Agent-Based Model of Genotype Editing

Chien-feng Huang
Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Jasleen Kaur
Indiana University, Bloomington, IN 47406, USA

Ana Maguitman
Universidad Nacional del Sur, Bahía Blanca, Argentina

Luis M. Rocha*
Indiana University, Bloomington, IN 47406, USA
*To whom correspondence should be addressed: rocha@indiana.edu

Abstract

Evolutionary algorithms rarely deal with ontogenetic, non-inherited alteration of genetic information because they are based on a direct genotype-phenotype mapping. In contrast, in Nature several processes have been discovered which alter genetic information encoded in DNA before it is translated into amino-acid chains. Ontogenetically altered genetic information is not inherited but extensively used in regulation and development of phenotypes, giving organisms the ability to, in a sense, re-program their genotypes according to environmental cues. An example of post-transcriptional alteration of gene-encoding sequences is the process of RNA Editing. Here we introduce a novel Agent-based model of genotype editing and a computational study of its evolutionary performance in static and dynamic environments. This model builds on our previous Genetic Algorithm with Editing, but presents a fundamentally novel architecture in which coding and non-coding genetic components are allowed to co-evolve. Our goals are: (1) to study the role of RNA Editing regulation in the evolutionary process, (2) to understand how genotype editing leads to a different, and novel evolutionary search algorithm, and (3) the conditions under which genotype editing improves the optimization performance of traditional evolutionary algorithms. We show that genotype editing allows evolving agents to perform better in several classes of fitness functions, both in static and dynamic environments. We also present evidence that the indirect genotype/phenotype mapping resulting from genotype editing leads to a better exploration/exploitation compromise of the search process. Therefore, we show that our biologically-inspired model of genotype editing can be used to both facilitate understanding of the evolutionary role of RNA regulation based on genotype editing in biology, and advance the current state of research in Evolutionary Computation.

Keywords

RNA Editing, Genotype Editing, Genetic Algorithms, agent-based modeling, coevolution, indirect genotype/phenotype mapping, dynamic environments, biologically-inspired computing.

1 Introduction

Although RNA editing (Benne, 1993; Bass, 2001) seems to play an essential role in regulation and development of biological organisms, not much has been advanced to understand the potential evolutionary advantages, if any, that RNA editing processes may

have provided. Here we continue our study of the evolutionary advantages of genotype editing by testing an *Agent-Based Model of Genotype Editing* (ABMGE) against a sizable set of static and dynamic fitness functions. The model here presented marks a substantial departure from our previous Genetic Algorithm with Editing (GAE) (Huang and Rocha, 2003; Rocha and Huang, 2004; Huang and Rocha, 2004). While the GAE allowed a population of genotypes to evolve under fixed editing constraints, the more realistic ABMGE here presented allows some editing parameters to co-evolve with the phenotype solutions encoded in the genotypes of agents in the population.

In previous work we had already shown that the GAE, with appropriate editing parameters, can outperform the traditional Genetic Algorithm (GA) (Holland, 1975) in static (Huang and Rocha, 2003, 2004) and dynamic environments (Rocha and Huang, 2004). But for the GAE to outperform the GA, we needed to define good editing parameters by hand. Thus, while we showed that genotype editing can *in principle* be beneficial in an evolutionary process, we did not show how such a process could on its own discover the benefits of genotype editing. With the ABMGE presented in this paper, we show, empirically, that most beneficial editing parameters can be co-evolved to provide evolutionary advantages in both static and dynamic environments—without much user intervention other than the usual length and variation parameters of GA genotypes as well as length and variation parameters for editors. In (Huang and Rocha, 2005) we presented a preliminary study of the ability of the ABMGE to track changing extrema in dynamic environments using a simple mutation on editor parameters. In (Rocha et al., 2006) we explored a new form of editor variation similar to genetic crossover. Here we present a much more extensive study of the ABMGE using several additional static and dynamic fitness functions, as well as larger-scale simulations with longer runs for these environments.

Due to the larger scale of the study here reported, we present novel insights as to how genotype editing allows agents to search fitness spaces differently from agents with non-edited genotypes. We show that genotype editing is particularly useful in dynamic environments when the fitness function changes drastically. We also present new evidence to support the claim that genotype editing allows a wider exploration of the fitness space without sacrificing exploitation of good areas of the genotype space. The overall goal of our research is to gain a deeper understanding of the evolutionary nature of genotype editing as well as to exploit its insights to improve evolutionary algorithms and their applications to complex problems.

2 RNA Editing

Evidence for the important role of non-protein coding RNA (ncRNA) in complex organisms (higher eukaryotes) has accumulated in recent years. "ncRNA dominates the genomic output of the higher organisms and has been shown to control chromosome architecture, mRNA turnover and the developmental timing of protein expression, and may also regulate transcription and alternative splicing" (Mattick, 2003).

RNA Editing (Benne, 1993; Bass, 2001), is a process of post-transcriptional alteration of genetic information. Perhaps the most famous form of RNA editing operates by deletion and/or insertion of bases in the messenger RNA (mRNA) molecules of organisms such as African Trypanosomes (Benne et al., 1986; Benne, 1993; Stuart, 1993). Collectively these insertion and deletion process are called *indels* (Furey et al., 2004). In this case, insertion/deletion editing is performed by small *guide RNA's* (gRNA's) encoded mostly by what was previously thought of as non-functional or non-coding

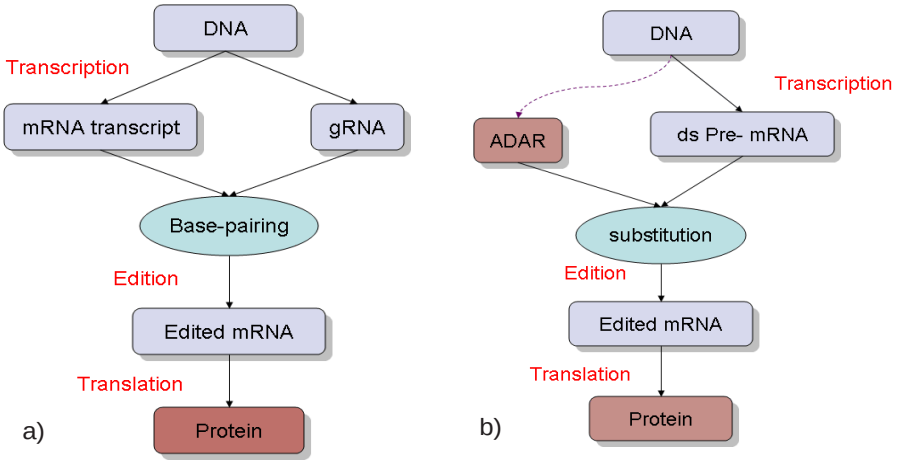

Figure 1: Schematics of insertion/deletion (a) and A-to-I substitution (b) RNA editing.

genetic material (Sturn and Simpson, 1990). [1] Guide RNA's (reviewed in (Simpson, 1999) and (Stuart et al., 1997)) are usually small sequences (compared to pre-edited mRNA's) that are complementary to the region around the site to be edited. gRNA molecules base-pair with mRNA regions to be edited and then insert, and sometimes delete, uridines into the mRNA (for examples see (Bass, 2001)). This editing alters the aminoacid chain encoded in the edited mRNA, sometimes extensively. Figure 1.a shows an abstract description of this biological process.

To appreciate the effect of this process let us consider Figure 2. The first example ((Benne, 1993), P. 14) shows a massive uridine insertion (lowercase u's); the amino acid sequence that would be obtained prior to any editing is shown on top of the base sequence, and the amino acid sequence obtained after editing is shown in the gray box. The second example shows how, potentially, the insertion of a single uridine can change dramatically the amino acid sequence obtained; in this case, a termination codon is introduced to create an open reading frame.

