. The affinity between the antigen and a conflicting antibody, and the affinity between two structural antibodies.

- Affinity between the antigen and a conflicting antibody
  It is an estimation of how far an antibody is from being a CSP solution. It is related to the number of satisfied constraints by an antibody. The key idea is that a solution of the CSP corresponds to the biggest value of the affinity function between Ab_c and Ag. This occurs when all the constraints are satisfied. We define the function A_{csp} to measure this affinity as the euclidean distance computed by:

$$\mathbf{A_{csp}(Ag, Ab_c)} = \sqrt{\sum_{\mathbf{i=1}}^{\mathbf{n}} (\mathbf{Ag_i} - \mathbf{Ab_{c_i}})^{\mathbf{2}}} \tag{1}$$

  The function A_{csp} prefers a pre-solution with a minimal number of violated constraints as it is usual for guiding the search of incomplete techniques.

- Affinity between two structural antibodies
  Two structural antibodies has a high affinity level when they are quite similar in terms of the values of these variables. The idea of using this measure, named HA_s, is to quantify how similar two pre-solutions are. To compute this interaction our algorithm uses the Hamming distance. The algorithm uses this measure to obtain a diversity of pre-solutions.

3.3 B-cell Representation

A B-cell is a structure with the following components:

- An Antibody $Ab = (Ab_c, Ab_s)$
- The number of clones of A_b to be generated for the clonal expansion procedure. This number is directly proportional to the A_{csp} value.
- The hypermutation ratio used in the affinity maturation step. This ratio is inversely proportional to the A_{csp} value.

3.4 The Algorithm - Network Artificial Immune System

The algorithm NAIS is shown in figure 1. NAIS works with a set of B-cells, following an iterative maturation process. Some of these B-cells are selected, doing a *clonal selection*, prefering those with bigger affinity values A_{csp}, that is, those that satisfy a greater number of constraints. It uses a Roulette Wheel selection, [17]. The algorithm generates a number of clones of the B-cells selected, that is done by the *clonal expansion* procedure. These clones follow a hypermutation process in the *affinity maturation* step.

The new set of B-cells is composed of a selected set of hypermutated B-cells.

Algorithm NAIS(**CSP**) **returns** memory B-cells
Begin
  $Ag \leftarrow$ Determine constraint graph connections(**CSP**, n);
  Initialize B-cells
  For i $\leftarrow 1$ **to** $B - cells\_NUM$ **do**
    Compute affinity value A_{csp}(B-cells[**i**])
  End For
  j $\leftarrow 1$;
  While (**j** $\leq MAX\_ITER$) **or** (not solution) **do**
    Select a set of B-cells by Roulette Wheel
    Generate Clones of the selected B-cells
    Hypermutate Clones
    For k $\leftarrow 1$ **to** $CLONES\_NUM$ **do**
      Compute affinity value A_{csp}(Clones[**k**])
    End For
    B-cells $\leftarrow$ **build_network**(CLONES);
    B-cells $\leftarrow$ **metadynamics**(B-cells);
  End While
  Return B-cells;
End

Figure1. NAIS Pseudocode

This selection is done in the *build_network* using the HA_s values in order to have a diversity of B-cells. A hypermutated B-cell could belong to the new set of

B-cells if the hypermutated B-cell is quite different from the B-cells in memory. This difference is measured using the hamming distance, controlling the minimal degree of required diversity by the ϵ parameter value. Thus, a B-cell be accepted to be a new memory member when $(1 - \frac{HA_s}{n}) > \epsilon$. The ϵ value is known as the threshold of cross reactivity.
Finally, the algorithm adds new B-cells randomly generated in the *metadynamics* procedure to this set of B-cells. This procedure allows the algorithm to do more exploration of the search space.

Hypermutation procedure: The hypermutation is a hill-climbing procedure that repairs the conflicts in the conflicting antibody. This procedure is inspired on min-conflicts algorithm proposed in [16]. Figure 2 shows the hypermutation procedure.

```
Hypermutation(B-cell)
Begin
Repeat
V = Select randomly a variable to be changed
If Abc(V) > 0 then
        Repeat
              Choose v a new value for V from its domain
              NAbc(V) = Number of conflicts for V using v
              If NAbc(V) < Abc(V) then
                Abs(V) = v
                Re-compute Abc
              End If
        Until (NAbc(V) < Abc(V)) or (Max_tries)
End If
Until Num_hyper
returns(B-cell)
End
```

