L.M. Rocha and J. Kaur [2007]."Genotype Editing and the Evolution of Regulation and Memory". *Proceedings of the 9th European Conference on Artificial Life*. LNAI, Springer, **4648**: 63-73.

Genotype Editing and the Evolution of Regulation and Memory

Luis M. Rocha and Jasleen Kaur

School of Informatics, Indiana University
Bloomington, IN 47406, USA
rocha@indiana.edu
http://informatics.indiana.edu/rocha

Abstract. Our agent-based model of genotype editing is defined by two distinct genetic components: a coding portion encoding phenotypic solutions, and a non-coding portion used to edit the coding material. This set up leads to an indirect, stochastic genotype/phenotype mapping which captures essential aspects of RNA editing. We show that, in drastically changing environments, genotype editing leads to qualitatively different solutions from those obtained via evolutionary algorithms that only use coding genetic material. In particular, we show how genotype editing leads to the emergence of regulatory signals, and also to a resilient memory of a previous environment

1 Introduction: RNA Editing

RNA Editing [Bass, 2001] refers to the post-transcriptional alteration of genetic information. It occurs in various forms such as insertion, deletion, or substitution. It can be implemented via non-coding RNAs (ncRNAs) such as *guide RNA's* or via enzymes (e.g. *adenosine deaminase acting on RNA* (ADAR), also known as *RNA Editase*) In either case, genetic information is altered after transcription and before translation (for an overview see [Huang et al., 2007]).

Previously we quantitatively established the advantages of genotype editing against the canonical evolutionary algorithm in various static and dynamic environments (e.g. [Huang et al., 2007]). Here, using our *Agent-Based Model of Genotype Editing* (section 2) in drastically changing environments (section 3), we focus instead on the qualitatively different evolutionary solutions attainable via genotype editing. Specifically, we show how genotype editing leads to the emergence of regulatory signals that allow agents to better adapt to radically different environments (section 4). We also show how the inclusion of non-coding genetic material, with the function of editing coding material, allows agents to evolve a memory of previous environments—a capacity not attainable by the canonical evolutionary algorithms which use only coding genetic material (section 5).

2 Modeling Genotype Editing

The Genetic Algorithm (GA) [Holland, 1975] is an idealized model of natural selection—and the canonical evolutionary algorithm. In a traditional GA, the code between genotype and phenotype is a direct and unique mapping . In biology, however, before a gene is translated into a protein it may be altered, namely by functional or non-coding RNA (ncRNA) used for editing or other regulatory functions. To study and exploit the biological principle behind RNA

Editing, we have introduced an *agent-based model of genotype editing* (ABMGE) [Rocha et al., 2006]. In this model, the agents in the population are defined by an artificial genome that contains functionally and operationally distinct *coding* and *non-coding* components: respectively, the *codome*, encoding solutions to a particular fitness function or environment, and the *editome*, producing editors which act on the coding component. Our goal is to understand the influence of editing, as a genomic (pre-translation) phenomenon, in the evolutionary process. Therefore, we have (1) explicitly separated an editome from a codome to better test its relative importance (rather than attempting to have it emerge from a common artificial genome), and (2) we strip our model of any post-translation dynamics (i.e. development) or epigenetic phenomena. We understand that both of these design choices may be unrealistic, but they are a reasonable and necessary starting point to understand editing as a genomic phenomenon.

Figure 1 depicts an agent in the ABMGE. In each generation, the coding component of an agent's genotype, the *codotype* may be **stochastically** edited by the agent's non-coding genotype, its *editype*, and produce a phenotype different from what is encoded in the codotype. The *codotype* of an agent is a n-bit string (or sequence) S, whereas the *editype* is a family of r editors each defined by a 3-tuple (E_j, F_j, v_j). E_j is a m-bit string ($m \ll n$), which may bind to S by exactly matching a substring (bit-pairing). F_j is the *editing function* that specifies how the editor edits the codotypes it matches, e.g. by inserting into or deleting bits from S. In this model, the length n of the codotype is fixed. Therefore, when x bits are inserted, the sequence is shifted to the right discarding x bits on the right end of the string. When x bits are deleted, the sequence is shifted to the left, and x bits are randomly generated on the right end of the string. v_j is the *concentration* of the editor and denotes the probability that the codotype string S encounters editor j before translation in each generation. When an editor j encounters the codotype string S (with probability v_j), it checks the codotype from left to right, one bit position at a time, performing its editing function F_j every time a match occurs between its editor substring E_j and S. Thus, **the same codotype may be edited differently since editor concentration is a stochastic parameter**. Indeed, the same genotype may produce different phenotypes in different generations, or even in the same population if clones exist. For instance, a codotype may fail to be edited in one generation (especially when the concentration of editors is small), edited by a single editor in the next, or edited by every editor in yet another generation. Table 1 describes the ABMGE algorithm.

