Exploring the Capability of Immune Algorithms: A Characterization of Hypermutation Operators

Vincenzo Cutello[1], Giuseppe Nicosia[1,2], and Mario Pavone[1]

[1] Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6, 95125 Catania, Italy
{cutello,nicosia,mpavone}@dmi.unict.it
[2] Computing Laboratory, University of Kent Canterbury, Kent CT2 7NF

Abstract. In this paper, an important class of hypermutation operators are discussed and quantitatively compared with respect to their success rate and computational cost. We use a standard Immune Algorithm (IA), based on the clonal selection principle to investigate the searching capability of the designed hypermutation operators. We computed the parameter surface for each variation operator to predict the best parameter setting for each operator and their combination. The experimental investigation in which we use a standard clonal selection algorithm with different hypermutation operators on a complex "toy problem", the trap functions, and a complex NP-complete problem, the 2D HP model for the protein structure prediction problem, clarifies that only few really different and useful hypermutation operators exist, namely: inversely proportional hypermutation, static hypermutation and hypermacromutation operators. The combination of static and inversely proportional Hypermutation and hypermacromutation showed the best experimental results for the "toy problem" and the NP-complete problem.

Keywords: clonal selection algorithms, hypermutation operators, hypermacromutation operator, aging operator, trap functions, 2D HP protein structure prediction problem.

1 Introduction

Immune Algorithms (IA's) based on the clonal selection principle [1], are a special kind of immune algorithms [2, 3] where the clonal expansion and the affinity maturation are the main forces behind the searching process. The hypermutation operator explores the actual fitness landscape by introducing innovations into the population of potential solutions to a given optimization problem. Hence, it is reasonable to test its effectiveness on any kind of IA, in particular, and evolutionary algorithm, in general. The clonal selection algorithm used in this research work is a modified version of a previous immune algorithm [4]. It has significant new features, hypermutation and aging operators [12], to face hard problems, allowing us to experimentally compare the different types of hypermutation operators. To this end, we ran many experiments to properly tune

the parameter values of each hypermutation operator, and in doing so, determining the corresponding parameter surfaces. Following the advice reported in [5], we test the class of hypermutation operators on a toy problem and a NP-complete problem. This pair of computational problems (for sake of completeness the author's advice in [5] is to use a real-world application as well), are robust test beds to analyze theoretically and experimentally the overall performance of evolutionary algorithms. We choose a complex, yet rarely used toy problem, trap functions, and a complex NP-complete problem, the 2D HP model for the protein structure prediction problem with a particular search space, the funnel landscape. We will tackle the trap functions and the 2D HP model for the protein structure prediction problem to assess the performances of the developed hypermutation operators. The experimental results reported in this article prove the good performances of the designed IA when it uses inversely proportional, or static, hypermutation coupled with hypermacromutation operator. Moreover, we will show that the hypermacromutation procedure is an useful variation operator to extend the regions of parameter surfaces with high success rate values, and, in particular, it helps in improving the regions where the hypermutation operator by itself performs poorly.

2 The Clonal Selection Algorithm

The IA we work with, uses only an entity type: the B cells. The B cell population, $P^{(t)}$, represents a set of candidate solution in the current fitness landscape at each time step (or generation) t. The B cell, or the B cell receptor, is a string of length ℓ, where every variable is associated either to one bit, or an integer, or a floating point, or a high level representation of data depending upon the given computational problem that the IA is facing. At each time step t, we have a B cell population $P^{(t)}$ of size d. The initial population, time $t = 0$, is randomly generated. The Algorithm uses two main functions:

- *Evaluate(P)* which computes the affinity (fitness) function value of each B cell $\boldsymbol{x} \in P^{(t)}$.
- *Termination_Condition()* which returns true if a solution is found, or a maximum number of fitness function evaluations (T_{max}) is reached.

It also uses three immune operators: cloning, hypermutation and aging; and a standard evolutionary operator: $(\mu + \lambda)$-selection operator.

Static Cloning Operator. The cloning operator, simply clones each B cell *dup* times, producing an intermediate population P^{clo} of size $d \times dup = Nc$. Throughout this paper, we will call it *static cloning operator*, as opposed to a *proportional cloning operator* [6], that clones B cells proportionally to their antigenic affinities. Preliminary experimental results using such an operator (not shown in this paper), showed us frequent premature convergence during the population evolution. In fact, proportional cloning gives more time steps to B cells with high affinity values, and the process can more likely be trapped into local minima of the given landscape.

Hypermutation Operators. The hypermutation operators act on the the B cell receptor of P^{clo}. The number of mutations M is determined by a specific function, *mutation potential*, that is, for each B cell at any given time step t. It is possible to define several mutation potentials. We tested our IA using static, proportional and inversely proportional hypermutation operators, hypermacromutation operator, and combination of hypermutation operators and hypermacromutation. The three hypermutation operators and the Hypermacromutation operator mutate the B cell receptors using different mutation potentials, depending upon a parameter c. If during the mutation process a constructive mutation occurs, the mutation procedure will move on to the next B cell. We call such an event: *Stop at the first constructive mutation* (FCM). We adopted such a mechanism to slow down (premature) convergence, exploring more accurately the search space. A different policy would make use of $M-$mutations ($M-mut$), where the mutation procedure performs all M mutations determined by the potential for the current B cell. The mutation potentials we used are:

(H1) *Static Hypermutation*: the number of mutations is independent from the fitness function f, so each B cell receptor at each time step will undergo at most $M_s(\boldsymbol{x}) = c$ mutations.

