An Immunological Approach to Combinatorial Optimization Problems

1 Introduction

The idea to use the theory of evolution as a search algorithm for optimization problems emerged, in the sixties, independently in the USA with the J. Holland 's *Genetic Algorithms* (GA) and in Germany with the I. Rechenberg 's *Evolution Strategies* (ES). Both approaches mimicked the collective learning process of natural populations. Today, there are several different types of *Evolutionary Algorithms* (EAs) (GA, ES, Evolutionary Programming, Genetic Programming, Immune Algorithms) that solve real world applications: "When we fly in the latest airplane, the design of the turbines may have been optimized by artificial evolution"[1].

Recently a new method has been proposed, denoted by *Extremal Optimization*, for finding solutions to hard optimization problems. The method is inspired by self-organizing processes often found in nature [2].

Moreover, there is a deep connection between NP-complete problems and models studied in statistical physics (e.g. spin glasses), that leads to determining computational complexity from characteristic 'phase transitions' in K-satisfiability [3]. This result strengthens the link between computational models and properties of physical systems.

The paper is organized as follows: in section 2 we briefly introduce the research field of Immunological Computation, in section 3 we describe the proposed Immunological Algorithms (IA), in 4 we define the test bed where we verify the IA's performance and report simulations' results. In section 5 we reports about other related works and finally in 6 we highlight some future directions.

2 Immunological Computation

The Immune System (IS) has to assure recognition of each potentially dangerous molecule or substance, generically called antigen, that can infect the host organism. The IS first recognizes it as *dangerous* or extraneous and then mounts a response to eliminate it. To detect an antigen, the IS activates a recognition process. Moreover, the IS only has finite resources and often very little time to produce antibodies for each possible antigen [4].

Our Immune Algorithm is based on the *theory of the clonal selection* first stated by Burnet and Ledeberg in 1959 [5]. This theory suggests that among all the possible cells with different receptors circulating in the host organism, only those who are actually able to recognize the antigen will start to proliferate by duplication (cloning). The increase of those population and the production

of cells with longer expected life-time assures the organism a higher specific responsiveness to that antigenic pattern, establishing a defense over time (immune memory). In particular, on recognition, B and T memory cells are produced. Plasma B cells, deriving from stimulated B lymphocytes, are in charge of the production of antibodies targeting the antigen.

This mechanism is usually observed when looking at the population of lymphocytes in two subsequent antigenic infections. The first exposition to the antigen triggers the *primary response*. In this phase the antigen is recognized and memory is developed. During the *secondary response*, that occurs when the same antigen is encountered again, a rapid and more abundant production of antibodies is observed, resulting from the stimulation of the cells already specialized and present as memory cells.

The *hypermutation* phenomenon observed during the immune responses is a consequence of the fact that the DNA portion coding for the antibodies is subjected to mutation during the proliferation of the B lymphocytes. This provides the system with the ability to generate diversity. The IS from an information processing point of view [6] can been considered like a problem learning and solving system. The antigen is the problem to solve, the antibody is the generated solution. At beginning of the primary response the antigen-problem is recognized by partial candidate solution (the B cell receptor). At the end of the primary response the antigen-problem is defeated-solved by candidate solutions (antibody or a set of antibodies in the case of multimodal optimization). Consequently the primary response corresponds to a training phase while the secondary response is the testing phase where we will try to solve problems similar to the original presented in the primary response [7].

The new field of *Immunological Computation* (or Artificial Immune System) attempts to use methods and concepts such ideas to design immunity-based system applications in science and engineering [8]. Immune Algorithms (IA) are adaptive systems in which learning takes place by evolutionary mechanisms similar to biological evolution.

Thus one wants, first, to understand the dynamics of such a complex behavior when they face particular computational problems that are normally solved by conventional specialized algorithms and, second, one wishes to develop new techniques that mimic the natural systems under study to catch their ability to solve problem otherwise difficult to be solved by conventional methods.