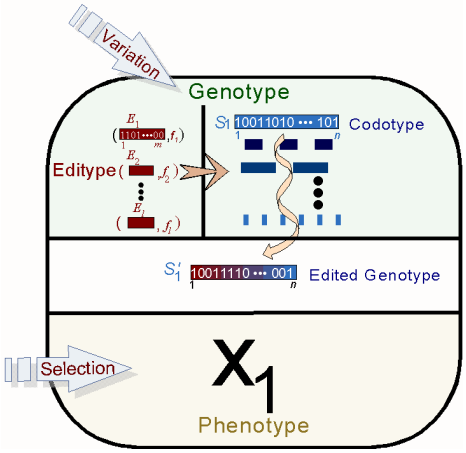

Fig. 1. Individual Agent in the ABMGE.

Notice that only the codome is used to encode phenotypic attributes. The editome is, in this sense, "non-coding"; its role is to change genetic information ontogenetically, more specifically, to model the post-transcriptional, pre-translation process of genotype editing. Finally, also note that **edits are not inheritable.** While agent fitness is calculated using the phenotype produced from the edited codotype, what is inheritable, and subjected to variation, is the unedited genotype (codotype plus editype).

Table 1. The ABMGE Algorithm

1. Randomly generate an initial population of l agents, each agent consisting of a codotype (a n-bit string) and an editype (a family of r editors (E_j, F_j, v_j)).
2. Edit each agent's codotype S: apply each editor with probability v_j; If E_j matches S at any position running from left to right, edit S with function F_j
3. Evaluate the fitness of the edited genotype of each agent.
4. Repeat until l offspring have been created.
a. select a pair of parent agents for mating;
b. apply codotype variation operators (mutation and crossover);
c. apply editype variation operators (editor mutation and crossover).
5. Replace the current population with the new one and go to step 2.

When two parents are selected for reproduction in our algorithm (step 4 in table 1), in addition to variation of codotypes as it is commonly done in the GA (mutation and crossover), the editype is also subjected to variation. In the current implementation, variation is only applied to editor strings E_j, while editing functions F_j and concentrations v_j remain unchanged. We will consider variation on these parameters in future work, though these were fixed not only to reduce the number of evolving parameters, but also to model physical characteristics of editors not amenable to evolution.

Editype mutation is implemented on editor bit-strings as usual: with a bit-flipping probability, P_{EdMut}, for each bit of an editor string per generation—P_{EdMut} is independent from the codotype bit-mutation probability: P_{Mut}. *Editype crossover*, is implemented as an exchange of editors between a pair of parent agents. We start with two parent agents a_1 and a_2, with r_1 and r_2 editors in their editypes, respectively. From this pair of parent agents, two offspring agents, a_3 and a_4, are produced whose editypes also contain r_1 and r_2 editors, respectively. However, x editors, chosen randomly from the editype of each agent, are swapped between the parent agents to produce the offspring, where $x \in [1, MIN(r_1, r_2)]$. Editype crossover occurs with a probability $P_{EdCross}$, which is independent from the codotype (one-point) crossover probability, P_{Cross}.

3 Testing on Drastic Environmental Oscillations

In previous work we showed that genotype editing outperforms the GA in many static and dynamic environments [Huang et al., 2007]. Our goal here is to understand how this advantage comes about, especially under drastic environmental changes. Therefore, we focus on a toy fitness function to better understand the adaptations enabled by genotype editing.