(H2) *Proportional Hypermutation*: the number of mutations is proportional to the fitness value, and for each B cell $\boldsymbol{x}$ is at most $M_p(f(\boldsymbol{x})) = (E^* - f(\boldsymbol{x})) \times (c \times \ell)$, where E^* is the minimum fitness function value known for the current instance problem. The shape of $M_p(f(\boldsymbol{x}))$ is a straight line.

(H3) *Inversely Proportional Hypermutation*: the number of mutations is inversely proportional to the fitness value. In particular, at each time step t, the operator will perform at most $M_i(f(\boldsymbol{x})) = ((1 - \frac{E^*}{f(\boldsymbol{x})}) \times (c \times \ell)) + (c \times \ell))$ mutations. In this case, $M_i(f(\boldsymbol{x}))$ has the shape of an hyperbola branch.

(M) *Hypermacromutation*: the number of mutations is independent from the fitness function f and the parameter c. In this case, we choose at random two integers, i and j such that $(i+1) \leq j \leq \ell$ and the operator mutates at most $M_m(\boldsymbol{x}) = j - i + 1$ values, in the range $[i, j]$.

Aging Operator. The aging operator eliminates old B cells from the populations $P^{(t)}$, $P^{(hyp)}$ and/or $P^{(macro)}$, so to avoid premature convergence. To increase the population diversity, new B cells are added by the *Elitist_Merge function*. The parameter τ_B sets the maximum number of generations allowed to B cells to remain in the population. When a B cell is $\tau_B + 1$ old it is erased from the current population, no matter what its fitness value is. We call this strategy, *static pure aging*. During the cloning expansion, a cloned B cell takes the age of its parent. After the hypermutation phase, a cloned B cell which successfully mutates, i.e. with a better fitness value, will be considered to have age equal to 0. Thus, an equal opportunity is given to each "new genotype" to effectively explore the fitness landscape. We note that for τ_B greater than the maximum number of allowed generations, the IA works essentially without aging operator. In such a limit case the algorithm uses a strong elitist selection strategy.

$(\mu+\lambda)$-selection with birth phase and no redundancy. A new population $P^{(t+1)}$, of d B cells, for the next generation $t+1$, is obtained by selecting the best B cells which "survived" the aging operator, from the populations $P^{(t)}$, $P^{(hyp)}$ and/or $P^{(macro)}$. No redundancy is allowed. Thus, each B cell receptor is unique, i.e. each genotype is different from all other genotypes. If only $d' < d$ B cells survived , the *Elitist_Merge* function creates $d - d'$ new B cells (*Birth phase*). Hence, the $(\mu + \lambda)$-selection operator (with $\mu = d$ and $\lambda = Nc$; or $\lambda = 2Nc$ if both variation operators are activated) reduces an offspring B cell population (created by cloning and hypermutation operators) of size $\lambda \geq \mu$ to a new parent population of size $\mu = d$. The selection operator chooses the d best elements from the offspring set and the old parent B cells, thus guaranteeing monotonicity in the evolution dynamic.

The properties of each immune operator is relatively well understood: the cloning operator explores the attractor basins and valleys of each candidate solution; the hypermutation operators introduce innovations in *exploring* the current population of B cell; the aging operator creates diversity during the searching process. The selection evolutionary operator directs the search process toward promising regions of the fitness landscape and *exploits* the information coded within the current population. While selection is a universal, problem- and algorithm-independent operator, hypermutation, and in general mutation and crossover, operators are specific operators, that focus on the structure of the given landscape. Here, we analyze the exploring capability of a class of hypermutation operators using a classical $(\mu + \lambda)$-selection operator, a simple cloning operator and a static pure aging process to stress the property of each hypermutation operator. Below we show the pseudo-code of the proposed Immune Algorithm (the boolean variables H, HM control, respectively, the hypermutation and the hypermacromutation operator).

Immune Algorithm$(\ell, d, dup, \tau_B, c, H, HM)$
$Nc := d * dup$;
$t := 0$;
$P^{(t)}$:= Initial_Pop();
Evaluate($P^{(0)}$);
while (¬ Termination_Condition()) **do**
  $P^{(clo)}$:= Cloning $(P^{(t)}, Nc)$;
  if (H) **then** $P^{(hyp)}$:= Hypermutation $(P^{(clo)}, c, \ell)$;
    Evaluate($P^{(hyp)}$);
  if (HM) **then** $P^{(macro)}$:= Hypermacromutation (P^{clo});
    Evaluate $(P^{(macro)})$;
  $(P_a^{(t)}, P_a^{(hyp)}, P_a^{(macro)})$:= Aging$(P^{(t)}, P^{(hyp)}, P^{(macro)}, \tau_B)$;
  $P^{(t+1)}$:= $(\mu + \lambda)$-Selection $(P_a^{(t)}, P_a^{(hyp)}, P_a^{(macro)})$;
  $t := t + 1$;
end_while