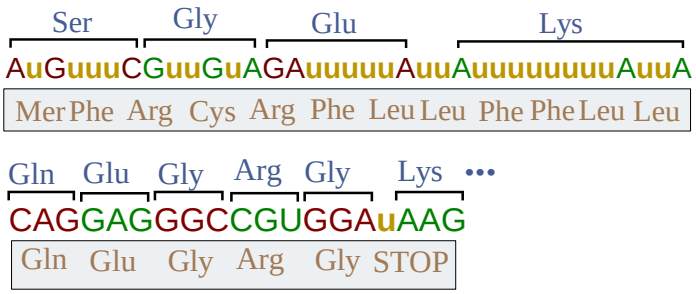

Figure 2: U-insertion in the RNA of Trypanosomes.

Other types of RNA Editing are tRNA and Viral G-addition editing, and small nucleolar RNA (snoRNA)-mediated nucleotide modification. But another common and important type of RNA editing happens via base substitution. This type of RNA editing exists in mitochondrial and chloropast RNA of many higher plants (with mostly C-to-U

[1] By non-coding genetic material we mean DNA not used to encode proteins (no open reading frame).

substitution) and, more importantly, in the genomes of higher eukaryotes such as mammals (with A-to-I substitution)—for reviews see (Blanc and Davidson, 2003) and (Maas et al., 2003). In this case, adenosine (A) is substituted for inosine (I) in double-stranded pre-mRNA, via the enzyme *adenosine deaminase acting on RNA* (ADAR), sometimes also known as *RNA Editase* (Chilibeck et al., 2006). In turn, inosine (I) is read as guanosine (G) in translation. A scheme of RNA editing via A-to-I substitution is depicted in Figure 1.b. This type of RNA Editing is known to be very important in the development of more complex organisms. For instance, the development of rats without the ADAR1 gene terminates midterm (Wang et al., 2000). This showed that A-to-I RNA Editing is more prevalent and important than previously thought. This type of RNA editing has also been identified in mammalian brains (Simpson and Emerson, 1996), including human brains (Mittaz et al., 1997). Lomeli et al. (1994) discovered that the extent of RNA editing affecting a type of receptor channels responsible for the mediation of excitatory postsynaptic currents in the central nervous system, increases in rat brain development. As a consequence, the kinetic aspects of these channels differ according to the time of their creation in the brains developmental process. More recently, Hoopengardner et al. (2003) found that RNA editing plays a central role in nervous system function. Indeed, many edited sites alter conserved and functionally important amino acids in protein sequences, some of which may play a role in nervous system disorders such as epilepsy and Parkinson Disease. Furey et al. (2004) have also found statistical evidence that a significant amount of nucleotide discrepancies in the human genome are due to RNA Editing (both A to I/G substitution and the lesser known T to C substitution).

The importance of RNA Editing is thus unquestionable, since it has the power to dramatically alter and regulate gene expression: "cells with different mixes of (editing mechanisms) may edit a transcript from the same gene differently, thereby making different proteins from the same opened gene." ((Pollack, 1994), P. 78). Let us now highlight some of the most salient features of RNA Editing, which are important for our model:

- A mRNA molecule may be more or less edited according to the concentrations of the editing operators it encounters. Thus, several different proteins encoded by the same gene may coexist in an organism or even a cell, if all (or some) of the mRNAs obtained from the same gene, but edited differently, are meaningful to the translation mechanism. This way, genotype editing can be successfully used for gene expression regulation from external or developmental cues (Rocha, 1995; Mattick, 2003).

- Genotype editing is not equivalent to mutation. In all cells, prokaryotic and eukaryotic, RNA is derived from DNA. Changes in genetic information can occur in a number of ways. For instance, when the DNA polymerase makes mistakes during DNA replication or when the RNA polymerase makes mistakes during RNA transcription. However, only changes that occur during DNA replication can become permanent and inheritable mutations. If changes occur during transcription, they get incorporated into that single transcript but not into other ones. Likewise, edited mRNA transcripts are not allowed inheritable variation: **what is inheritable, and subjected to variation, is the original non-edited gene** (Rocha, 1995; Rocha and Huang, 2004).

3 Modeling Genotype Editing

Genetic Algorithms (GA) (Holland, 1975) are based on an idealized model of Natural Selection. GA operate on an evolving population of artificial organisms, or agents. Each agent is comprised of a *genotype* (encoding a solution to some problem, typically in binary symbol strings) and a *phenotype* (the solution itself). Evolution occurs by iterated stochastic variation of genotypes [2] and selection of the best phenotypes in an environment according to how well the respective solution solves a problem (or fitness function). While idealized, GA capture the process of adaptation to an environment of a population of agents under genetic variation and phenotypic selection. Table 1 lists the steps of a simple genetic algorithm.

Table 1: GA Algorithm

1. Randomly generate an initial population of l agents, each consisting of a genotype (a n-bit string).
2. Evaluate the fitness of each agent.
3. Repeat until l offspring have been created.
a. select a pair of parent agents for mating;
b. apply genotype variation operators (mutation and crossover);
4. Replace the current population with the new population.
5. Go to Step 2 until terminating condition.

In a traditional GA, the code between genotype and phenotype is a direct and unique mapping[3]. In biological genetic systems, however, before a gene is translated into the space of proteins it may be altered through interactions with other types of molecules, namely RNA editors such as gRNA's or the enzyme ADAR. Based upon this analogy, Rocha (1995) proposed the expansion of the traditional GA with a process of stochastic editing of genotypes, prior to them being translated into solutions. In this model, genotype editing is performed by a small set of smaller genetic strings, the *editors*, which stochastically base-pair with genotypes. Each editor is associated with a different editing function, such as insertion, deletion or substitution of symbols into the original genotypes. In each generation, before translation into the space of solutions, each genotype has a certain probability (defined by the editor's "concentration") of encountering an editor. When there is an encounter, if the editor matches some subsequence of the genotype, the editor's function is applied and the genotype is altered.

We have previously tested this artificial genotype editing system with a *Genetic Algorithm with Editing* (GAE). In that model, the set of editors does not evolve and is the same for every agent in the population (Huang and Rocha, 2003; Rocha and Huang, 2004; Huang and Rocha, 2004). Using various static fitness functions (Royal Road, Optimal Control, and Michaelwicz) we first explored the effects of several parameters of the GAE model, such as the length of editor strings, their concentrations, editing functions and the like (Huang and Rocha, 2003). We later studied these parameters more extensively with many randomly generated experiments with additional static fitness functions (e.g. the Shaffer F_7 function) (Huang and Rocha, 2004). Finally, we tested

[2] Often known as chromosomes in evolutionary computation.

[3] Indirect mappings between genotype and phenotype have been modeled both in Evolutionary Computation (e.g. Kargupta (1996)) and Artificial Life (e.g. Dellaert and Beer (1994) and Rocha (2001)), but none of these models explores the mechanism of genotype editing.

the advantages of genotype editing in dynamic environments, that is, when the fitness function varies in time (Rocha and Huang, 2004).

With the GAE we have demonstrated that, with specific combinations of editor parameters, genotype editing can improve on the simple genetic algorithm's (GA) search performance on the various fitness functions tested. In particular, for the functions we tested, we were always able to find editor strings which lead the GAE to outperform the GA. The variety of functions tested allow us to establish that genotype editing, with specific parameters, can improve the search performance on a series of evolutionary scenarios: fitness functions that are amenable to search, epistatic, multi-modal, and dynamic. We have also showed that to be advantageous, the editing parameters must lead to moderate editing frequency (the total number of genotype edits in a generation). Too much editing is deleterious, and too little does not improve upon the regular GA.

However, to effectively study the evolutionary advantages of genotype editing, we need a model that allows good editing parameters to evolve automatically with the evolutionary search process. In summary, while with the GAE we showed that genotype editing can *in principle* be beneficial in an evolutionary process, we did not show how such a process could on its own discover the benefits of genotype editing. In the next section we present a model of genotype editing where beneficial editors can be co-evolved to provide evolutionary advantages in both static and dynamic environments—without much user intervention other than the usual length and variation parameters of GA genotypes as well as length and variation parameters for editors.

