Towards an Immune System that solves CSP

María-Cristina Riff and Marcos Zúñiga

***Abstract*— Constraint satisfaction problems (CSPs) widely occur in artificial intelligence. In the last twenty years, many algorithms and heuristics were developed to solve CSP. Recently, bio-inspired algorithms have been proposed to solve CSP. They have shown to be more efficient than systematic approaches in solving hard instances. Given that recent publications indicate that Immune systems offer advantages to solve complex problems, our aim here is to propose an efficient immune system which can solve CSPs. We propose an immune system which is able to solve hard constraint satisfaction problems. The tests were carried out using random generated binary constraint satisfaction problems on the transition phase.**

I. Introduction

Constraint satisfaction problems (CSPs) widely occur in artificial intelligence. They involve finding values for problem variables subject to constraints on which combinations are acceptable. 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 constraint network is composed of the variables, the domains and the constraints. Over the few years, many algorithms and heuristics were developed to find a solution in constraint networks. 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 [4], [8], [10], [12], [11], [13], [14]. Given that recent publications indicate that Immune systems offer advantages in solving complex problems [1], [3], [19] our aim here is to propose an efficient immune system which can solve a wide range of binary CSPs. The contributions of this paper are:

- An Immune System which can solve hard CSPs
- A comparison of the performance with other well known incomplete approach

The paper is structured as follows. In the next section, we briefly explain the Artificial Immune Framework. In section 3, we define the Constraint Satisfaction Problem, and we also identify the conceptual relationship between the problem with the components of the Artificial Immune Framework. In section 4 we introduce our new approach CD-NAIS. The results of tests and a comparison with other incomplete method are given in section 5. In our summary, we give some conclusions and future works.

Supported by Fondecyt Project 1060377

Department of Computer Science, Universidad Técnica Federico Santa María, Valparaíso, Chile, María-Cristina.Riff@inf.utfsm.cl

Projet ORION, INRIA Sophia-Antipolis, 92250 Route des Lucioles, France, Marcos.Zuniga@sophia.inria.fr

II. Artificial Immune Systems

Artificial Immune Systems (AIS) are adaptive systems inspired by immunological theory [1]. From the information processing point of view an AIS is a parallel and a distributed adaptive system. It uses learning, memory and associated recovery to do recognition and classification tasks. Roughly speaking, the principal function of an immune system is to protect the individual from the repeated attacks of external agents. The system recognizes and discards doing an immune answer coming from one of the two levels: the innate immune system or the adaptive immune system. The cells of the innate immune system are immediately able to take action against external attacks. In the adaptive immune level, the antibodies are produced as an answer to specific infections. These cells can develop a memory, thus they will be able to recognize a future similar attack. L. de Castro [1] proposed a framework to design an AIS with the following components:

1) A representation to create abstract models of organs, cells and immune molecules (antigen, antibody). A molecule can, in general, be represented by a form space S like an attribute chain (set of coordinates) of size L. Thus, an attributes chain $m =< m_1, m_2, ..., m_L >$ corresponds to a point in the form space.
2) A set of functions called affinity functions to quantify the interactions between the AIS components (organs, cells and molecules).
3) A set of algorithms to simulate the immune behavior. In this work we use two of the most known models: the clonal selection and the immune network. The clonal selection manages the interaction of the immune system components. The immune network is used to simulate both the dynamic and metadynamic behavior.

III. Problem Formulation

We consider a constraint satisfaction problem (CSP) as defined by Mackworth [9], which can be stated briefly as follows: Given a set of variables, a domain of possible values for each variable, and a conjunction of constraints, each constraint is a relation defined over a subset of the variables, limiting the combination of values that the variables in this subset can take. The goal is 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.

1-4244-1340-0/07$25.00 ©2007 IEEE

A. Notions on CSP

A *Constraint Satisfaction Problem* (CSP) is composed of a set of *variables* $V = \{X_1, \ldots, X_n\}$, their related *domains* $D_1, \ldots, D_n$ and a set θ containing η *constraints* on these variables. The domain of a variable is a set of values to which the variable may be instantiated. Each variable X_j is *relevant* (in the next we denote *being relevant for* by $\rhd$), to a subset of constraints $C_{j_1}, \ldots, C_{j_k}$ where $\{j_1, \ldots, j_k\}$ is some subsequence of $\{1, 2, \ldots, \eta\}$. A constraint which has exactly one relevant variable is called a *unary constraint*. Similarly, a *binary constraint* has exactly two relevant variables. A binary CSP is associated with a constraint graph, where nodes represent variables and arcs represent constraints. If two values assigned to variables that share a constraint are not among the acceptable value-pairs of that constraint, this is an *inconsistency* or constraint violation.