3 First test bed: the Trap Functions

The trap functions are complex *toy problem*, that can help in understanding the efficiency and the searching capability of evolutionary algorithms [5]. Toy problems such as ones-counting, Basin-with-a-barrier, Hurdle-problem, play a central role in understanding the dynamics of algorithms [7]. They allow algorithm designers to devise new tools for mathematical and experimental analysis and modelling. One can tackle toy problems to build-up a fruitful intuition on how the algorithms work. In this paper we used the trap functions to show the main differences between the hypermutation operators. The trap functions [8], simply, take as input the number of 1's of bit strings of length ℓ :

$$f(x) = \widehat{f}(u(x)) = \widehat{f}\left(\sum_{k=1}^{l} x_k\right) \tag{1}$$

An IA to face this toy problem would naturally use B cell receptors of bit string. For our experiments we will use two trap functions: a *simple trap function* and a *complex trap function*. The simple trap function is defined as follows:

$$\widehat{f}(u) = \begin{cases} \frac{a}{z}(z-u), & \text{if } u \leq z \\ \frac{b}{l-z}(u-z), & \text{otherwise.} \end{cases} \tag{2}$$

There are many possible choices for the parameters a, b and z. We choose the values used in [8]: $z \approx (1/4)\ell$; $b = \ell - z - 1$; $1.5b \leq a \leq 2b$; a a multiple of z. The simple trap function is characterized by a global optimum (for a bit string of all 0's) and a local optimum (for a bit string of all 1's) that are the each other *bit-wise* complement.

The complex trap function, how we will see, is more difficult to investigate, since there are two directions to get trapped. Its definition is the following:

$$\widehat{f}(u) = \begin{cases} \frac{a}{z_1}(z_1 - u), & \text{if } u \leq z_1 \\ \frac{b}{l-z_1}(u - z_1), & \text{if } z_1 < u \leq z_2 \\ \frac{b(z_2-z_1)}{l-z_1}\left(1 - \frac{1}{l-z_2}(u-z_2)\right) & \text{otherwise.} \end{cases} \tag{3}$$

We note that for $z_2 = \ell$ the complex trap function becomes the simple trap function. In this case, the possible values of the parameter z_2 are determined by the following equation $z_2 = \ell - z_1$. An *ad hoc* operator that mutates all the bits of the string does not obtain the global maximum of the complex trap function. The experimental results are shown in the next tables, where the following notation is used: $S(type)$ and $C(type)$. S and C mean, respectively, Simple and Complex trap function, while *type* varies with respect to the parameter values used by simple and complex trap functions: type I ($\ell = 10, z = 3, a = 12, b = 6$), type II ($\ell = 20, z = 5, a = 20, b = 14$), type III ($\ell = 50, z = 10, a = 80, b = 39$), type IV ($\ell = 75, z = 20, a = 80, b = 54$), type V ($\ell = 100, z = 25, a = 100, b = 74$). For the complex trap function $z_1 = z$ and $z_2 = l - z_1$.

All the reported experimental results have been averaged over 100 independent runs, allowing at most $T_{max} = 5 \times 10^5$ fitness function evaluations (FFE)

Table 1. The best results obtained by IA with static, proportional, inversely proportional hypermutation and hypermacromutation with $M-$mutations ($M-mut$) strategy.

Trap	*Static*	*Proportional*	*Inversely*	*Hypermacro M-mut*
S(I)	**100**, *576.81*	**100**, *604.33*	**100**, *504.76*	**100**, *4334.94*
	$\tau_B = 25$, $c = 0.9$	$\tau_B = 100$, $c = 0.2$	$\tau_B = 5$, $c = 0.3$	$\tau_B = 1$
S(II)	**34**, *82645.85*	**100**, *50266.61*	**97**, *58092.70*	**5**, *5626.8*
	$\tau_B = 20$, $c = 1.0$	$\tau_B = 25$, $c = 0.8$	$\tau_B = 20$, $c = 0.2$	$\tau_B = 50$
S(III)	**0**	**0**	**0**	**5**, *174458.6*
				$\tau_B = 50$
S(IV)	**0**	**0**	**0**	**0**
S(V)	**0**	**0**	**0**	**0**
C(I)	**100**, *389.67*	**100**, *404.94*	**100**, *371.15*	**100**, *1862.09*
	$\tau_B = 100$, $c = 0.6$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 5$
C(II)	**100**, *36607.15*	**100**, *22253.70*	**100**, *44079.57*	**56**, *84001.29*
	$\tau_B = 15$, $c = 0.1$	$\tau_B = 25$, $c = 0.3$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 50$
C(III)	**1**, *15337.00*	**0**	**2**, *236484.50*	**1**, *185318*
	$\tau_B = 50$, $c = 0.7$		$\tau_B = 25$, $c = 0.1$	$\tau_B = 50$
C(IV)	**0**	**0**	**0**	**0**
C(V)	**0**	**0**	**0**	**0**