3 Immune Algorithms

Following the track of the computer experiments performed by Nicosia *et al.* [7] we focus our attention to the problem solving ability of the IS and present a new immune algorithm. Our approach uses a simplified model of the immune system to explore the problem solving feature. We consider only two entities: antigens and antibodies. The antigen is the combinatorial optimization problem and the antibody is the candidate solution. Antigen is a set of variables that models the problem. Antibodies are modeled as binary strings of length l.

The input is the antigen problem, the population size (d) and the number of clones for each cell (dup). The set $S^{d\times l}$ denotes a population of d individuals of length l, and it represents the space of feasible and unfeasible candidate solutions. After a random initialization and evaluation of cell populations $P^{(0)}$, the loop iterates the cloning of all antibodies, each antibodies produce dup clones, generating the population P^{clo}. Next step is to mutate a random bit for each antibody in P^{clo} generating the population P^{hyp}. The mechanism of mutation of the cell receptor is modeled by a random process with parameter l, i.e. the length of the cell receptor. This immunological operator is important because it modifies continuously the receptors in presence like a neural network whose structure (the layers, the number and the nature of neurons) would change in time. After the evaluation of P^{hyp} at time t the algorithm selects the best d antibodies from $(P^{hyp} \sqcup P^{(t)})$[1] (a simple elitist strategy) and creates the new set $P^{(t+1)}$). The output is basically the candidate solutions-clones that have solved-recognized the antigen.

This simplistic view does not represent a strong limitation because in general one can give whatever meaning to the bit string representing the candidate solution and use much more complicated mutation operator than the simple bit-flip, e.g., any map $f : \{0,1\}^l \to \{0,1\}^l$ could determine a different search algorithm. The underlying evolutionary *engine* remains the same.

A pseudo-code version of the algorithm is given below.

Immune Algorithm
$t := 0$;
Initialize $P^{(0)} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_d}\} \in S^{d\times l}$
Evaluate $P^{(0)}$;
while (T($P^{(t)}$)= 0) *do*
  P^{clo} := Cloning $(P^{(t)}, dup)$;
  P^{hyp} := Hypermutation (P^{clo});
  Evaluate (P^{hyp});
  $P^{(t+1)}$:= Select the d highest affinity individual $(P^{hyp} \sqcup P^{(t)})$;
  $t := t + 1$;
od

T denotes a termination criterion, that is, the algorithm terminates if a solution is found, or a maximum number of evaluations is reached.

In evolutionary computation the selection of an appropriate population size is important and could greatly affect the effectiveness and efficiency of the optimization performance. For this reason EA's with dynamic population size achieve better convergence rate and discover as well any gaps or missing tradeoff regions at each generation [9].

All evolutionary algorithms need to set an optimal population size in order to discover and distribute the *nondominated individuals* along the Pareto front [10]. If the population size is too small, EAs may suffer from *premature convergence*,

[1] Note that this is a multi-set union, since we want to allow an individual to appear more than once.

while if the population size is too large, undesired computational overwork may be incurred and the waiting time for a fitness improvement may be too long in practice. We propose here a simple search algorithm with only two parameters d and dup. The correct setting of these parameters allows to discover the non-dominanted individuals without using dynamic population. We observe that the evolutionary engine uses a process of *expansion and reduction*. The expansion from population $P^{(t)}$ with $\mid d \mid$ individuals into population P^{hyp} with $\mid d \times dup \mid$ individuals is performed by cloning and hypermutation operators, the reduction from P^{hyp} with $\mid d \times dup \mid$ into $P^{(t+1)}$ with $\mid d \mid$ is performed by means of a selection operator. The expansion phase explores the fitness landscape at a given generation t, the reduction phase decides which individuals to select for the next generation $t+1$.

4 NP-complete problems

To test our algorithm we chose two NP-complete problems.