The *small Royal Road* SRR_1, as depicted in Table 2 is a miniature of the class of the "Royal Road" functions [Mitchell et al., 1992]. This function is an

Table 2. Small royal road function SRR_1

t_1	= 11111********************************;	c_1	= 10
t_2	= *****11111*************************;	c_2	= 10
t_3	= **********11111*******************;	c_3	= 10
t_4	= ***************11111**************;	c_4	= 10
t_5	= ********************11111*********;	c_5	= 10
t_6	= ************************11111*****;	c_6	= 10
t_7	= ****************************11111**;	c_7	= 10
t_8	= ********************************11111;	c_8	= 10

idealized fitness environment defined by a set of schemata $T = \{t_1, \ldots, t_8\}$. The fitness of a bit string (codotype) S is defined as $F(S) = \sum_{t \in T} c_t \sigma_t(S)$, where each c_t is the value assigned to schema t as defined in Table 2; $\sigma_t(S) = 1$ if schema t exists in S and 0 otherwise. The single optimum fitness for SRR_1 is obtained by the string with 40 1's, and its value is 80. Consider another Small Royal Road function, SRR_0, in which each schemata is comprised of five 0's rather than 1's as SRR_1 above, but with all other parameters the same as SRR_1. With these two functions, we create the *oscillatory royal road* (ORR), which oscillates between SRR_1 and SRR_0 at every p generations.

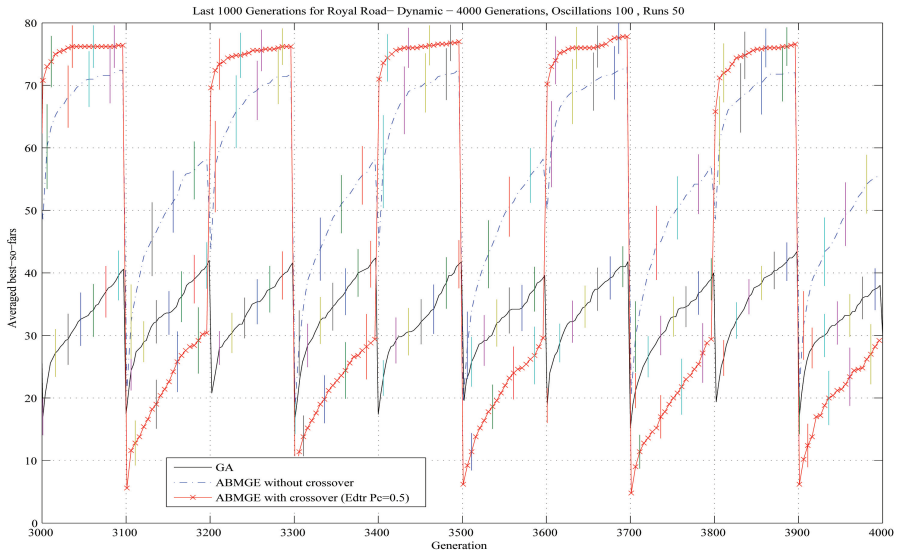

Fig. 2. Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on ORR, $p = 100$, 50 runs, 4000 generations (last 1000 shown).

We contrasted the GA with two versions of the ABMGE: with (ABMGE^C) and without editype crossover. The experiments we report here with ORR use binary tournament selection, and a population of 40 agents over 4000 generations for 50 runs. Codome variation (in both the ABMGE and the GA) is implemented with one-point crossover and mutation rates of $P_{Cross} = 0.7$ and $P_{Mut} = 0.005$, respectively—the best values we had previously found for the GA (see [Huang et al., 2007]). In every run, all editome parameters (of the ABMGE) are randomly generated as follows: $r \in \{1, \cdots, 5\}$ (number of editors for each agent); $m \in \{2, \cdots, 4\}$ (size of editor strings E_j); $v_j \in [0, 1]$ (editor

concentration); F_j, the editor functions, insert or delete $x \in \{1, \cdots, 3\}$ bits which are randomly generated when the editor is created but fixed thereafter for each run. In addition to various editype mutation and crossover probabilities ($P_{EdMut} \in \{0.01, 0.05\}$ and $P_{EdCross} \in \{0, 0.3, 0.5, 0.7, 0.9\}$), we tested different oscillation periods ($p \in \{50, 100, 200, 250\}$). Figure 2 depicts the mean best-so-far fitness[1] (last 1000 generations) for $P_{EdMut} = 0.05$ and $P_{EdCross} = 0.5$ and period $p = 100$. Table 3 details the performance of the three algorithms tested, at the end of the last two environment oscillations (generations 3899 and 3999).[2]

Table 3. Mean fitness and 95% confidence interval for 50 runs of the GA, $ABMGE$, and $ABMGE^C$ with the ORR at $p = 100$, at the end of the last two oscillations.