and with $c \in \{0.1, 0.2, ..., 1.0\}$, and $\tau_B \in \{1, 5, 10, 15, 20, 25, 50, 100, 150, 200, \infty\}$. The population size $d = 10$ and the duplication parameter $dup = 1$ were set to minimal values to stress the real searching capability of the used hypermutation operators. The Hypermacromutation operator is a random procedure independent from the parameter c, for this reason we vary only the τ_B parameter. Tables 1 and 2 show the best results obtained by IA in terms of Success Rate (SR) and Average number of Evaluations to Solutions (AES). Each entry in the tables indicate, in the first row, SR (in bold face) and AES (in italic), and in second row, the best parameter values that allowed the hypermutation operators to reach the best results. In table 1 we report the performances of IA when working with each hypermutation operator alone: static, proportional, inversely proportional hypermutation and hypermacromutation with $M-$mutations ($M-mut$) strategy (hypermacromutation performs all M mutations determined by the mutation potential). IA works better when it uses hypermacromutation. Indeed, this operator is the only one that solves the simple trap function type III and IV. The other operators are practically the same with the inversely proportional slightly better than static and proportional hypermutation.
Table 2 shows the results obtained by IA using both perturbation operators (columns $2-4$), while the first column reports the IA with hypermacromutation FCM (Stop at the first constructive mutation) only.
If we compare the hypermacromutation $M-$mut (table 1 fourth column) versus

hypermacromutation *FCM* (table 2 first column), it is clear how the *Stop at first constructive mutation* strategy improves the performance of the hypermacromutation operator and, in general, of all hypermutation operators. In fact, we adopt this strategy for all hypermutation operators, a simple scheme to prevent the premature convergence during the search process of IA. Moreover, the proportional hypermutation performs poorly whether or not combined with the hypermacromutation operator.
Finally, the usage of coupled operators, static and inversely proportional hypermutation with hypermacromutation (see table 2), is the key feature to effectively face the trap functions. The results obtained with this setting are comparable to the results gained in [8], where the authors, though, in their theoretical and experimental research work, use only cases C(I), C(II) and C(III) for the complex trap function.

Table 2. The best results obtained by IA with hypermacromutation FCM, and combinations of static, proportional, inversely proportional hypermutation and hypermacromutation FCM.

Trap	*Hypermacro FCM*	*Static+Macro*	*Proportional+Macro*	*Inversely+Macro*
S(I)	**100**, *1495.90*	**100**, *452.94*	**100**, *834.01*	**100**, *477.04*
	$\tau_B = 1$	$\tau_B = 25$, $c = 0.2$	$\tau_B = 50$, $c = 0.2$	$\tau_B = 15$, $c = 0.2$
S(II)	**28**, *64760.25*	**99**, *37915.17*	**100**, *51643.11*	**100**, *35312.29*
	$\tau_B = 1$	$\tau_B = 25$, $c = 0.1$	$\tau_B = 20$, $c = 0.7$	$\tau_B = 100$, $c = 0.2$
S(III)	**15**, *15677.53*	**100**, *18869.84*	**1**, *243038.00*	**100**, *20045.81*
	$\tau_B = 100$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 20$, $c = 0.2$	$\tau_B = \infty$, $c = 0.1$
S(IV)	**24**, *40184.83*	**100**, *37871.71*	**0**	**100**, *42082.00*
	$\tau_B = 200$	$\tau_B = 25$, $c = 0.1$		$\tau_B = 25$, $c = 0.2$
S(V)	**27**, *139824.44*	**100**, *78941.79*	**0**	**100**, *80789.94*
	$\tau_B = 1$	$\tau_B = 100$, $c = 0.1$		$\tau_B = 50$, $c = 0.2$
C(I)	**100**, *826.78*	**100**, *367.67*	**100**, *644.67*	**100**, *388.42*
	$\tau_B = 50$	$\tau_B = 10$, $c = 0.1$	$\tau_B = 10$, $c = 0.1$	$\tau_B = 10$, $c = 0.2$
C(II)	**96**, *54783.25*	**100**, *24483.76*	**100**, *23690.82*	**100**, *29271.68*
	$\tau_B = 15$	$\tau_B = 10$, $c = 0.2$	$\tau_B = 50$, $c = 0.3$	$\tau_B = 5$, $c = 0.2$
C(III)	**39**, *112533.18*	**27**, *172102.85*	**1**, *147114.00*	**24**, *149006.5*
	$\tau_B = 15$	$\tau_B = 20$, $c = 0.1$	$\tau_B = 50$, $c = 0.1$	$\tau_B = 20$, $c = 0.1$
C(IV)	**5**, *227135.80*	**2**, *99259.00*	**0**	**2**, *154925.00*
	$\tau_B = 15$	$\tau_B = 25$, $c = 0.5$		$\tau_B = 15$, $c = 0.4$
C(V)	**2**, *353579.00*	**0**	**0**	**0**
	$\tau_B = 15$			