4.1 The Minimum Hitting Set Problem.

An instance of the Minimum Hitting Set problem consists of a collection S of subsets of a finite set U and a positive integer $k \leq \mid U \mid$. Question: Is there a subset $U^{'} \subseteq U$, with $\mid U^{'} \mid \leq k$, such that $U^{'}$ contains at least one element from each subset in S?

This problem is NP-complete. Indeed, membership to NP can be easily observed, since a guessed proposed hitting set $U^{'}$ can be checked in polynomial time. NP-hardness is obtained by polynomial reduction from the Vertex Cover [11].

We work with a fitness function that allows us to allocate feasible and unfeasible candidate solutions.

$$f_{hs}(\boldsymbol{x}) = Cardinality(\boldsymbol{x}) + (\mid S \mid - Hits(\boldsymbol{x}))$$

The candidate solution must optimize both terms. Each population member $\boldsymbol{x}$ must minimize the size of set $U^{'}$ and maximize the number of hit sets. If $(\mid S \mid - Hits(\boldsymbol{x})) = 0$, $\boldsymbol{x}$ is a hitting set, that is a feasible solution.

The used fitness gives equal opportunity to the evolutionary process to minimize both terms. For example, if we have a collection S of 50000 sets and the following two individuals: $\boldsymbol{x}_1$, with $Cardinality(\boldsymbol{x}_1) = 40$, $Hits(\boldsymbol{x}) = 49997$, $f_{hs}(\boldsymbol{x}_1) = 43$; $\boldsymbol{x}_2$, with $Cardinality(\boldsymbol{x}_2) = 42$, $Hits(\mathbf{2}) = 49999$, $f_{hs}(\boldsymbol{x}_1) = 43$, it is difficult to decide which individual is the best, the choice is crucial and strongly influences the subsequent search in the landscape.

We test our IA by considering randomly generated instances of the Minimum Hitting Set problem. Fixed $\mid U \mid = 100$ we construct two types of instances with $\mid S \mid$ equal respectively to 1000 and 10000 (denoted by "hs100-1000" and "hs100-10000"). The third instance, $\mid S \mid = 50000$ ("hs-100-50000") is a very

hard instances for the Minimum Hitting Set problem[2]. The best solution found has cardinality 39 [12].

Table 1. Minimum Hitting Set Instances

	hs100-1000		hs100-10000		hs100-50000	
d	25	100	50	200	50	200
dup	15	30	15	15	15	10
$best$	6	6	9	9	39	39
$\#min$	1	3	3	5	3	2
AES	2275	18000	6800	27200	45050	98200

Our experimental results are reported in Table 1. One can see the best hitting set found by the IA for each instance, the parameter values and the average number of evaluations to solutions (AES). For each problem instance we performed 100 independent runs. We also provide the number of minimal hitting sets found.

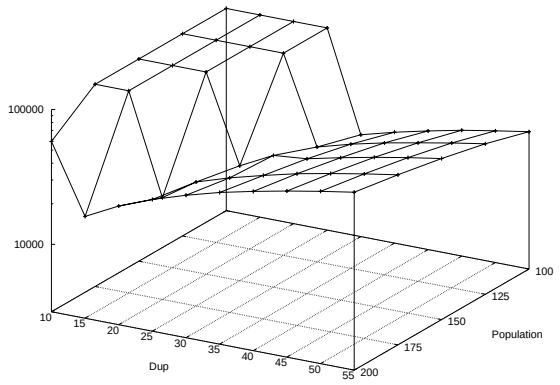

Fig. 1. 3D representation of experimental results, with dimensions: d, dup and AES

By inspecting the results, one can also see that increasing the values of d and dup, the number of optimal solutions found increases as well, and in turn, the average number of fitness evaluations. Last column shows that for hard instances,

[2] All the three instances are available at
http://www.dmi.unict.it/~ cutello/eracop.html

even if we increase the population size and decrease the dup, the number of found solutions decreases.