Figure2. Hypermutation Procedure

Given a B-cell, a variable of its structural antibody is randomly selected to be changed. In case of the selected variable does not participate in any constraint violation (i.e. $Ab_c(V) = 0$), the procedure tries to modify another variable that could be involved in a conflict. The value of this variable is modified, such that, this value allows to reduce the number of conflicts recorded on the corresponding conflicting antibody. Thus, this procedure changes the structural antibody and also, as a consequence, the conflicting antibody.
The procedure Re-compute(Ab_c) does a partial evaluation of the conflicting antibody, only considering the variables related to the variable V. That is, just those variables which are relevants with V for a specific constraint. This kind of partial evaluation is quite useful in the constraint research community in order

to reduce the computational time spent evaluating the constraints satisfaction for a new instantiation of the variables.
The hypermutation procedure uses two parameters: Max_tries and Num_iter. The parameter Max_tries corresponds to the maximum number of values to be tried for a given variable V. The parameter Num_iter corresponds to the maximum number of the B-cell variables that could be mutated.

4 Tests

The goal of the following benchmarks is to evaluate the performance of NAIS for solving CSP. The algorithm has been tested with randomly generated binary CSPs, [5]. The tests are to evaluate the NAIS behaviour when it is calibrated using the technique REVAC for tuning. We compare NAIS calibrated with GSA [4] that is a sophisticated evolutionary algorithm that solves CSPs and which strongly uses knowledge coming from the constraints research community. We also compare NAIS with SAW, [14]. SAW has been compared with both complete and incomplete well-known algorithms for CSP obtaining better results in most of the cases tested.
The hardware platform for the experiments was a PC Pentium IV Dual Core, 3.4Ghz with 512 MB RAM under the Mandriva 2006 operating system. The algorithm has been implemented in C. The code for NAIS is available in a web page[1].

4.1 Problems tested on the hard zone

The idea of these tests is to study the behavior of the algorithm solving hard problems. We use two models to generate binary CSPs. That is because GSA has been reported using model B proposed in [5] and SAW has been reported using model E, [14].

Model B: The binary CSPs belonging to the hard zone are randomly generated using the model proposed by B. Smith in [5]. This model considers four parameters to obtain a CSP. That is, the number of variables (n), the domain size for each variable (m), the probability that exists a constraint between two variables (p_1), and the probability of compatibility values(p_2). This model exactly determines the number of constraints and the number of consistent instantiations for the variables that are relevant for a given constraint. Thus, for each set of problems randomly generated the number of constraints are $\frac{p_1 n(n-1)}{2}$ and for a given constraint the number of consistent instantiations are $m^2 p_2$. Given (n, m, p_1) B. Smith defines a function to compute critical p_2 values, those values that allow to obtain CSPs on the transition phase, that is the problems that are harder to be solved.

[1] http://www-sop.inria.fr/orion/personnel/Marcos.Zuniga/CSPsolver.zip

$$\hat{\mathbf{p}}\mathbf{2}_{\text{crit}}(\mathbf{n}, \mathbf{m}, \mathbf{p_1}) = \mathbf{m}^{-\frac{\mathbf{2}}{(\mathbf{n}-\mathbf{1})\mathbf{p_1}}} \tag{2}$$

Model E: This model also considers four parameters (n, m, p, k). The parameters n and m have the same interpretation than in model B. For binary CSPs whose constraints have two relevant variable $k = 2$ in model E. The higher p the more difficult, on average, problem instances will be.

4.2 REVAC

The Relevance Estimation and Value Calibration has recently been proposed in [15]. The goal of this algorithm is to determine the parameter values for evolutionary algorithms. It is also able to identify which operators are not relevant for the algorithm. Roughly speaking, REVAC is a steady-state evolutionary algorithm that uses a real-value representation, where each value corresponds to a parameter value of the algorithm to be tunned. Each chromosome in REVAC is evaluated by the performance obtained by the algorithm (to be tunned) using its parameter values. A new individual is randomly created, however the value for each variable is obtained only considering the values that are in the population for this variable. It does 1000 evaluations.
In order to apply REVAC for calibrating NAIS, we have selected 14 problems, two from each category $< 10, 10, p_1, p_2 >$ using model B. The performance for each chromosome is computed as the number of satisfied constraints by the solution obtained by NAIS using these parameter values. The parameter values found by this tuning procedure were:

- $n_1 = 0.3$, rate of cells to be expanded,
- $n_2 = 0.4$, rate of cells to be incorporated on the memory
- $\epsilon = 0.40$, threshold reactivity between clones
- $B - cells = 5$
- Number of clones = 100

That means that NAIS requires to do more exploration than it does using a hand-made calibration. In the hand-made calibration the hypermutated cell is accepted if it differs at least in a 54% ($\epsilon = 0.46$) from the memory cells. Now, it must differ at least in a 60% to be accepted. Furthermore, the number of cells to be expanded has been reduced in 0.2. The procedure required around 14 hours, computational time, to determine these parameter values.