		GA		ABMGE		ABMGE^C	
Generation	Function	Mean Fit.	Conf. int.	Mean Fit.	Conf. int.	Mean Fit.	Conf. int.
3899	SRR_1	43.4	3.18	72	4.22	76.6	2.9
3999	SRR_0	38	3.3	55.6	4.56	29.2	5.1

The GA degrades its performance in time; at the end of 4000 generations its mean performance on both environments eventually reaches the same level (close to 40). This means that the GA ultimately converges to a population of genotypes with a balanced number of all-1 and all-0 schema. Table 4 depicts the best genotypes attained by the GA in both environments (SRR_1 and SRR_0) at the end of a run. This behavior highlights the difficulty a canonical evolutionary algorithm faces in a drastically changing environment.

Table 4. Example of best genotypes produced by the GA in a single run

Generation	Function	Genotype	Fitness
3899	SRR_1	11111-11111-11111-01011-11111-00000-11111-00000	50
3999	SRR_0	01110-11111-01111-00000-11111-00000-11011-00000	30

As for the ABMGE, the version with both editype crossover and mutation (ABMGE^C) performs best on the first fitness environment (SSR_1), and every time it repeats, but it performs rather poorly on the second environment—slightly worse than the GA as generations progress (figure 2). In contrast, the ABMGE with editype mutation alone, is almost as good as the ABMGE^C on the first environment, but much better on the second environment where it progressively improves its performance—well beyond that of the regular GA (figure 2). As it can be seen in table 3, both versions of the ABMGE are quite significantly better than the GA at the end of the last time SSR_1 occurs (generation 3899); the mean fitness of both versions of the ABMGE are not significantly different. The ABMGE with mutation only is, however, quite significantly better than the GA and the ABMGE^C at the end of the last time SSR_0 occurs (generation 3999). From these results, we conclude that editype mutation alone seems to offer a much better agent architecture in drastic environmental changes. Indeed, the ABMGE with editype mutation alone, leads to agents which do well on both dramatically different fitness landscapes—as opposed to the GA which ultimately settles to agents that are mediocre in both. We discuss next why and

[1] Fitness of the best individual that has been seen thus far in a given environment period; vertical bars represent 95-percent confidence intervals.

[2] Results for $p = \{50, 200\}$ at `http://informatics.indiana.edu/rocha/editing`.

how the ABMGE manages to outperform the GA and why the two variations of the ABMGE perform so differently in this dynamic environment.

4 Evolving simple regulatory signals

Table 5 depicts one of the best agents evolved by generation 3899 (SRR_1) with the ABMGE with mutation alone. The second editor of this agent, with substring $E_2 = \{11\}$ inserts two 1's after matching the codotype: ($\{11\} \rightarrow 11$). Once the substring $\{11\}$ is found in the codotype S, every bit in the remaining portion of S, to the right of the match position, is edited to "1", as 11 is guaranteed to occur at the next bit position for E_2 to match again. Moreover, if 11 occurs in the first 2 bit positions of the codotype, the entire codotype is edited to an all-1 bit-string. We refer to this process as *repetitive massive insertion* (RMI), which is similar to the massive u-insertion observed in Trypanosoma in nature [Bass, 2001]. Notice that since this editor exists with a high concentration (0.966) and the first two positions of the codotype are 11, most of the time (97%) the codotype is massively edited into the maximum fitness value of SRR_1.

Table 5. Example Agent evolved at generation 3899 (end of last SSR_1 period) with the $ABMGE$ with editype mutation alone.