B. Immune Components for CSP

Our algorithm uses three immune components: Antigen, Antibody and B-cells. Basically, the Antigen represents the information for each variable given by the constraint graph. Thus, it just depends on the problem and not 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. The immune components in our approach are defined as follows:

Definition 3.1: (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, $\forall i, i = 1, \ldots, n$.

Thus, the antigen represents the maximal number of inconsistencies for each variable. 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 3.2: (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,e.g. $(X_1, \ldots, X_n) \rightarrow (14, 18, \ldots, p, q)$

Remark: The structural segment corresponds to an instantiation **I** of the CSP.

Definition 3.3: (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. For the Artificial Immune Systems, affinity is a measurement of the interaction between two immune components. 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.

- **Interaction between $Ag \leftrightarrow Ab_c$:**

  It is an estimation of how far the antibody is from being a CSP solution. 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 Af_d to measure this affinity as:

$$\mathbf{Af_d(Ag, Ab_c)} = \sqrt{\sum_{\mathbf{i=1}}^{\mathbf{n}} (\mathbf{Ag_i - Ab_{c_i} + Fd_i})^{\mathbf{2}}} \quad (1)$$

  where Fd_i is called the dispersion factor defined by:

$$\mathbf{Fd_i} = \frac{\mathbf{d_i}}{\mathbf{(n-1)} \cdot \mathbf{Ag_i}} \quad (2)$$

  with d_i equal to:

$$\mathbf{d_i} = \sum_{\mathbf{i \neq j, j=1}}^{\mathbf{n}} |\mathbf{Ab_{c_i} - Ab_{c_j}}| \quad (3)$$

  The function Af_d does not only prefer a pre-solution with a minimal number of violated constraints, but it also takes into account how hard for the algorithm to repair this pre-solution could be. This is done by including the function d_i as a conflicts dispersion measure. Thus, given two pre-solutions that satisfy the same number of constraints, the algorithm prefers the one with the smaller number of variables involved in the constraints violations. The value of the dispersion factor Fd_i belongs to $[0, 1[$. The Fd_i value is equal to 0 either when any of the variables are in conflict (it is a solution) or when all the variables are involved in the same number of conflicts.
- **Interaction between $Ab_{s_i} \leftrightarrow Ab_{s_j}$:**

  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 prefers to have a diversity of pre-solutions.

Finally, 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 Af_d value.
- The hypermutation ratio used in the affinity maturation step. This ratio is inversely proportional to the Af_d value.

IV. Constraint Directed - Network Artificial Immune System

We called our algorithm CD-NAIS which stands for Constraint Directed - Network Artificial Immune System. It is shown in figure 1.

The algorithm works with a set of B-cells, following an iterative maturation process. Some of these B-cells are selected, doing a *Clonal Selection*, preferring those with bigger affinity values Af_d. It uses a Roulette Wheel selection. The algorithm generates clones of the B-cells selected, that is done by the *Clonal Expansion*, and these clones follow a hypermutation process in the *Affinity Maturation* step. The hypermutation is a hill-climbing procedure that repairs the Antibody. The new set of B-cells is composed of a selected set of hypermutated B-cells. 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 and only if $(1-\frac{HA_s}{n}) > \epsilon$. Therefore, the $epsilon$ value is used by the algorithm to manage the minimal degree of diversity. The ϵ value is known as the threshold of crossing reactivity. The function *Build Network*, using the clones list ordered by affinity (from higher to lower affinity), sequentially selects the clones with higher affinity which suppression counter is zero. If a clone presents a high similarity ($(1 - \frac{HA_s}{n}) < \epsilon$) with a clone already selected to be part of the new set of B-cells, its suppression counter will be incremented. If the list of clones is over and there is still space in the new B-cells set, the remaining spaces will be filled with the clones presenting a lower suppression counter. The algorithm adds new B-cells randomly generated by the *Metadynamic* procedure to this set of B-cells, suppressing the lower affinity B-cells, by a pre-defined new B-cells insertion rate n_2.

Thus, the algorithm does exploration and exploitation.