4.3 Tests with Calibration

Comparison between NAIS and GSA using Model B: Because the reported results of GSA, [4] has been evaluated with the problems in the hardest zone as they have been generated in [5], we run the calibrated NAIS using problems generated using the parameters (n, m, p_1, p_2). We consider the problems

with $n = m$, where $n = 10$. The p_1 and p_2 values are those belonging to the hardest zone. In figure 3 $c0.3\_t0.7$ means $p_1 = 0.3$ and $p_2 = 0.7$. The following table shows the percentage of problems solved and the time required for GSA and those obtained by NAIS considering 10.000, 50.000 and 75.000 evaluations.

Category	GSA		NAIS					
	50.000 ev.	time [s]	10.000 ev.	time [s]	50.000 ev.	time [s]	75.000 ev.	time [s]
c0.3_t0.7	93.33	2.18	87.5	0.68	93.75	1.85	97.5	1.98
c0.5_t0.5	84.4	2.82	91	0.61	98.33	1.08	99.33	0.98
c0.5_t0.7	100	2.76	84	0.61	83.33	2.79	84.33	3.91
c0.7_t0.5	16.7	10.35	87	0.77	87	2.09	90.67	3.34
c0.7_t0.7	100	2.51	80.67	0.66	80.33	2.15	82	4.52
c0.9_t0.5	3.3	13.77	83.67	0.52	86.33	2.15	87.33	4.19
c0.9_t0.7	99.0	1.58	75.33	0.82	75.33	3.92	75.67	6.03

Figure3. Success rate and CPU time for NAIS and GSA

NAIS has a higher satisfaction rate than GSA, moreover it converges very quickly to good solutions. Furthermore, considering just 10.000 evaluations the average succes rate for NAIS was around 83% instead of 72% for GSA. However, in some categories GSA outperforms NAIS.

Comparison between NAIS and SAW using Model E: We have generated 250 problem instances in the hard zone using Model E. Figure 4 shows the success rate and the time in seconds for both algorithms.

p	SAW		NAIS					
	100000 ev.		10000 it.		50000 it.		75000 it.	
	succes	time [s]	succes	time [s]	succes	time [s]	succes	time [s]
0.24	100	0.74	100	0.32	100	0.33	100	0.3
0.25	100	2.33	100	0.44	100	0.43	100	0.4
0.26	97	6.83	100	0.56	100	0.6	100	0.61
0.27	60	11.39	100	1.2	100	1.1	100	1
0.28	25	18.66	98.4	2.06	100	2.26	100	2.05
0.29	17	20.57	84	4.11	99.6	5.24	100	5.41
0.3	5	22.27	47.6	6.99	84.4	17.17	90	21.02
0.31	1	22.47	16.8	8.62	38.4	35.1	46.8	48.91
0.32	0	22.39	24	8.22	59.6	27.64	63.6	37.95
0.33	1	22.38	24.4	8.3	59.6	27.4	68.4	34.85

Figure4. Success rate and CPU time for NAIS and SAW

We can observe that NAIS outperforms SAW in both time and success rate. Moreover, the average success rate for SAW is 40.6% instead of a 69.2% for NAIS, just considering 10.000 iterations. NAIS required, for these number of iterations, in average, just 4.1 seconds.

Figure 5 shows the results for NAIS and SAW. We can observe in NAIS the transition phase in $p = 0.31$.