4 Second test bed: the Protein Structure Prediction

A possible approach to model the protein folding problem is the well-known Dill's lattice model (or HP model) [9]. It models proteins as two-dimensional (2D) *self-*

avoiding walk chains of ℓ monomers on the square lattice (two resides cannot occupy the same node of the lattice). There are only two monomer types: the H and the P monomers, respectively for hydrophobic and polar monomers. In this model, each H–H topological contact, that is, each lattice nearest-neighbor H–H contact interaction, has energy value $\epsilon \leq 0$, while all other contact interaction types (H–P, P–P) have zero energy In general, in the HP model the residues interactions can be defined as follows: $e_{HH} = - \mid \epsilon \mid$ and $e_{HP} = e_{PH} = e_{PP} = \delta$. When $\epsilon = 1$ and $\delta = 0$ we have the typical interaction energy matrix for the standard HP model [9]. The native conformation is the one that maximizes the number of contacts H–H, i.e. the one that minimizes the free energy function. This model has a strong experimental justification. During the folding process of real protein the hydrophobic residues tend to interact with each other, forming the *hydrophobic kernel* of the native structure, while the hydrophilic resides are on the external surface of protein, forming the interface with the watery environment. The HP model has the great practical advantage of formalizing the protein primary structure as a binary sequence s of H's and P's (i.e., $s \in \{H, P\}^{\ell}$) and the conformational space as a square lattice. It is worth to say that is possible to extend the model on triangular 2D lattices and on 3D lattices. Finding the global minimum of the free energy function for the protein folding problem in the 2D HP model is NP-hard [10]. In general, the HP model is used by biologists to study the theoretical properties of the folding processes while computer scientists use its hard instances to analyze the performance of designed algorithms. In this section, we run the IA with the HP model as hard benchmarks. We used the first nine instances of the *Tortilla 2D HP Benchmarks*[3] to test the searching capability of the designed IA.

The input protein is a sequence $s \in \{H, P\}^{\ell}$, where ℓ, the length, represents the number of amino-acids. The B cell (or B cell receptor) is a sequence of *relative directions* [11] $r \in \{F, L, R\}^{\ell-1}$; where each r_i, is a relative direction with respect to the previous direction (r_{i-1}), with $i = 2, \ldots, \ell - 1$, (i.e., there are $\ell - 2$ relative directions) and r_1 the non-relative direction. Hence, we obtain an overall sequence r of length $\ell - 1$. The sequence r detects a *self-avoiding* conformation, i.e. a 2D conformation suitable to compute the energy value of the hydrophobic-pattern of the given protein. Hence $f(\boldsymbol{x}) = e$ is the energy of conformation coded in the B cell receptor $\boldsymbol{x}$, with $-e$ the number of topological contacts $H - H$ in the 2D lattice. It is worthwhile to note here, that all the implemented operators try to mutate each B cell receptor M times, maintaining the self-avoiding property.

The parameter surfaces of hypermutation operators. To understand the searching ability of single or combined hypermutation operators, when changing the parameter values, we performed a set of experiments on a PSP instance, *Seq2*, *hhpphpphpphpphpphpphpphh*, ($\ell = 24$ and minimum energy value known $E^* = -9$). We chose the instance *Seq2*, because *Seq1* is a relative easy instance. Indeed, for each combination of hypermutation operators there are many parameter

[3] *http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html*

settings which produce $SR = 100$ (experimental results not shown in this paper). The hydrophobic pattern *Seq2* is a good trade-off between complexity instance and time needed to perform a complete experimental analysis to discover the best parameter setting for each hypermutation operator. The duplication parameter *dup* varies from 1 to 10 and the constant c from 0.1 to 1.0. As a function of *dup* and c we show either the success rate (SR) or the average number of evaluations to solutions (AES). The 3D plots obtained are the characteristic *parameter surfaces* for each combination of the hypermutation operators.

We set the population size to a minimal value $d = 10$, to stress the property of each operator when working with few points in the conformational space. This strategy gives us a good measure of the "real" performance of the single hypermutation procedures. The aging parameter τ_B was set to 5. This value produces almost always the best convergence rate, or similar behavior for all hypermutation operators (simulation results not shown in this paper). Moreover, the *Termination_Condition()* function allowed at most $T_{max} = 10^5$ fitness function evaluations and we performed for each value pair of the parameters, 100 independent runs.

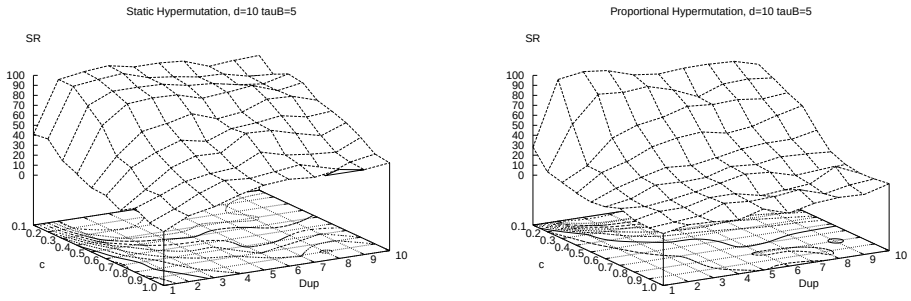

Fig. 1. SR as a function of the values *dup* and c for the Static Hypermutation Operator (left plot) and Proportional Hypermutation Operator (right plot) on sequence 2.

Using the SR and AES values, our experimental protocol has the following four objectives:

1. to plot the characteristic parameter surface of each hypermutation operator,
2. to analyze the joint effects of hypermacromutations and hypermutations,
3. to determine the best hypermutation operator or the best combination operators,
4. to find the best setting of parameter values for each operator and combination of operators, that is, the best delimited region on the parameter surfaces that maximize the SR value and minimize the AES value.