In figure 1, we show the 3D graphic of a set of numerical simulation. To understand the algorithm's behavior when changing the parameters, we performed a set of experiments on a simple Hitting Set instance, "hs100-1000", in which d and dup are given several values.

The problem solving ability of the algorithm depends heavily upon the number of individuals we "displace"on the space $S^{d \times l}$ and on the duplication parameter dup. The population size varies from 100 to 200 while dup from 10 to 55. The Z axis shows the average number of evaluations to solutions (we allowed $T_{max} = 100000$ evaluations and performed 100 independent runs). The population d strongly influences the number of solutions found. Indeed, when increasing d the number of nondominated solutions found increases. On the other hand, the average number of fitness function evaluation also increases. The value of dup influences the convergence speed.

In figure 2, we can see the fitness function values for the optimization process in action during the first 70 generations. The minimum hitting set is obtained at generation 60. Subsequently, other antibodies with the same cardinality are discovered, completing the set of (found) nondominated solutions.

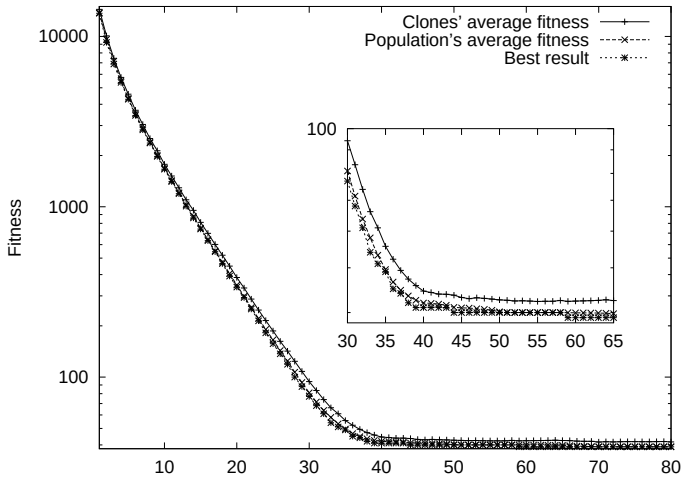

Fig. 2. Fitness function versus generations. Y axis is in log scale and origin is fixed at X-value 39.

4.2 The 3-SAT Problem.

3-SAT is a fundamental NP-complete problem in propositional logic [11]. An instance of the 3-SAT problem consists of a set V of variables, a collection C

of clauses over V such that each clause $c \in C$ has $\mid c \mid = 3$. The problem is to find a satisfying truth assignment for C. The 3-SAT, and in general K-SAT, for $K \geq 3$, is a classic test bed for theoretical and experimental works.

The fitness function for 3-SAT is very simple, it computes only the number of satisfied clauses

$$f_{sat}(\boldsymbol{x}) = \#\text{SatisfiedClauses}(C, \boldsymbol{x})$$

For our experiments, we used A. van Gelder's 3-SAT problem instance generator, MKCNF.C[3].

The program mkcnf.c generates a "random" constant-clause-length CNF formula and can force formulas to be satisfiable. For accuracy, we perform our tests in the transition phase, where the hardest instances of the problem are located [13]. In this case we have $\mid C \mid = 4.3 \mid V \mid$.

The generated instances are the following:

(i1) "sat30-129" number of variables 30, number of clauses 129 and random seed 83791 (mkcnf.c's input parameter useful to reproduce the same formula),
(i2) "sat30-129" (83892),
(i3) "sat30-129" (83792),
(i4) "sat40-172" with $\mid C \mid = 40$, $\mid V \mid = 172$ and random seed 62222, and
(i5) "sat50-215" with random seed 82635.

Table 2. 3-SAT Instances

	(i1)	(i2)	(i3)	(i4)	(i5)
d	50	50	50	50	75
dup	20	30	20	30	15
AES	14632	292481	11468	24276	50269

The experimental results for the above 3-SAT instances are shown in Table 2. We underline that for each instance we performed 50 independent runs.