4 Agent-based Model of Genotype Editing

In our previous models of genotype editing (the GAE model), the set of editors did not evolve and was the same for every agent in the population (Huang and Rocha, 2003; Rocha and Huang, 2004; Huang and Rocha, 2004). In our novel *Agent-based Model of Genotype Editing* (ABMGE), the agents in the population are defined by an artificial genome that contains both *coding* and *non-coding* components. The coding component encodes solutions to a particular fitness function or environment, while the non-coding component defines a set of editors which act on the coding component. Let us refer to the coding portion of the artificial genome as the *codome*, and to the non-coding portion as the *editome*. In each generation, the coding component of an agent's genotype, the *codotype* may be stochastically edited by the agent's non-coding editors, its *editype*, and produce a solution/phenotype different from what is encoded. This way, the ABMGE defines a different set of editors for each agent, and furthermore allows them to co-evolve with solutions encoded in the genotypes of their respective agents. Figure 3 depicts an agent in the ABMGE; table 2 depicts its algorithm.

Notice that our agents possess functionally and operationally distinct codomes and editomes—for instance with separate variation operations (more details below). It would probably be more biologically realistic to extract both functions from a common artificial genome (e.g. as defined by (Reil, 1999)). However, here we want precisely to explore the influence of an editome in the evolutionary process, therefore we functionally separate it from the codome to better test its relative importance.

4.1 Co-evolving Codotype and Editype

The genome of our agents consists of a codome and an editome. The *codotype* of each agent is a specific n-bit string, whereas the *editype* consists of a specific family of r editors each defined by a m-bit string: $(E_1, E_2, \ldots, E_r)$. The length of the editor strings

Table 2: 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 E_j with probability v_j; If E_j matches S at some position, 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.
5. Go to Step 2 until terminating condition.

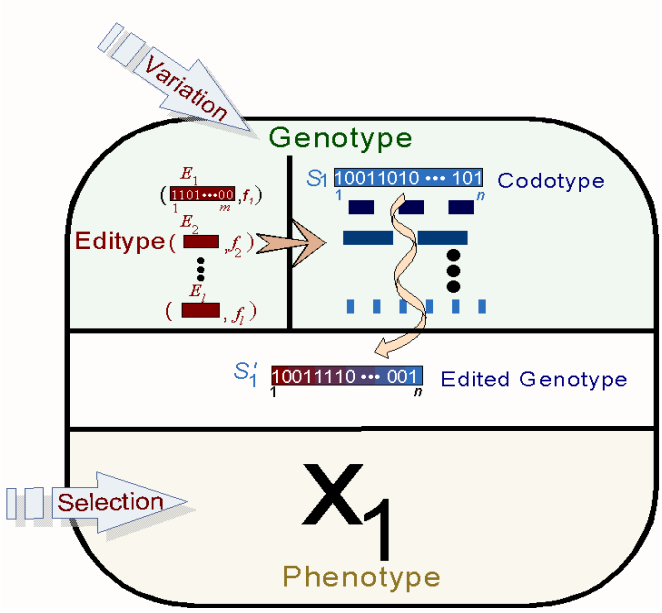

Figure 3: Individual Agent in the ABMGE.

is defined much smaller than that of the codotype strings: $m << n$, usually an order of magnitude. An editor E_j is said to match a substring, of size m, of a codotype string S at position k if $e_i = s_{k+i}, i = 1, 2, \ldots, m$, $1 \leq k \leq n - m$, where e_i and s_i denote the i^{th} bit value of E_j and S, respectively. For each editor E_j there exists an associated editing function F_j, that specifies how a particular editor edits the codotypes it matches. For instance, when the editor matches a portion of a codotype string, a number of bits may be inserted into or deleted from it.

If the editing function of editor E_j is to add one specific allele at s_{k+m+1} when E_j matches S at position k, then all alleles of S from position $k+m+1$ to $n-1$ are shifted one position to the right (the allele at position n is removed). Analogously, if the editing function of editor E_j is to delete an allele, an allele at s_{k+m+1} is deleted when E_j matches S at position k. All the alleles after position $k+m+1$ are shifted in the inverse

direction (one randomly generated allele is assigned at position n). Finally, let the concentration of the editor family be defined by $(v_1, v_2, \ldots, v_r)$; i.e., the concentration of editor E_j is denoted as v_j. Then the probability that S encounters E_j is given by v_j. In the remaining of this article, when we refer to an editor we mean a 3-tuple (E_j, F_j, v_j).

In the previous GAE model, an agent's genotype consisted entirely of a codotype—every bit of the genotype was "functional" in that it was used to encode a phenotype. Moreover, every agent in the population faced the same "post-transcriptional" editors which were fixed at the start of each run. In other words, every agent possessed the same fixed editype. In contrast, in the ABMGE, agents face individualized, heterogenous post-transcriptional editors. Notice that while the editome is an additional portion of the genome, it cannot be seen simply as another chromosome (in the sense used in evolutionary computation), because it is not encoding a (phenotype) solution or a part of a solution. 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. In our model, the editome is used not to encode phenotypic attributes (as a typical chromosome in evolutionary computation does), but rather to model the post-transcriptional, pre-translation process of genotype editing.

As fit agents are selected for reproduction, their editypes propagate to the next generation together with their codotypes. Thus, codotypes *co-evolve* with editypes to deal with a specific fitness function. Coevolutionary algorithms, methods by which multiple populations of agents (or species) are adapting to each other, have been seriously studied in the field of Evolutionary Computation (Ficici and Pollack, 2003; Panait et al., 2004). They are popular augmentations of traditional evolutionary algorithms. In nature, coevolution is the process of reciprocal genetic change of one species in response to another. The reciprocal change observed in coevolution can be considered either as a competitive arms race, or cooperative methodology where separate populations evolve components of the solution (Potter and De Jong, 2000).

The ABMGE relies on a kind of co-evolution of editypes with codotypes encoding solutions to an optimization problem. However, the algorithm does not rely on distinct populations of agents, as it is typically done in coevolutionary algorithms. Instead, there is a single population of agents which possess genotypes with two types of "genetic" information: solution-encoding codotypes, and editypes which perform non-inheritable (ontogenetic) stochastic changes to the codotypes. Because codotypes and editypes are tied together in an agent, they cannot evolve independently. In the sense that they evolve jointly and in mutual dependence, we can think of codotypes and editypes as *co*-evolving[4]. Indeed, the co-evolution happens because an agent will not be able to survive deleterious variation on either the codotype or the editypes. Changes either to the codotype or editype must produce advantageous or neutral phenotypic changes for an agent to survive.

In the progression of the ABMGE runs, as fitter agents are selected for reproduction, their editypes propagate to the next generation. Therefore, the ABMGE establishes an automatic process for generating good editors for a particular problem, together with good solutions for the problem. Instead of manually choosing specific editor parameters such as specific editor strings (as the GAE demands) one only needs to specify the range of the editor parameters in the very beginning of the experiments.

[4]The Oxford English Dictionary defines the prefix *co-* when applied to verbs as pertaining to "a joint subject: as, co-engage to engage along with others, co-sustain to sustain jointly" and to adjectives and adverbs as pertaining to mutual interaction as in "*co-embedded* embedded together, *co-harmonious* unitedly harmonious, *co-intersecting* intersecting mutually;".

A few additional observations are necessary to understand the ABMGE and the implications of a genotype with separate codotype and editype:

- It is important to note that **the "post-transcriptional" editing of codotypes is not akin to mutation, because edits are not inheritable**. Just like in biological organisms, in our model, it is the unedited genotype (codotype plus editype) that is reproduced, while agent fitness is calculated using the phenotype produced from the edited codotype. Therefore, the unedited and edited codotypes can be viewed as mimicking coding (functional) DNA and edited mRNA's in biological organisms, respectively—even though this model does not include a true DNA/RNA distinction.