Codotype	11011-10000-01000-01001-00000-10100-00000-11000		
Editype	Editor Subtrings Functions Concentrations	{000} insert 1 bit: {0} 0.383	{11} insert 2 bits: {11} 0.966
Edited Genotype	11111-11111-11111-11111-11111-11111-11111-11111		
Comparison	Hamming Distance = 28	Unedited Fitness = 0	Edited Fitness = 80

Once the environment changes to SRR_0, this agent evolves to deal with the completely new environment quite effectively. Table 6 describes one of its descendants in the last generation of the run (at the end of SRR_0). We can see that the editors are the same, but now the codotype has changed to contain more 0's, and no $\{11\}$ substring. This allows the first insertion editor ($\{000\} \rightarrow 0$) to come into action while simultaneously preventing the second editor to act. The first editor, also using RMI, converts to 0 every allele from the match position all the way to the right end of the codotype, as 000 is guaranteed to occur at the next position. Therefore, if the codotype contains 000 in its first 3 bit positions, it is edited to an all-zero bit-string (the maximum fitness phenotype in SRR_0).

Table 6. Example Agent evolved at generation 3999 (end of last SSR_0 period) with the ABMGE with editype mutation alone; descendent from agent in table 5

Codotype	00000-00000-00000-00000-00000-00000-01000-00100		
Editype	Editors Functions Concentrations	{000} insert 1 bit: {0} 0.383	{11} insert 2 bits: {11 0.966
Edited Genotype	00000-00000-00000-00000-00000-00000-00000-00000		
Comparison	Hamming Distance = 2	Unedited Fitness = 60	Edited Fitness = 80

The way the agents of tables 5 and 6 use RMI, makes the first (leftmost) bit positions of their codome most important. Indeed, without us pre-specifying such a role for these bits, simple "promoter" or "regulatory" signals emerge in agents with genotype editing. With the appropriate signal at the start of the

codotype ({000} or {11}), any codotype can be edited to produce an all-0 or all-1 phenotype. Moreover, once RMI occurs via one of the editors, the other editor can no longer act as it will no longer find matching substrings. Therefore, evolutionary pressures on the codotype in environment SRR_0 will quickly lead to the appearance of 000 "regulatory" signals towards the left end of the codotype, and when in environment SSR_1 to the appearance of 11 such signals.

Notice that if editors were equally likely, all bit positions to the right of these "regulatory" signals would be largely neutral. But since editors are randomly generated at the start of the simulation, this is rarely the case. In the case of the agents of tables 5 and 6, the first editor has a lower concentration (0.383) than the second (0.966) so it will lead to RMI much less often. Evolution compensates this with a bias towards more zeroes in the codotype; in environment SRR_0 (table 6), the fitness of the unedited codotype is fairly high (60) because it contains a large majority of zeroes. Thus, even if editing does not occur, the agent does well in the SRR_0 environment. Moreover, even in environment SRR_1, the codotype of the agent (in table 5) contains a majority of 0's (28 0's for 12 1's). As long as the "promoter" signal {11} is present in the leftmost bits, the overwhelming majority of the time (97%), the second editor will lead to the optimal phenotype in this environment. This makes it easier, once in environment SRR_0 again, to drfit to a majority zero codotype with no {11} signals.

It is interesting to note at this point, that with the oscillation of environments at every 100 generations, the GA with the same parameters gets stuck on evolving agents with a balanced number of schema with 1's and 0's—and thus mediocre fitness values in both environments. In contrast, the emergence of regulatory signals on the left side of the codotype of the ABMGE agents, allows them to quickly produce maximum (or at least high) fitness phenotypes in both environments. This way, genotype editing as modeled here, leads to the emergence of a functionally distinct role for a small substring of the codotype of agents. In this case, with the operational constraints of left to right decoding, we observe that the first few bits of the codotype instantiate a regulatory box whose (allele) value or signal leads to completely different phenotypes able to cope well with the two drastically different environments. When environments change, evolution only needs to "re-write" these signals, rather than the entire codotype. It is important to emphasize that the regulatory-signal behavior of the agents of tables 5 and 6 is not rare. RMI is observed in most best agents evolved in these conditions, with some variations (such as editor mutation knocking out detrimental editors in the appropriate environment). Due to space restrictions we do not show additional agents here, as well as a discussion of emergent signals in other fitness environments which we will leave for future work.