V. Tests

The goal of the following benchmarks is to evaluate the performance of CD-NAIS for solving CSP. The algorithm has been tested with randomly generated binary CSPs, [5]. We have done two kinds of test. The goal of the first set of experiments is to validate that CD-NAIS is able to solve hard instances of CSPs. The second test is to do a comparison between CD-NAIS and the most successful evolutionary algorithm reported, named SAW [14].

function CD-NAIS(CSP)
Begin
$Ag = compute\_connections(CSP, n)$;
$BCELLS = random(BCELLS\_NUM, n)$;
For i=1 **to** BCELLS_NUM **do**
  Compute BCELLS[i] affinity
End For
While $(j \leq MAX\_ITER)$ **or** not solution **do**
  Select a set of BCELLS
  CLONES = *Clonal Expansion*(BCELLS selected);
  CLONES = *Affinity Maturation* (CLONES);
For $i = 1$ **to** $CLONES\_NUM$ **do**
  Compute CLONES[i] affinity
End For
BCELLS = *Build Network*(CLONES);
BCELLS = *Metadynamic*(BCELLS);
End While
Return BCELLS;
End

Fig. 1. CD-NAIS Pseudocode

A. Hardware

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.

B. Parameters

The parameter values of the algorithm for both set of experiments, determined by tuning, are:

- $n_1 = 0.5$, rate of cells to be expanded,
- $n_2 = 0.1$, rate of cells to be incorporated on the memory
- $\epsilon = 0.46$, threshold reactivity between clones
- B-cells $= 10$
- Number of clones = 100

C. Tests in the hard zone

The idea of these tests is to study the behavior of the algorithm solving hard problems. We have fixed $p_1 = 1$, with $n = m$ in $\{10, 15, 20\}$ with $p_2 \in \{p_{2_{crit}} - 0.1; p_{2_{crit}} - 0.05; p_{2_{crit}}; p_{2_{crit}} + 0.05; p_{2_{crit}} + 0.1\}$. Where $p_{2_{crit}}$ is computed using the function proposed by B. Smith in [5] to obtain problems on the transition phase. For the 15 problem classes we have tested 20 there were instances using 20 runs. That is 400 algorithm executions. The algorithm has used a maximum number of iterations of 2000. The results are shown in the graphs in figures 2, 3, 4. When considering the 400 executions, the first figure shows the successful rate. The problem becomes harder to be solved for $< 20, 20, 1, p_{2crit} >$ configuration. The problems in the transition phase are more difficult as we could expect. The second graph shows the average of the final number of conflicts remaining for each class. For the configurations with 10 variables, the number of constraints is 45. The average number of conflicts unsolved is around 0.06 which

means that CD-NAIS have found a solution for most of the problems in this configuration. In the configuration with 15 variables, the number of constraints is 105. From these, on average, the number of remaining conflicts is 0.4, and in the worst cases, just one constraint is violated. As the number of variables increases the problems became harder. With 20 variables the number of constraints is 190. For the worse cases, 5 constraints remain unsatisfied and for the best case all are satisfied. In order to evaluate the complexity of the algorithm, the third graph illustrates the average of the number of constraints checks for each class of problems. From these graphs we can observe that the transition phase is identified for CD-NAIS.

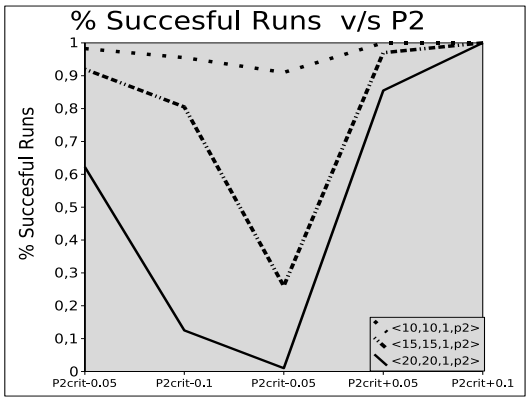

Fig. 2. Different problems tested, comparison of % Succesful runs

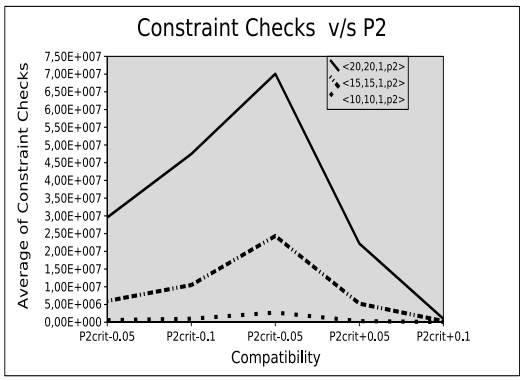

Fig. 3. Different problems tested, comparison of Number of Constraints Checks

D. Experimental Comparison

This set of experiments is done to compare CD-NAIS with SAW which is the best reported evolutionary algorithm to solve CSPs. The problems are those generated and used by Craenen et al. [14] to compare evolutionary algorithms