Figure 1 (left plot) shows the parameter surface of the static hypermutation operator; the region with high success rate values is for $dup \in \{2, \ldots, 10\}$ and

$c \in \{0.1, \ldots, 0.4\}$. In this area the static hypermutation has an average $SR > 80$. For $c > 0.5$ the SR decreases softly. IA obtains the best performance, $SR = 98$, for $dup = 3$ and $c = 0.1$ and a computational average effort of $AES = 29131,28$.

In figure 1 (right plot) we report the parameter surface of the proportional hypermutation operator. For this operator the region with high SR is a straight line, $c = 0.1$. For $c > 0.1$ the SR decreases quickly, with a high slope. For $dup = 3$ and $c = 0.1$ the IA reaches the highest success rate, $SR = 96$, with $AES = 32958,87$.

Figure 2 (left plot) shows the performance of the inversely proportional hypermutation operator. There is a large region with high SR values ($dup \in \{2, \ldots 10\}$ and $c \in \{0.2, \ldots, 1.0\}$), with an average $SR > 75$. The highest peak, $SR = 93$, is obtained for $dup = 4$ and $c = 0.5$ with $AES = 29721,4$.

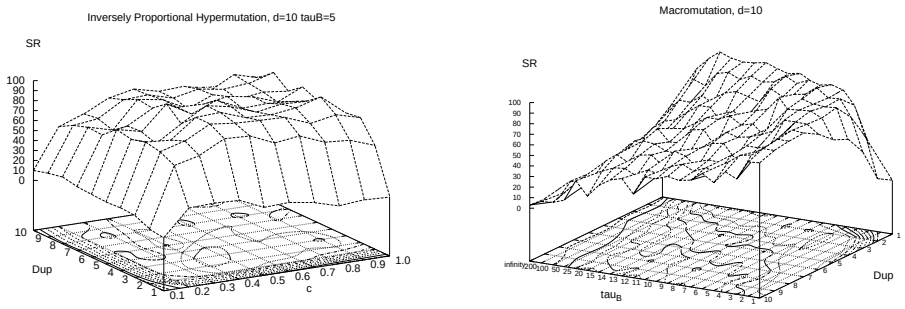

Fig. 2. SR versus parameter values *dup* and c for the Inversely Proportional Hypermutation Op. (left plot) and Hypermacromutation Op. (right plot) on sequence 2.

The parameters surface of the Hypermacromutation operator is shown in figure 2 (right plot). The operator does not use the parameter c, hence we vary the cloning parameter *dup* from 1 to 10 and the aging parameter τ_B in the set $\{1, 2, 3, \ldots, 15, 20, 25, 50, 100, 200, \infty\}$. We note that setting the parameter τ_B at a value higher than the possible number of generations, is equivalent to giving the B cell an infinite life, that is, we turn off the aging operator. When $\tau_B = \infty$ one has the worst SR values (2–8). For $dup = 1$ and $\tau_B \in \{6, \ldots, 50\}$ we have high SR values (with the best SR value 96 for $\tau_B = 20$). For $dup = 2$ we have high SR values when $\tau_B \in \{3, \ldots, 25\}$. Increasing *dup* we obtain high SR only for $\tau_B \in \{1, 2\}$.

To show the effects of combining the hypermacromutation operator with the above described three hypermutation operators, we overlap the two parameters surface generated by IA with just one hypermutation operator, with the results of combining hypermutation and hypermacromutation operators. Figure 3 shows the parameters surface of the Static Hypermutation Operator in terms of SR and AES. In the left plot, we illustrate how the combination of hypermacromutation

and static hypermutation outperforms the static hypermutation operator along, in the region where it performed poorly ($c \geq 0.5$).

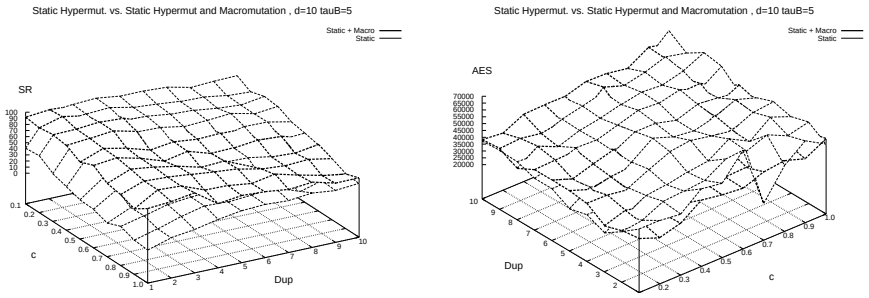

Fig. 3. SR (left plot) and AES (right plot) versus parameter values *dup* and c for the sequence 2. The surface parameters of Static Hypermutation operator and the combination of Static Hypermutation operator and Hypermacromutation Operators.

Moreover, the computational effort of the IA with both operators is smaller than the IA with only the static hypermutation operator (fig. 3 right plot). This is the general behavior of the designed IA when we use the combination of hypermutation and hypermacromutation operators (see fig. 4). The hypermacromutation operator makes the IA less sensitive to parameter values. It extends the region with high SR and raise the overall parameters surface. For the proportional hypermutation operator (see fig. 4 (left plot)), this phenomenon is particularly evident, in fact the operator alone, performs poorly. Whereas, when we add the hypermacromutation we observe a clear improvement in the IA performance.