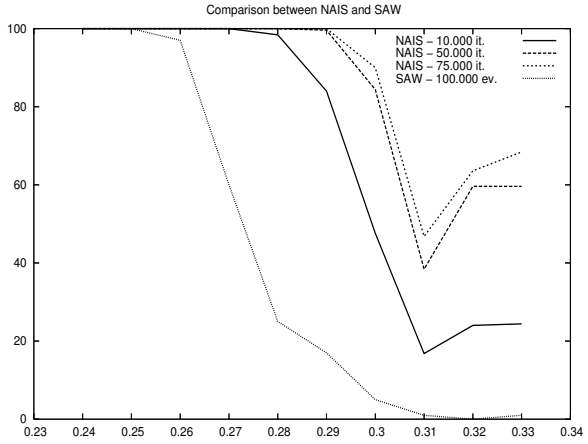

Figure5. Different problems tested, comparison of % Succesful runs

5 Conclusions

The artificial immune systems have some interesting characteristics from the computational point of view: pattern recognition, affinity evaluation, immune networks and diversity. All of these characteristics have been included in our algorithm. The B-cell structure is useful to determine both the solution of the problems and also to identify conflicts. The conflicting antibody is used by the algorithm to guide the reparation of the solutions (hypermutation process), giving more priority to the variables involved in a higher number of conflicts. For the problems in the hardest zone NAIS just using 10.000 iterations (avg. 4.1 seconds) solved, on average, 28% more problems than SAW, one of the best known evolutionary algorithm. The calibrated NAIS solved more problems than GSA, that is a sophisticated genetic algorithm which incorporated many constraints concepts to solve CSP. Artificial Immune Systems is a promising technique to solve constrained combinatorial problems.

6 Future Work

A promising research area is to incorporate some parameter control strategies into the algorithm. The tuning process to define the parameter values for NAIS has been a time consuming task.

References

[1] de Castro L.N. and Timmis J., Artificial Immune Systems: A New Computational Intelligence Approach, Ed. Springer, 2002.

[2] de Castro L.N. and von Zuben F.J., Learning and Optimization Using the Clonal Selection Principle, IEEE Transactions On Evolutionary Computing, Vol. 6, No. 3, pp. 239-251, Junio 2002.

[3] Dasgupta D., Artificial Immune Systems And Their Applications, Springer-Verlag, 2000.

[4] Dozier G., Bowen J. and Homaifar A., Solving Constraint Satisfaction Problems Using Hybrid Evolutionary Search, IEEE Transactions on Evolutionary Computation, Vol. 2, No. 1, pp. 23-33, Abril 1998.

[5] Smith B. and M. E. Dyer M.E., Locating the phase transition in constraint satisfaction problems, Artificial Intelligence, 81, pp. 155-181, 1996.

[6] Timmis J. and Neal M., Investigating the Evolution and Stability of a Resource Limited Artificial Immune System, Proceedings of the IEEE Brazilian Symposium on Artificial Neural Networks, pp. 84-89, 2000.

[7] Cheeseman P.,Kanefsky B. and Taylor W., Where the Really Hard Problems Are. Proceedings of IJCAI-91, pp. 163-169, 1991.

[8] Eiben A.E., van Hemert J.I., Marchiori E. and Steenbeek A.G., Solving Binary Constraint Satisfaction Problems using Evolutionary Algorithms with an Adaptive Fitness Function. Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN-V), LNCS 1498, pp. 196-205, 1998.

[9] Mackworth A.K., Consistency in network of relations. Artificial Intelligence, 8:99-118, 1977.

[10] Marchiori E., Combining Constraint Processing and Genetic Algorithms for Constraint Satisfaction Problems. Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97), 1997.

[11] Riff M.-C., A network-based adaptive evolutionary algorithm for CSP, In the book "Metaheuristics: Advances and Trends in Local Search Paradigms for Optimisation", Kluwer Academic Publisher, Chapter 22, pp. 325-339, 1998.

[12] Tsang, E.P.K., Wang, C.J., Davenport, A., Voudouris, C. and Lau,T.L., A family of stochastic methods for constraint satisfaction and optimization, Proceedings of the 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP), London, pp. 359-383, 1999.

[13] Solnon C., Ants can solve Constraint Satisfaction Problems, IEEE Transactions on Evolutionary Computation, 6(4), pages 347-357, 2002.

[14] Craenen B., Eiben A.E., and van Hemert J.I., Comparing Evolutionary Algorithms on Binary Constraint Satisfaction Problems, IEEE Transactions on Evolutionary Computation, 7(5):424-444, 2003.

[15] Nannen V. and Eiben A.E., Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. Proceedings of the Joint International Conference for Artificial Intelligence (IJCAI), pp. 975-980, 2007.

[16] Minton S., Johnston M., Philips A. and Laird P., Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, Vol. 58, pp. 161-205, 1992.

[17] Michalewicz Z., Genetic Algorithms + Data Structures = Evolution Programs, Springer, 1992.

[18] Garret S., Parameter-free adaptive clonal selection, IEEE Congress on Evolutionary Computation, 2004.