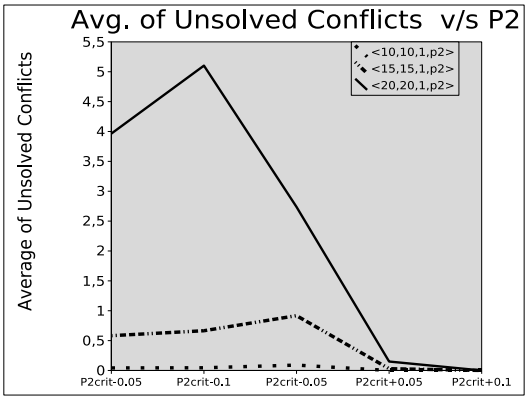

Fig. 4. Different problems tested, comparison of Average of final unsolved conflicts

p	SAW		CD-NAIS			
	10^5 ev.	time	1.000 it.	time	6.600 it.	time
0.24	100	0.74	93.2	0.51	100	0.48
0.25	100	2.33	81.2	0.73	99.6	0.91
0.26	97	6.38	62	0.96	96.8	2.20
0.27	60	11.39	38.4	1.21	80.8	4.05
0.28	25	18.66	18.8	1.38	53.6	6.30
0.29	17	20.57	4.4	1.51	18	8.97
0.3	5	22.27	1.6	1.54	11.6	9.45
0.31	1	22.47	0.8	1.56	1.2	10.18
0.32	0	22.39	0.12	1.57	2.4	10.13
0.33	1	22.38	0.15	1.57	3.6	10.09

Fig. 5. Comparison of the Percentage of Problems Solved

on binary CSPs[1]. They generated the problems using model $\mathrm{E}(n, m, p, k)$. The tests have 250 problem instances using model $\mathrm{E}(20, 20, p, 2)$ and all problems have at least one solution. They fixed the maximum number of evaluations in 100.000 for each algorithm. This limit is equivalent in our approach to 6.600 iterations. We have also tested CD-NAIS using a maximum of 50.000 and 1.000 iterations. These results and the average time invested in seconds for each of the 250 runs are shown in figure 6. From this table we can observe that CD-NAIS can solve more problems and quicker than SAW. The percentage of problems solved by SAW is 40% instead of 52% solved by CD-NAIS using a limited 6.600 iterations. CD-NAIS with a limit of 1.000 iterations can obtain a success rate of 30% in less than 2 seconds per run. Using 50.000 iterations CD-NAIS improves it success rate, solving 61% of problems requiring on average just 35 seconds per run, using in the worst case 74 seconds. The code for CD-NAIS is available in the web page http://www-sop.inria.fr/orion/personnel/Marcos.Zuniga/CSPsolver.zip

[1]We have obtained the problems and the SAW code from the web page http://www.xs4all.nl/˜bcraenen/resources.html

p	50.000 it.	time
0.24	100	0.52
0.25	100	0.92
0.26	100	1.72
0.27	99.6	5.28
0.28	91.2	20.41
0.29	61.6	45.00
0.3	24	63.33
0.31	7.6	73.54
0.32	10	71.29
0.33	10.4	72.10

Fig. 6. Percentage of Problems Solved in 50.000 iterations for CD-NAIS

VI. Conclusions

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 the involved variables in the 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 CD-NAIS has solved, on average, 12% more problems than SAW, the best known evolutionary algorithm. Artificial Immune Systems is a promising technique to solve constrained combinatorial problems.

VII. Future Work

A promising research area is to incorporate some parameter control strategies into the algorithm. Parameter control strategies have shown to be very useful in genetic algorithms and it seems to be a good mechanism to guide the algorithm to different levels of exploration/exploitation depending on the state of the search.

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.

[19] Vincenzo Cutello, Giuseppe Nicosia: A Clonal Selection Algorithm for Coloring, Hitting Set and Satisfiability Problems. pp. 324-337, WIRN/NAIS 2005.