To conclude, we note that the first three instances involve different formulae with the same number of variables and clauses. We can observe that among difficult formulae, there are even more difficult ones (at least for our algorithm), as the different number of evaluations proves. Such a phenomenon can be observed also for formulae with a higher number of variables and clauses. In determining a truth assignment AES grows proportionally to the difficulty of satisfying the formula.

In table 3 we show the result of the last set of experiments. In this case we use the immune algorithm with a SAW (stepwise adaptation of weights) mechanism

[3] Available by anonymous ftp at
ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSS/instances

Table 3. 3-SAT Instances

Case	$\|V\|$	$\|C\|$	Randseed	SuccessR	AES	SuccessR	AES
1	30	129	83791	1	2708	1	6063
2	30	129	83892	0.94	22804	0.96	78985
3	30	129	83792	1	12142	1	31526
4	40	172	83792	1	9078	1	13328
5	40	172	72581	0.82	37913	1	2899
6	40	172	62222	1	37264	0.94	82031
7	50	215	87112	0.58	17342	1	28026
8	50	215	82635	1	42137	1	60160
9	50	215	81619	0.26	67217	0.32	147718
10	100	430	87654	0.32	99804	0.06	192403
11	100	430	78654	0.04	78816	0.44	136152
12	100	430	77665	0.32	97173	0.58	109091

[14]. A weight is associated with each clause, the weights of all clauses that remains unsatisfied after T_p generation are incremented ($\delta w = 1$). Solving a constraint with a high weight gives high reward and thus the more pressure is given on satisfying those constraints, *the hardest clauses.* In the table, the parameter $SuccessR$ represents the number of times that the formula is satisfied. The last two columns refer to experimental results in [15] performed with an evolutionary algorithms with SAW mechanism. The table shows that the IA with SAW mechanism, outperforms in many cases the results in [15], both in terms of success rate and computational performance, i.e. a lower number of fitness evaluations.

4.3 General remark about the obtained results

Last set of experimental results shows how a simple randomized search algorithm coupled with a mechanism for adaptive recognition of hardest constraints, is sufficient to obtain optimal solutions for any combinatorial optimization problem. Proving the above statement, is the major goal of our research. The above is, obviously, consistent with the latest assumptions in Evolutionary Algorithms, that the most important features are the fitness function and the crossover operator.

5 Related works

The *clonal selection algorithm* described in [16] represent a straightforward usage of the ideas upon which the theory of the clonal selection is stated in [5]. The clonal selection is itself a Darwinian evolution process so that similarities with Evolution Strategies and Genetic Algorithms are natural candidates.

Our immune algorithm, instead, does not use proportional cloning and hypermutation inversely proportional to fitness value. We designed and use very

simple cloning and hypermutation operators. Moreover, there is neither a *birth operator* to introduce diversity in the current population nor a mutation rate (p_m) to flip a bit, B cells memory, nor threshold m_c to to clone the best cells in the present population. We had simplified the immunological approach in order to better analyze and predict the algorithm's dynamics.

In this sense, also the approach described in [17] is similar to ours in that the solution is found by letting a population of unbiased potential solutions to evolve in a fitness landscape. Indeed, we are able to find similar results to their numerical experiments. The general philosophy agreement is expressed by using similar coding scheme, evaluation functions and the three immunological operators, i.e. selection, cloning and mutation.

Major differences with the above mentioned paradigms are cited below.

1. We consider relevant for the searching ability:
   (a) the size of the recognizing clones (the parameter *dup*), since it determines the size of the fitness landscape explored at each generation,
   (b) the number of individuals (d) since it determines the problem solving capacity. This is in contrast with most of the artificial evolution methods where a typical fixed population's size of *a thousand or less* individuals is used [1].
2. We consider only two immunological entities, antigens and antibodies, two parameters, *dup* and d, and simple immune operators.