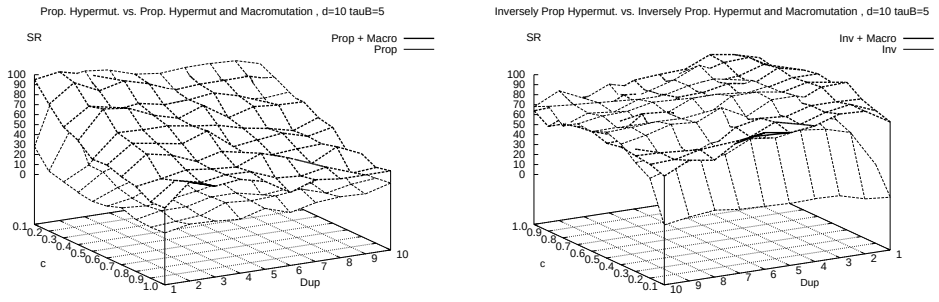

Fig. 4. Combination of hypermutation and hypermacromutation operators.

In the right plot, we show the surface parameters of inversely proportional hypermutation operator and the combination of inversely proportional hypermutation operator and hypermacromutation operators(upper surface): the hypermacromutation extends the region with high SR values and in particular improves the region where the inversely proportional hypermutation operator alone performed poorly ($c \in \{0.1, 0.2\}$ and $dup \in \{1, 2, 3\}$).

Experimental Results. To assess the overall performance of all hypermutation operators, we tested them using the tortilla benchmarks. Using the region with high SR value for the protein sequences *Seq1* and *Seq2*, we tried to predict suitable parameter values for each hypermutation operator. In practice, we take the centroid, a pair (dup, c), that maximizes SR and minimizes the AES value. Table 3 shows the SR and AES values of each operator for the protein sequences *Seq1*, *Seq2*, *Seq3*, *Seq6* and *Seq9* (the "simple" protein instances). Table 4 shows the SR and AES values of each operator for the protein sequences *Seq4*, *Seq5*, *Seq7* and *Seq8* (the "complex" protein instances).

Table 3. The SR and AES of all hypermutation operators and their combination on the sequence $1, 2, 3, 6, 9$. The acronym *b.f.* stands for *best found* (result b.f. by IA).

Sequence	1	2	3	6	9
Length	20	24	25	50	20
E^*	-9	-9	-8	-21	-10
Static $(5, 0.1)$	**100**	**100**	**100**	**76.6**	**100**
	15915,63	*29702,6*	*96211,9*	*425199,82*	*26004,86*
Prop $(3, 0.1)$	**100**	**100**	**100**	*b.f. -19*	**100**
	14954,33	*36254,56*	*132119,64*		*21775.06*
Inv Prop $(5, 0.6)$	**100**	**100**	**83,33**	**40**	**100**
	21252	*48756,56*	*135694,32*	*513626,33*	*18547,96*
Macro $(1, \tau_B = 15)$	**100**	**100**	**100**	**16,67**	**100**
	25418,83	*39410,9*	*79592,1*	*469941,2*	*27852,13*
Static+Macro $(3, 0.3)$	**100**	**100**	**96.66**	**80**	**100**
	24362,93	*33496,83*	*89598,31*	*394373,08*	*29148,5*
Prop+Macro $(2, 0.1)$	**100**	**100**	**100**	**30**	**100**
	19355,73	*25627,5*	*69555,07*	*548883,44*	*30683,47*
Inv P+Macro $(2, 0.4)$	**100**	**100**	**100**	**53,33**	**100**
	14443,7	*39644,1*	*95146,97*	*538936,43*	*17293,87*

Near the operator names, we put the parameter values (dup, c) with the best average performance for all the benchmark instances. The table contains the SR and AES values. We performed for each instance, 30 independent runs and allowed at most $T_{max} = 10^6$ fitness function evaluations (FFE). All operators reach $SR = 100$ for the protein sequences *Seq1*, *Seq2* and *Seq9* (the simplest instances of benchmark). For the protein configuration *Seq3*, the Inversely Pro-

portional operator and the combination of Static Hypermutation and Hypermacromutation operator obtain respectively $SR = 83,3$ and $SR = 96,6$. Most likely these operators, given the current parameter settings, need a higher T_{max} value. For protein sequence *Seq6*, only three operators obtain a SR value higher than 50% (Static, Static and Hypermacromutation, and Inversely Proportional and Hypermacromutation). We note that only the proportional hypermutation operator does not reach the minimum energy value $E^* = -21$, since it gets trapped in a local minimum $E^* = -19$. In table 4 we show the overall perfor-

Table 4. The SR and AES of hypermutation operators and their combination on the sequence 4, 5, 7, 8. The acronym *b.f.* stands for *best found* (result b.f. by IA).