5 Memory and the value of "junk" non-coding genotype

Another interesting behavior observed with both versions of the $ABMGE$ is the quick recovery of a high level of performance, every time the environment changes back to the first environment presented to the population (SRR_1). Table 7 shows the mean best-so-far fitness at the last two environment transitions. When the environment changes from SRR_0 to SRR_1 (generation 3800), the traditional

GA achieves a mean best-so-far performance of only 19.4, whereas the $ABMGE$ without editype crossover achieves a significantly higher 48.6, which is in turn significantly lower than the mean best-so-far performance of 65.8 achieved by the $ABMGE^C$ with editype crossover; this behavior is observed throughout most of the simulation every time the first environment repeats (figure 2). This indicates that an editome enables the agent population to preserve a ***memory*** of the first environment it encountered—though editype crossover preserves such memory much more effectively. Interestingly, once the environment changes from SRR_1 to SRR_0 (table 7), the $ABMGE^C$ observes the worst performance setback to 6.2 (from 76.6, in table 3), which is significantly lower than both the GA (17.2 from 43.4) and the $ABMGE$ with editype mutation alone (21.8 from 72). Thus, while it is clear that genotype editing allows agents to evolve a memory mechanism of the first environment, we still need to investigate why editype crossover leads to such a poor performance on the second environment?

Table 7. Mean fitness and 95% confidence interval for 50 runs of the GA, $ABMGE$, and $ABMGE^C$, at the start of the last two environment oscillations.

		GA		ABMGE		ABMGE^C	
Generation	Transition	Mean Fit.	Conf.	Mean Fit.	Conf.	Mean Fit.	Conf.
3800	$SRR_0 \rightarrow SRR_1$	19.4	2.40	48.6	7.83	65.8	5.86
3900	$SRR_1 \rightarrow SRR_0$	17.2	3.30	21.8	5.38	6.2	2.14

After observing the best agents evolved, we conclude that the $ABMGE^C$ performs rather poorly on the second environment because it is particularly successful on the first—as it also is in most static (more complex) environments already tested [Huang et al., 2007]. Indeed, our editype crossover is especially good at spreading the best editors discovered through the entire population. Since editype crossover allows agents to swap editors, it quickly leads to the evolution of agents which contain exclusively very good editors for the first environment encountered. Table 8 shows a typical best agent at generation 3899 (the end of the last time SRR_1 occurs). As we can see, every single editor this agent contains is capable of RMI and occurs in high concentration. Therefore, the overwhelming majority of the time, the codotype is edited into the maximum fitness configuration (all-1).

Table 8. Agent at generation 3899 (end of last SSR_1 period) evolved with $ABMGE^C$.

Codotype	11110-10101-01111-00001-00001-10010-01000-00010			
Editype	Editors	{111}	{111}	{111}
	Functions	ins 1 bit: {1}	ins 1 bit: {1}	ins 1 bit: {1}
	Concentrations	0.841	0.866	0.885
Edited Genotype	11111-11111-11111-11111-11111-11111-11111-11111			
Comparison	Hamm. Dist. = 23	Unedited Fitness = 0		Edited Fitness = 80

Once the environment switches to SRR_0, however, because the population of agents contains only editors that are particularly suited for the first environment, and since editor functions and concentrations are fixed at the start of each run, the best the $ABMGE^C$ can do is to evolve agents that knock-off the editors (via editor mutation) or mutate the codotype to be immune to them. Such an agent (descending from the agent in table 8) is displayed in table 9. This

behavior is quite different from what we observed with the $ABMGE$ without editype crossover in section 4. Indeed, because in that case agents cannot swap editors, many useless or neutral editors hitchhike with advantageous ones in the first environment. Later on, when the environment changes, these unused editors may become useful in the new environment (see agents of tables 5 and 6).

Table 9. Example Agent descending from agent in table 8 evolved at generation 3999 (end of last SSR_0 period) with the $ABMGE^C$.

Codotype	11111-00000-00000-00000-00000-00110-00000-00000			
Editype	Editors	{101}	{101}	{101}
	Functions	ins 1 bit: {1}	ins 1 bit: {1}	ins 1 bit: {1}
	Concentrations	0.841	0.866	0.885
Edited Genotype	11111-00000-00000-00000-00000-00110-00000-00000			
Comparison	Hamm. Dist. = 0	Unedited Fitness = 60		Edited Fitness = 60

Hitchhiking editors are a type of "junk" material in the genotype of agents. In the first environment, only some of the editype (the useful editors) gets used. Later on, when the environment changes, new environmental pressures can turn "junk" editype into useful editing material. Interestingly, while the editors that are useful in the first environment may get knocked-off or ineffective in the second, once the environment changes again, they can quickly become useful once more—thus granting the $ABMGE$ without editype crossover a means to evolve agents with a quickly recoverable memory of both environments.