6 Conclusions

One relevant disadvantage of our IA is that the search process may stop with a local minimum, when you are working with a small population size and a duplication parameter not sufficiently large. Moreover, the computational work increases proportionally to the size of these parameters. This slows down the search process in the space of feasible solutions although it gives better chances of finding good approximations to optimal solutions. The selection operator makes use of elitism, which on one hand speeds up the search, but on the other hand, may force the population to get trapped around a local minimum, reducing the diversity.

In conclusion, our algorithm is simple, efficient and it is certainly suitable for further studies and tests and a deep theoretical analysis.

Acknowledgments

GN gratefully acknowledge the Italian Universities Consortium Computing Center (CINECA) and University of Catania project "Young Researcher" for partial support.

References

1. Brooks, R.: The relationship between matter and life. Nature **409** (2001) 409–411.

2. Boettcher, S., Percus, A.: Nature's way to optimizing. Artificial Intelligence **119** (2000) 275–286.
3. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determing computational complexity from characteristic 'phase transitions'. Nature **400** (1999) 133–137.
4. Perelson, A. S., Weisbuch, G., Coutinho, A. Eds.: Theoretical and Experimental Insights into Immunology. New York, NY: Springer-Verlag (1992).
5. Burnet, F. M.: The Clonal Selection Theory of Acquired Immunity. Cambridge, U.K.: Cambridge Univ. Press (1959).
6. Forrest, S., Hofmeyr, S. A.: Immunology as Information Processing. Design Principles for Immune System & Other Distributed Autonomous Systems. New York: Oxford Univ. Press, SFI Studies in the Sciences of Complexity (2000).
7. Nicosia, G., Castiglione, F., Motta, S.: Pattern Recognition by primary and secondary response of an Artificial Immune System. Theory in Biosciences **120** (2001) 93–106.
8. Dasgupta, D. Ed.: Artificial Immune Systems and their Applications. Berlin, Germany: Springer-Verlag (1998).
9. Tan, K. C., Lee, T. H., Khor, E. F.: Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization. IEEE Transactions on Evolutionary Computation **5** (2001) 565–588.
10. Coello Coello, C. A.: An Updated Survey of GA-Based Multiobjective Optimization Techniques. ACM Computing Survey **32** (2000) 109–143.
11. Garey, M. R., Johnson, D. S.: Computers and Intractability: a Guide to the Theory of NP-completeness. New York: Freeman (1979).
12. Cutello, V., Mastriani, E., Pappalardo, F.: An Evolutionary Algorithm for the T-constrained variation of the Minimum Hitting Set Problem. Proc. of the IEEE World Congress on Computational Intelligence. Honolulu, HI (2002).
13. Mitchell, D., Selman, B., Levesque, H. J.: Hard and easy distributions of SAT problems. Proc. of the AAAI. San Jose, CA (1992) 459–465.
14. Eiben, A. E., van der Hauw J. K., van Hemert J. I.: Graph coloring with adaptive evolutionary algorithms. J. of Heuristics **4** (1998) 25–46.
15. Bäck, T., Eiben, A. E., Vink, M. E.: A superior evolutionary algorithm for 3-SAT. Proc. of the 7th Annual Conference on Evolutionary Programming. Lecture Notes in Computer Science **1477** (1998) 125–136.
16. De Castro, L. N., Von Zuben, F. J.: The Clonal Selection Algorithm with Engineering Applications. Workshop Proc. of the Genetic and Evolutionary Computation Conference (GECCO'00). Las Vegas, NV: Morgan Kaufmann (2000) 36–37.
17. Forrest, S., Perelson, A., Allen, L., Cherukuri, R.: Self-Nonself discrimination in a computer. Proc. of the IEEE Symposium on Research in Security and Privacy. Oakland, CA: IEEE Press (1994) 202–212.