- Just like a mRNA molecule may be edited differently according to specific editing molecules and their concentrations it encounters, in our model **the same codotype may be edited differently because editor concentration is a stochastic parameter** that specifies the probability of a given editor encountering the codotype before translation. This means that the same agent, in distinct generations, may produce different phenotypes. It also means that if the same codotype and even the same genotype (codotype plus editype) is repeated in the agent population, the phenotypes of the respective agents may be different.

- In Rocha's formulation (Rocha, 1995), any bit-string editor function is possible, including substitution. In the present work we use only insertion and deletion functions. However, given that in our model there is no equivalent of the transcription of RNA from DNA and we use a two-symbol encoding, what we are modeling is a **generic process of non-inheritable alteration of an agent's genetic information** (codotype) via editing, before it is translated into a solution (phenotype)—not a specific type of RNA Editing.

We should also notice that genotype editing as we study it here, is not the same as the Baldwin effect as studied by Hinton and Nowlan (1987) and subsequent developments of their model. The phenotypes of our agents with genotype editing, do not change (or learn) ontogenetically. In Hinton and Nowlan's experiments, the environment is defined by a very difficult ("needle in a haystack") fitness function, which can be made more amenable to evolutionary search by endowing the phenotypes with time to "learn" ontogenetically. Eventually, they observed, this learning allows genetic variation to discover, and genetically encode fit individuals. In contrast, genotype editing does not grant agents more "ontogenetic learning time"; it simply changes inherited genetic information before translation but the phenotype, once produced, is fixed. Also, as we show below, it is advantageous in environments very amenable to evolution, such as the Royal Road (sections 5.1.1 and 5.2.1) and De Jong functions (section 5.1.2), which are the opposite of "needle in a haystack'.

4.2 Editype Variation

The variation operations of codotypes (mutation and crossover) operate just like in a regular GA. Therefore, here we reserve detailed explanations only for the variation operations of the editype. When two parents are selected for reproduction in our algorithm (step 4 in table 2), in addition to variation of codotypes as it is commonly done in GA, the editype is also subjected to variation. In the current implementation, we only apply variation to the set of editor strings E_j, while the associated functions F_j and

concentrations v_j remain unchanged; we will consider variation on these parameters in future work.

We implemented *editype mutation* on editor bit-strings exactly as it is typically done in codotype bit-strings: a bit-mutation probability, P_{EdMut}, defines the probability that each individual bit of an editor string is flipped in a given generation. In our model, the editype bit-mutation probability P_{EdMut} is independent from the codotype bit-mutation probability: P_{Mut}. In (Huang and Rocha, 2005) we presented a preliminary study of the ability of the ABMGE to track changing extrema in dynamic environments using editype mutation only. In (Rocha et al., 2006) we introduced a form of editor variation similar to genetic crossover which we explore in much more detail here. Indeed, in the present work we present a much more extensive study of the ABMGE with editype mutation and crossover than previously done. We present longer and larger-scale simulations with several additional static and dynamic fitness functions.

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 is a random integer (sampled from a uniform distribution) from the interval $[1, MIN(r_1, r_2)]$. Editype crossover occurs with a probability $P_{EdCross}$, which is independent from the typical codotype (one-point) crossover which occurs with a probability P_{Cross}. Therefore, when two parents are selected in step 4 of table 2, we may have that (1) no crossover of any kind occurs, in which case the parents are reproduced as they are; (2) codotype crossover occurs with no editype crossover; (3) editype crossover occurs with no codotype crossover; and (4) both types of crossover occur. In future work we will explore making both types of crossover dependent. See figure 4 for a depiction of agent variation mechanisms in the ABMGE.

5 Testing the ABMGE

In our previous work, we have shown that the GAE, with appropriate editors, can outperform the traditional GA in static and dynamic environments (Huang and Rocha, 2003; Rocha and Huang, 2004; Huang and Rocha, 2004). With the simulations we report here, we now show that the ABMGE is capable of co-evolving appropriate editor parameters to provide evolutionary advantages in both static and dynamic environments—without much user intervention. Thus, the ABMGE, unlike the GAE, does not require a user to specify editing parameters in order to significantly outperform the traditional genetic algorithm.

5.1 Static Environments

The performance of evolutionary algorithms is typically evaluated by monitoring improvement of the solutions discovered by the population of evolving agents in each generation. In many practical problems, a traditional performance measure is the "best-so-far" curve that plots the fitness of the best individual that has been seen thus far by generation n. We tested the behavior of the ABMGE with various fitness functions which we have previously used to test the simpler GAE (Huang and Rocha, 2003; Rocha and Huang, 2004; Huang and Rocha, 2004).

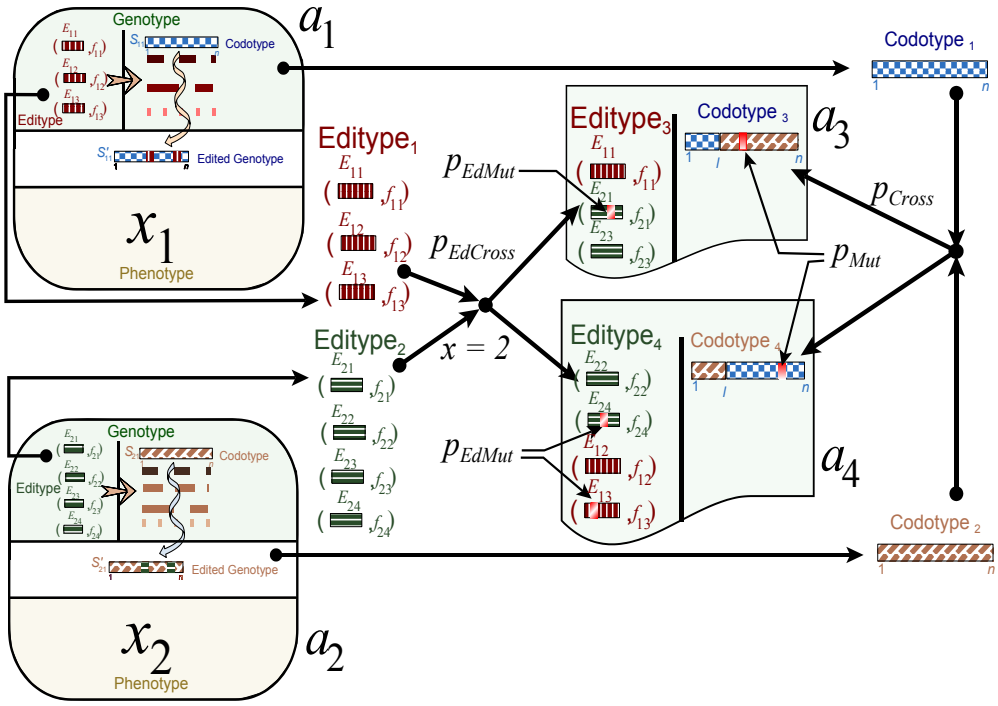

Figure 4: Agent variation in the ABMGE for two selected parents with $r_1 = 3$, $r_2 = 4$, and $x = 2$.

5.1.1 Small Royal Road

The first fitness function tested is a miniature of the class of the "Royal Road" functions (Forrest and Mitchell, 1993): the *small Royal Road* SRR_1, as depicted in Table 3. This function is 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 3; $\sigma_t(S) = 1$ if schema t exists in S and 0 otherwise. The optimum fitness for SRR_1 is ascribed to a single string with 40 1's, and its value is $10 \times 8 = 80$. This Royal Road function is selected as a testbed because it serves as an idealized fitness environment with a single optimum, particularly amenable to evolutionary search .