6 Discussion

Ours is an effort to investigate organization principles enabled by non-coding DNA in general, and genotype editing in particular. Our goal is above all to understand how genotype editing works and what kind of search process it leads to. Here, using only the ORR function, we report three novel observations:

1. **The evolution of "promoter signals"** for editing regulation (section 4), which in this case resulted in massive insertion reminiscent of massive U-insertion RNA editing in Trypanosomes [Bass, 2001]. The emergence of a "promoter box" in the genome of our agents shows us that non-coding RNA may have played an essential role in the origin of gene regulation.
2. **The emergence of memory of previous environments.** Whereas the GA must start over every time the environment changes, agents with genotype editing quickly recover good levels of performance once previous environments return (section 5). Thus, agents with genotype editing are better equipped to deal with changing environments.
3. **The value of redundancy or "junk" editome hitchhiking** (section 5). When variation and selection of editypes is very effective (e.g. our editype crossover mechanism), only the best editors for a given environment survive leading to the best memory of the first environment encountered, but poor performance once the environment changes. But when "junk" editome material is allowed to hitchhike, it grants agents additional genetic material for future regulation in the second environment.

One could argue that that editors leading to RMI are useful in the case of our ORR fitness function, but not necessarily in other fitness functions. In this

case, the optimal phenotypes are obtained with all-zero or all-one codotype configurations. Therefore, they are especially amenable to the repetitive insertion that evolution exploited. Nonetheless, it is obvious that any other repetitive pattern (e.g. 1010101...) can be reached in the same manner. Moreover, in principle, a large enough set of editors can edit a given codotype into any desirable sequence. Therefore, genotype editing is by no means restricted to the sort of homogeneous repetitive insertion useful for the *ORR*. One can easily conceive a set of editors that use repetitive insertion only in a portion of the codotype in tandem with more localized and specialized edition (namely using deletion or non-repetitive insertion). Indeed, in our previous work we showed that the AB-MGE is also advantageous in many other dynamical environments considerably more complicated than the *ORR* [Huang et al., 2007]. In such cases, genotype editing may have allowed evolution to discover regulatory signals leading to more sophisticated repetitive behavior other than "all-0" or "all-1". We are pursuing a detailed analysis of editing in such functions to report in future publications.

We conclude that genotype editing offers a significantly distinct, and evolutionarily advantageous biological design principle. This advantage is particularly interesting in dynamic environments as agents become better equipped (using emergent regulatory signals and memory) to deal with changing conditions. While our highly idealized model does not capture the reality of biology, it implies that the process of RNA editing in nature is, likewise, advantageous in evolution. Our results emphasize the importance of genetic regulation by non-coding genetic components, offering another piece of conceptual evidence that it plays an essential role in phenotypic development and evolution.

7 Acknowledgements

We are grateful to Chien-feng Huang and Ana Maguitman for sharing code used on some of the experiments. Luis M. Rocha is partially funded by NSF (BCS-0527249). We are also grateful to the FLAD Computational Biology Collaboratorium at the Gulbenkian Institute in Oeiras, Portugal, for providing facilities used to conduct part of this research.

References

[Bass, 2001] Bass, B. (2001). *RNA Editing. Frontiers in Molecular Biology Series*. Oxford University Press.

[Benne, 1993] Benne, R. (1993). *RNA Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

[Huang et al., 2007] Huang, C., Kaur, J., Maguitman, A., and Rocha, L. (2007). Agent-based model of genotype editing. *Evolutionary Computation*, 15(3):In Press–.

[Mitchell et al., 1992] Mitchell, M., Forrest, S., and Holland, J. (1992). In *ECAL 1992*.

[Rocha et al., 2006] Rocha, L. M., Maguitman, A., Huang, C.-F., Kaur, J., and Narayanan, S. (2006). An evolutionary model of genotype editing. In *Artificial Life X*, pages 105–111. MIT Press.