Sequence	4	5	7	8
Length	36	48	60	64
E^*	-14	-23	-36	-42
Static $(5, 0.1)$	**20** *304971.33*	*b.f. -22*	*b.f. -35*	*b.f. -39*
Prop $(3, 0.1)$	**3,33** *355512*	*b.f. -19*	*b.f. -32*	*b.f. -31*
Inv Prop $(5, 0.6)$	**10** *462321*	*b.f. -22*	*b.f. -34*	*b.f. -38*
Macro $(1, \tau_B = 15)$	**16,67** *466177,4*	**6,67** *483651,5*	*b.f. -34*	*b.f. -36*
Static+Macro $(3, 0.3)$	**23,33** *347078,14*	*b.f. -22*	*b.f. -34*	*b.f. -38*
Prop+Macro $(2, 0.1)$	**26,66** *375700,25*	*b.f. -22*	*b.f. -34*	*b.f. -37*
Inv P+Macro $(2, 0.4)$	**23,33** *388323,43*	*b.f. -22*	*b.f. -34*	*b.f. -39*

mance on the complex instance of the tortilla benchmark. All operators reach the minimum energy value $E^* = -14$ for the protein conformation *Seq4*, on average with $SR \in \{3, 33, \ldots, 26, 66\}$. There is a clear improvement when IA runs using a combination of hypermutation operators and the hypermacromutation operator. In the remaining protein sequences, no operator obtains the minimum known energy value E^*, except for the *Seq5* where the hypermacromutation operator reaches $E^* = -23$ with $SR = 6,67$. In the sequence *Seq7* only the Static Hypermutation operator overtakes the conformation with energy value nearest the best known $E^* = -36$. Whereas for the instance *Seq8*, the Static Hypermutation operator and the combination of Inversely Proportional Hypermutation and Hypermacromutation operators achieve the best value ($b.f. = -39$). By inspecting the performance of each operator, it emerges how the static hypermutation, hypermacromutation and the combination of inversely proportional hypermutation and hypermacromutation achieve the best results.

In [12] is reported the comparisons with the state-of-art algorithms for the 2D HP PSP problem. The shown results suggest that the IA with hypermacromutation and aging operators is comparable to and, in many protein instances, outperforms the best algorithms.

5 Conclusions

In this paper we propose a modified version of an IA for a toy problem, trap function, and a NP-complete problem, 2D HP protein structure prediction problem, using simple cloning operator, static pure aging and seven types of different hypermutation operators. Moreover, the used standard IA adopts the *Stop at first constructive mutation strategy* to improve the performance of hypermacromutation operator and in general of all hypermutation operators; we set this strategy for all hypermutation operators, a trick to try to prevent the premature convergence during the search process. For both problems the usage of coupled operators, static and inversely proportional hypermutation with hypermacromutation is the key feature to effectively face the different computational problems; while the proportional hypermutation operator with or without hypermacromutation almost always performs poorly. In particular, for the NP-complete problem, for each hypermutation operator we determined the characteristic parameter surface and the best delimited region on the parameter surfaces that maximizes the SR value and minimizes the AES value. This region has been used to predict the best parameter value setting for all operators and their combinations. When overlapping the parameter surfaces with and without hypermacromutation, we discover that this new kind of perturbation operator makes the IA less sensitive to parameter values setting, and, in general, it improves the performance in terms of SR and AES values.

References

1. Cutello V., Nicosia G.: The Clonal Selection Principle for in silico and in vitro Computing. In L. N. de Castro and F. J. Von Zuben editors, Recent Developments in Biologically Inspired Computing, (to appear) (2004)
2. Dasgupta D. (ed.): Artificial Immune Systems and their Applications. Springer-Verlag, Berlin, Germany (1999)
3. De Castro L. N., Timmis J.: Artificial Immune Systems: A New Computational Intelligence Paradigm. Springer-Verlag, London, UK (2002)
4. Cutello V., Nicosia G., Pavone M.: A Hybrid Immune Algorithm with Information Gain for the Graph Coloring Problem. GECCO '03, Lecture Notes in Computer Science, pp.171-182, 2723 (2003)
5. Goldberg D. E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Vol. 7, Boston (2002)
6. De Castro L. N., Von Zuben, F. J.: Learning and optimization using the clonal selection principle. IEEE Trans. on Evol. Comp., 6(3), pp. 239-251, (2002)
7. Prugel-Bennett A., Rogers, A.: Modelling GA Dynamics. Proceedings Theoretical Aspects of Evolutionary Computing, Springer, (2001)

8. Nijssen S., Back, T.: An analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions. IEEE Trans. on Evol. Comp., 7(1), pp.11-22, (2003)
9. Dill K. A.: Theory for the folding and stability of globular proteins. Biochemistry, 24(6), pp. 1501-9 (1985)
10. Crescenzi P., Goldman D., Papadimitriou C., Piccolboni A., Yannakakis M.: On the complexity of protein folding. J. of Comp. Bio., 5(3), pp. 423-466 (1998)
11. Krasnogor N., Hart W. E., Smith J., Pelta D.A.: Protein Structure Prediction with Evolutionary Algorithms. GECCO '99, 2, pp. 1596-1601, Morgan Kaufman (1999)
12. Cutello V., Nicosia G., Pavone M.: An Immune Algorithm with Hyper-Macromutations for the 2D Hydrophilic-Hydrophobic Model. CEC'04, 1, pp. 1074-1080, IEEE Press (2004)