Table 3: 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

We contrasted the traditional GA with two versions of the ABMGE: with and with-

out editype crossover [5]. Our experiments with SRR_1 use binary tournament selection (Goldberg and Deb, 1991), a population of 40 agents over 200 generations for 50 runs—in every run, all parameters are randomly re-generated. Codotype variation (in both the ABMGE and the simple 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 previously found for the GA (see section 7). For the editome parameters (of the ABMGE), we employ the guidelines discovered in (Huang and Rocha, 2004) to randomly generate editor families. The size of the editor family r for each agent is randomly sampled from $\{1, \cdots, 5\}$; the editors are generated as randomized bit-strings S_j of size m also randomly sampled from $\{2, \cdots, 4\}$; the editor concentration is randomly generated from [0,1]; and the editor function inserts or deletes a number of bits randomly chosen from $\{1, \cdots, 3\}$ (which are fixed once the editor is generated).

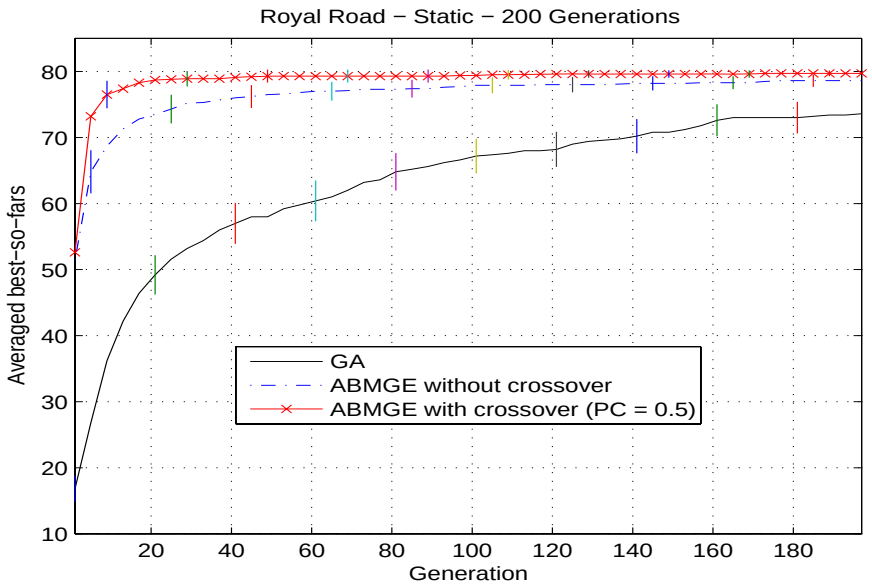

Figure 5: Performance of GA, ABMGE without editype crossover, and ABMGE with editype crossover ($P_{EdCross} = 0.5$) on SRR_1.

We conducted our experiments on SRR_1 for various values of editype variation: $P_{EdMut} \in \{0.01, 0.05\}$ and $P_{EdCross} \in \{0, 0.3, 0.5, 0.7, 0.9\}$. Figure 5 depicts the results for both $P_{EdMut} = 0.05$ and $P_{EdCross} = 0$, which we refer to as ABMGE without editype crossover, as well as $P_{EdCross} = 0.5$, which we refer to as ABMGE with editype crossover. One can see that the averaged best-so-far reached by both versions of the ABMGE is much closer to 80 (the maximum) than what the traditional GA obtains at the end of the experiments, and significantly better for the ABMGE with both editype mutation and crossover. This was observed for all values of $P_{EdCross}$ tested.[6] The ABMGE, with randomly generated editypes clearly outperforms the GA. This means that

[5] The parameter settings used in this subsection are applied to all the other functions tested in this article, unless otherwise specified.

[6] The value of the averaged best-so-far performance metric is calculated by averaging the best-so-fars (the fitness of the best individual that has been seen thus far by generation n) obtained at each generation for all runs, where the vertical bars overlaying the metric curves represent the 95-percent confidence intervals. This applies to all the experimental results obtained in this paper.

the ABMGE is capable of automatically discovering advantageous editors from randomly generated ones. Table 4 shows the editype of an agent at the end of a ABMGE run for SRR_1—this editype is very prevalent in the population at the end of that run.

Table 4: Parameters of a 3-editor editype of an agent at generation 200.

	editor 1	editor 2	editor 3
length	5	3	5
alleles	{1, 1, 0, 1, 0}	{0, 1, 0}	{1, 1, 1, 1, 1}
concentration	0.4151	0.7103	0.5090
function	delete 2 bits	delete 3 bits	add 3 bits

5.1.2 De Jong Function F_3

De Jong's function F_3 (De Jong, 1975) is defined as:

$$f(\overline{x}) = \sum_{i=1}^{N} \text{integer}(x_i), \tag{1}$$

where $\overline{x} = [x_1, x_2, \ldots, x_N]^T$, $-5.12 \leq x_i \leq 5.12$ for $1 \leq i \leq N$.[7] A two-dimensional sketch of this function is presented in Figure 6. As can be seen, this function is a simple, unimodal step function where we can expect the global optimum to be easily located. Thus to contrast the performance of the GA with the ABMGE, we need to compare the mean function evaluations required for locating the optimum, rather than the actual maximum value attained.

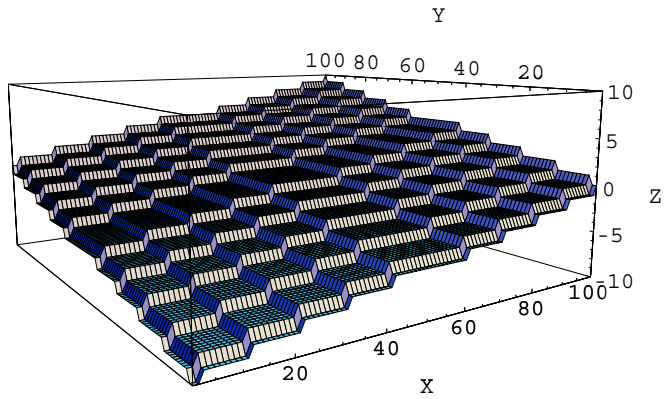

Figure 6: Two-dimensional De Jong function F_3. The X and Y-axis represent the parameters x_1 and x_2 used to compute $f(\overline{x})$, which is represented on the Z-axis

In our tests for this function, five variables (N=5) are used. Each variable is encoded using 10 bits. The 5 blocks of 10 bits each are then concatenated to form a string of length 50. For both the GA and ABMGE we use a population of 50 agents over 200 generations for 100 runs. For the editypes of the agents in the ABMGE, we tested

[7] The original version of De Jong function F_3 is $\sum_{i=1}^{5} \text{integer}(x_i)$, and the goal was to minimize $f(\overline{x})$. This function was modified here to be consisting with the maximization goal of all other functions used

three different range scenarios for the size of the family of the editors, r: (i) $\{1, 2\}$, (ii) $\{1, \cdots, 5\}$, and (iii) $\{1, \cdots, 8\}$. Again, we tested the ABMGE with ($P_{EdCross} = 0.5$) and without ($P_{EdCross} = 0$) editype crossover. In either case, we used ($P_{EdMut} = 0.05$). The values of the other parameters remain the same as those used for the Royal Road testbed SRR_1 in subsection 5.1.1.

Table 5: Mean and standard error function evaluations to the optimum with De Jong's function F_3.

Algorithm	Editype Parameters	Mean (Std. Err.) eval. to optimum
GA	N/A	3237.25 (194.15)
ABMGE	No Crossover and $r \in \{1, 2\}$	1422.96 (178.37)
ABMGE	No Crossover and $r \in \{1, \cdots, 5\}$	898.74 (172.19)
ABMGE	No Crossover and $r \in \{1, \cdots, 8\}$	586.62 (72.82)
ABMGE	Crossover and $r \in \{1, 2\}$	770.09 (122.77)
ABMGE	Crossover and $r \in \{1, \cdots, 5\}$	329.15 (50.75)
ABMGE	Crossover and $r \in \{1, \cdots, 8\}$	252.68 (46.98)

Table 5, shows the mean function evaluations required for locating the optimum for the GA and the ABMGE for the three editype family size ranges tested (the standard errors are shown in the parentheses). By contrasting these cases, it is obvious that the ABMGE significantly outperforms the GA in amenable search spaces such as the one defined by De Jong's function F_3 and the Royal Road of section 5.1.1. Furthermore, the ABMGE with both editype mutation and crossover significantly outperforms the ABMGE without editype crossover in this type of function, for the same editype parameters. Figure 7 depicts the GA against the ABMGE with and without crossover, for the same size of editor families $r \in \{1, \cdots, 8\}$.

5.1.3 The Optimal Control Test Problem

Optimal Control problems often arise in many different fields of engineering and science. This class of problems has been well studied from both theoretical and computational perspectives. The models used to describe optimal control problems almost always involve nonlinearity in nature. This often results in the existence of multiple local optima in the area of interest. (See (Hager and Pardalos, 1998) for a sample of the available material and applications.) In this subsection we employ an artificial optimal control problem designed in (Huang, 2002). The constraints of the artificial optimal control problem are:

$$\frac{d^2z(t)}{dt^2} + sin(z(t))\frac{dz(t)}{dt} + sin(t)cos(z(t))z(t)^3 = \quad sin(t)u_1^2 + cos(t)u_2^2 + sin(t)u_1u_2,$$
$$z(t_0) = 2, \dot{z}(t_0) = 2, t \in [0, 1]. \tag{2}$$

The goal is to maximize $z(t_f)^2$ by searching for two constant control variables, u_1 and u_2 ($-5 \leq u1, u2 \leq 5$). A sketch of this function is displayed on the left side of Figure 8. There are clusters of spikes at two corners of the search space, and a hill that occupies most of the space. The magnified view on the right side of Figure 8 shows a clearer view of the height and area of the hill. As can be seen, the height of the hill is much lower than that of the spikes. Nonetheless, since it occupies most of the

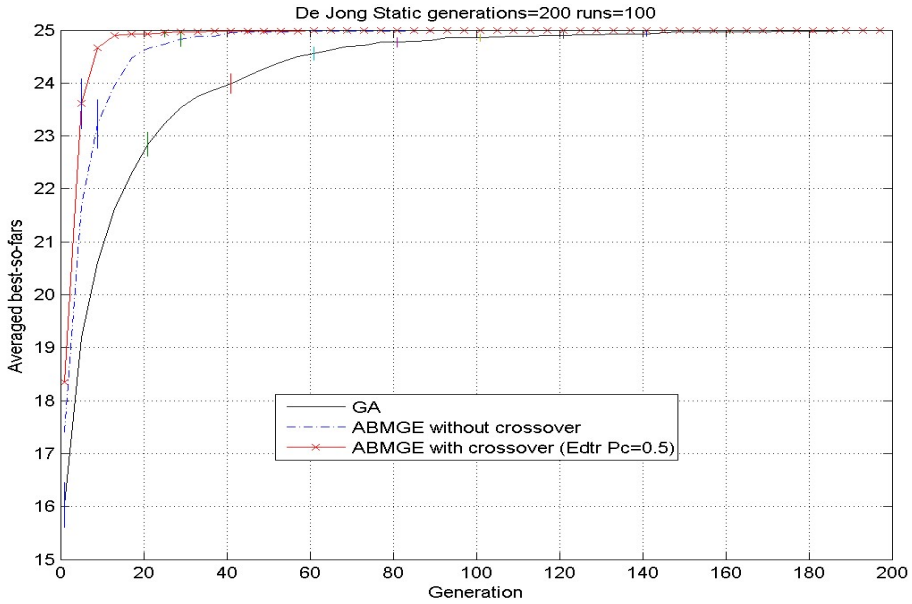

Figure 7: Averaged best-so-far performance on De Jong function F_3 for $r \in \{1, \cdots, 8\}$.

search space, one can expect that most of the population individuals in an evolutionary algorithm would be attracted to the hilltop, which would thus miss the higher fitness peaks. Compared to the functions used in sections 5.1.1 and 5.1.2, the search space of this fitness function is characterized by hard-to-find optima.

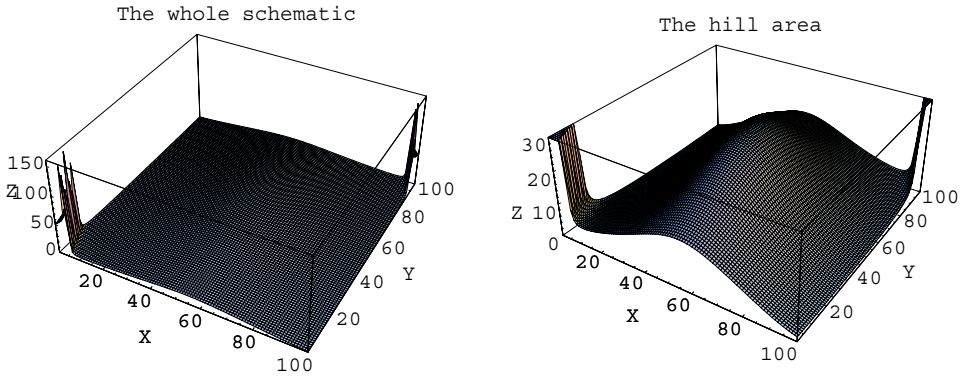

Figure 8: The optimal control test function. The X and Y axes represent u_1 and u_2 which are used to compute $z(t_f)^2$, which is plotted on Z-axis.

In our tests, each of the two variables is encoded by 50 bits, and thus each individual is a binary string of length 100. For both the GA and ABMGE we use a population of 50 agents over 200 generations for 100 runs. We again contrast the GA with the ABMGE, with and without editype crossover, using the same parameters for the codotypes as the GA. The editype parameters are the same employed in the previous subsections, except that three different sizes of the family of the editors for each agent

were examined: $r \in \{1,2\}, \{1,\cdots,5\}$, and $\{1,\cdots,10\}$. In all three cases, both versions of the ABMGE significantly outperformed the GA. Figure 9 depicts the averaged best-so-far performance for the GA against the most favorable ABMGE range scenario for size of editor families $r \in \{1,\cdots,10\}$.

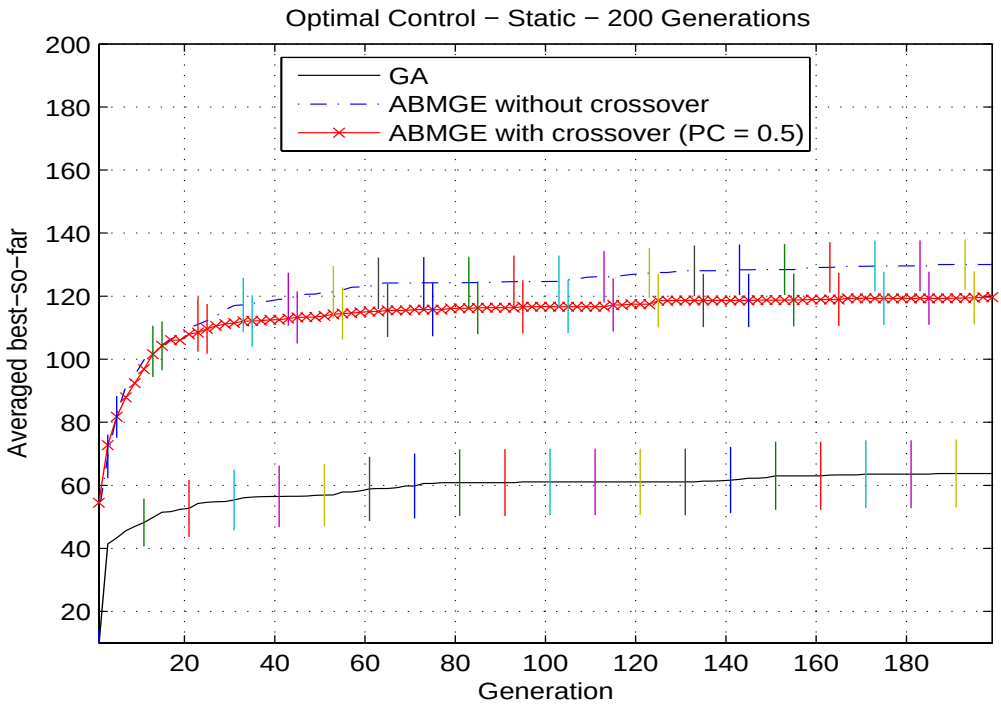

Figure 9: Best-so-far performance on the optimal control problem for $r \in \{1,\cdots,10\}$

While there is no significant difference between the two versions of the ABMGE (with and without editype crossover), both quite significantly outperform the GA. We also examined how often a given algorithm reached the harder to find peaks in the search space. The best-so-far value reached by the GA at the end of 200 generations is only 27.01 (the fitness at the center hilltop) for nearly 60 out of 100 runs. In contrast, the ABMGE explores more of the search space and extends the best-so-fars to higher ranges, showing that it tended to find the higher peaks more often than the GA. Indeed, even in the least favorable editype size range scenario ($r \in \{1,2\}$), the ABMGE is only caught in the center hilltop in 20 out of 100 runs. This can be seen in the histograms of best-so-far reached at the end of 200 generations for the 100 runs, depicted in Figure 10.

5.1.4 Modified Schaffer's Function F_7

In addition to the more amenable Small Royal Road SRR_1 and De Jong F_3 fitness functions, as well as the relatively simple Optimal Control Test problem of the previous section, we also tested the much harder, multimodal modified *Schaffer's function* F_7 (Huang, 2002) :

$$f_7(\overline{x}) = 2.5 - (x_1^2 + x_2^2)^{0.25}[sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1] \quad (3)$$

where $-1 \leq x_i \leq 1$ for i = 1, 2. A sketch of this function is displayed in Figure 11. To attain the global optimum (2.5) at the center of the search space, the population has

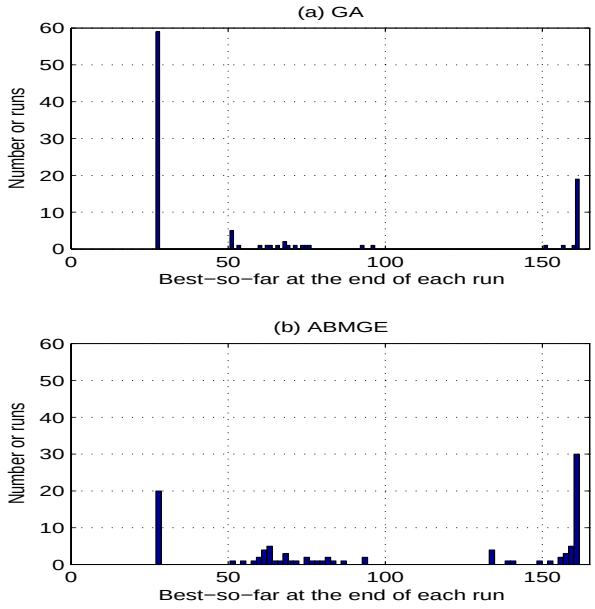

Figure 10: Distribution of best-so-fars for the GA and the ABMGE without editype crossover for the optimal control problem, for $r \in \{1, 2\}$

to move across many deep wells and high barriers. Since there are many local optima in the search space, the population of an evolutionary algorithm can easily converge to any of them. The multimodality of the problem is hence expected to present substantial difficulty to evolutionary search.

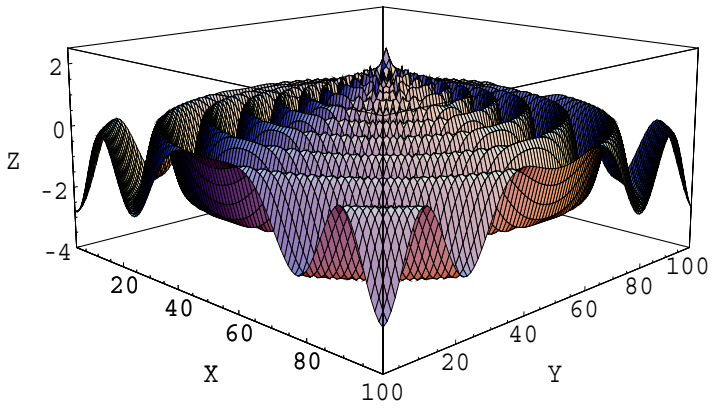

Figure 11: Modified Schaffer function F_7.

Each of the two variables of F_7 is encoded by 50 bits, and thus each individual is a binary string of length 100. We contrasted the traditional GA with two versions of the ABMGE: with and without editype crossover. We used the same parameters as before with an agent population size of 50 agents, and three different sizes of the family of the editors for each agent were examined: $r \in \{1, \cdots, 5\}, \{1, \cdots, 10\}$, and $\{1, \cdots, 20\}$, and we computed statistics for 100 runs. We conducted our experiments for various values of editype crossover probability $P_{EdCross} \in \{0, 0.3, 0.5, 0.7, 0.9\}$.

The ABMGE significantly outperformed the GA for all cases, except when $r \in \{1, \cdots, 5\}$, in which case the two were statistically indistinguishable. However, for larger editypes ($\{1, \cdots, 10\}$, and $\{1, \cdots, 20\}$), the ABMGE very significantly outperformed the GA. Figure 12 depicts the results for $r \in \{1, \cdots, 20\}$, $P_{EdMut} = 0.05$, and $P_{EdCross} = 0.5$. One can see that the averaged best-so-fars reached by both versions of the ABMGE clearly outperform the traditional GA, and the ABMGE with both editype mutation and crossover also significantly outperforms the ABMGE without editype crossover. This was observed for all values of $P_{EdCross}$ tested.

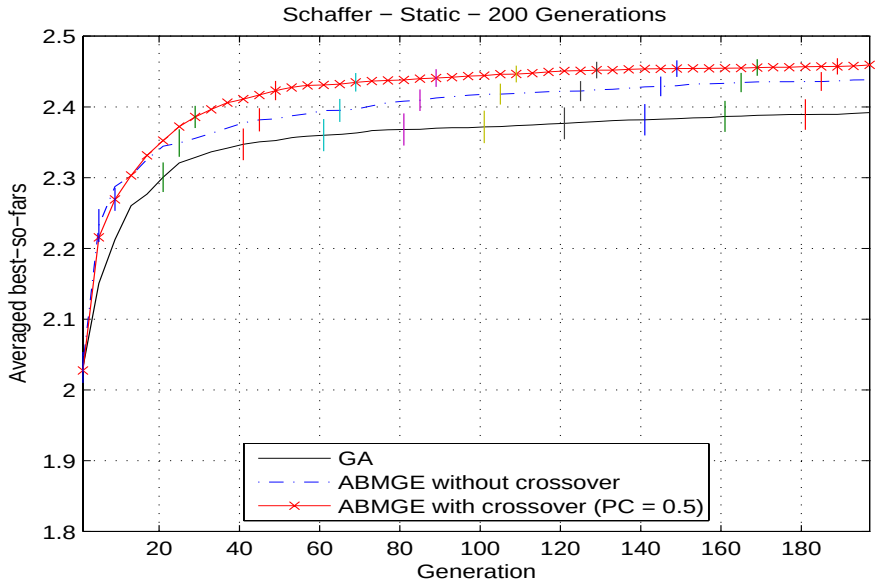

Figure 12: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on F_7.

5.1.5 Schwefel's Function

In this section we tested a multimodal deceptive function – *Schwefel's function* (Schwefel, 1981):

$$f(\overline{x}) = \sum_{i=1}^{N} x_i \sin(\sqrt{|x_i|}), \tag{4}$$

where $-500 \leq x_i \leq 500$ for i = 1,...,n. A plot of this function is displayed in Figure 13.

Schwefel's function is deceptive in the sense that the global maximum is geometrically distant, over the parameter space, from the next best local optima. Therefore, the agents in the population of an evolutionary algorithm are potentially prone to convergence in the wrong direction. The deceptiveness of the problem is thus expected to present substantial difficulty to evolutionary search.

We illustrate the search performance by using two variables for the Schwefel function. Each of the two variables is encoded by 50 bits; each individual is thus a binary string of length 100. We contrasted the traditional GA with two versions of the ABMGE: with and without editype crossover. We used the same parameters as before except the agent population size is 30 agents, and the size of the family of editors for each agent

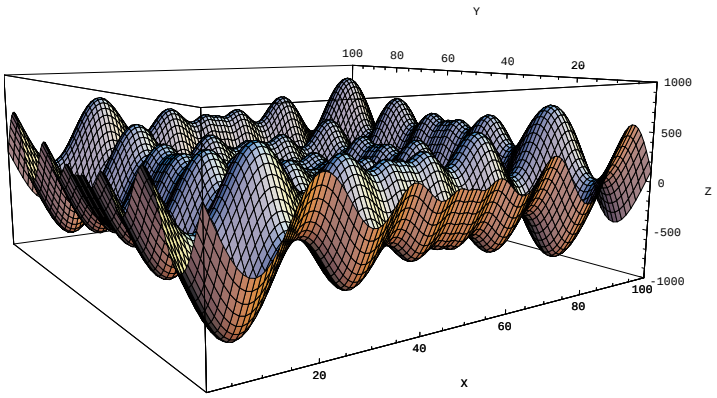

Figure 13: Modified Schwefel function.

used was: $r \in \{1, \cdots, 10\}$, and we computed statistics for 100 runs. We conducted our experiments for editype crossover probability $P_{EdCross} = 0.5$.

The results of these runs are depicted on figure 14, where one can see that the averaged best-so-far reached by both versions of the ABMGE outperforms the traditional GA, though the two versions of the ABMGE were statistically indistinguishable. This indicates that both versions of the ABMGE can achieve better search performance than the GA in the deceptive Schwefel function.

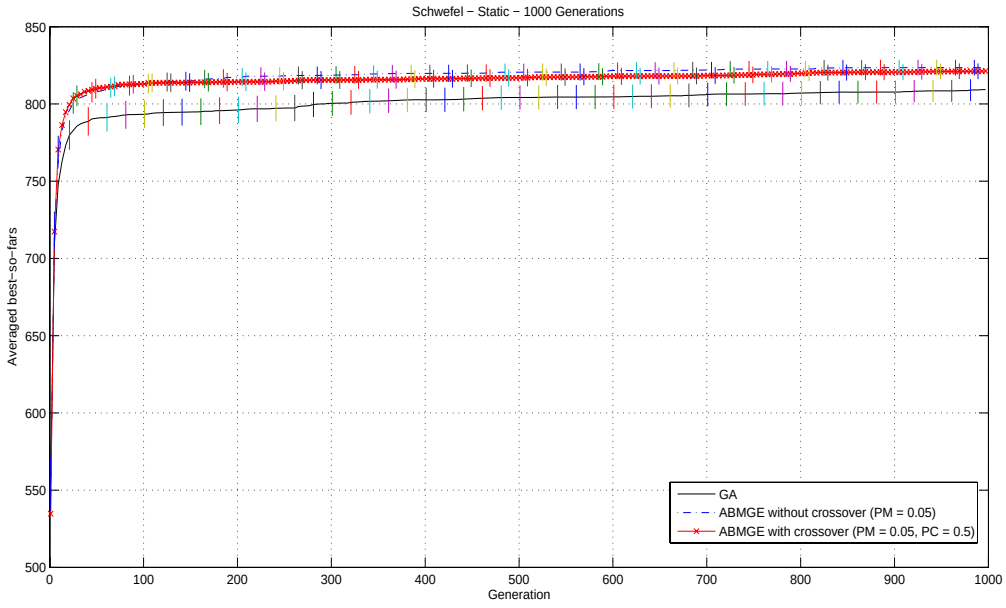

Figure 14: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on Schwefel's Function.

5.2 Dynamic Environments

We know that some biological organisms, namely parasites that go through dramatic environmental changes, use RNA editing to their advantage to regulate gene expres-

sion, thus achieving profound phenotypic plasticity to different environments (Rocha, 1995; Rocha and Huang, 2004). We have shown that our previous model of genotype editing, the GAE, can provide a means for artificial agents with genotypes to gain greater phenotypic plasticity in dynamic environments (Rocha and Huang, 2004). In those experiments, by linking changes in the environment with genotype editing parameters such as concentrations of editors, the artificial agents were shown to use genotype editing to their advantage, namely by allowing them to regulate their phenotypes contextually (Rocha, 1995).

While in (Rocha and Huang, 2004) we showed that genotype editing can be used to provide phenotypic plasticity in a few dynamic environments, the GAE does not provide an automatic mechanism to discover good editor families to enable such plasticity. In the experiments we describe below, instead of arbitrarily linking editor concentrations to changes in the environment, as we did with the GAE (Rocha and Huang, 2004), we simply let the co-evolution of editypes and codotypes of the ABMGE operate automatically in dynamic environments.

Evolutionary optimization in static environments, such as those of the previous subsection, involve the search of the extrema of functions. For dynamic environments, where the fitness function changes in time, the interest is not so much to locate the extrema but to follow it as closely as possible. This section compares the extrema-tracking performance of the traditional GA and our two versions of the ABMGE: with and without editype crossover. To perform this study we adapted the static fitness functions used above to a dynamic setting.[8]

5.2.1 Oscillatory Small Royal Road: drastic environmental changes

Consider another Small Royal Road function, SRR_0, in which each schema is comprised of all 0's rather than 1's as SRR_1 of section 5.1.1, but with all other parameters the same as SRR_1. Our first dynamic fitness function, the *oscillatory royal road* (ORR), oscillates between SRR_1 and SRR_0 at every p generations. Because SRR_0 and SRR_1 are maximally different, we are able to study the effects of drastic environmental changes when we oscillate between them: as the agent population adapts to one of the fitness landscapes, the environment changes drastically to the other one.

The parameters of the dynamic environment simulations using ORR are the same as those used for the static SRR_1 of section 5.1.1. In addition to various editype mutation and crossover probabilities, we tested different oscillation periods: $p \in \{50, 100, 200, 250\}$. Figure 15 depicts the results for $P_{EdMut} = 0.05$ and $P_{EdCross} = 0.5$ and period $p = 100$ for the first 1000 generations of a simulation of 50 runs for 4000 generations; Figure 16 depicts that last 1000 generations of this same simulation, and Figure 17 depicts the entire simulation.[9]

It is clear that it is difficult for a population to re-adapt to an entirely new environment, as the best-so-far solution significantly declines when the environment changes. The traditional GA, as time progresses degrades its performance even when the first fitness environment (SSR_1) returns. Indeed, at the end of 4000 generations for $p = \{50, 100, 200\}$ its performance on both environments eventually reaches the same level (close to 40 for $p = 50$ and $p = 100$, and close to 50 for $p = 200$). This means that

[8]Notice that the best-so-far measurement we used in stationary environments is problematic in dynamic environments since it has to be assured that the best solution found thus far is the best solution for the current environment. Therefore, the best-so-far solution we track is re-evaluated whenever a change in the environment occurs.

[9]Results for periods: $p = \{50, 200\}$, various details and separate plots for each algorithm are available as supplemental materials on line at http://informatics.indiana.edu/rocha/editing.

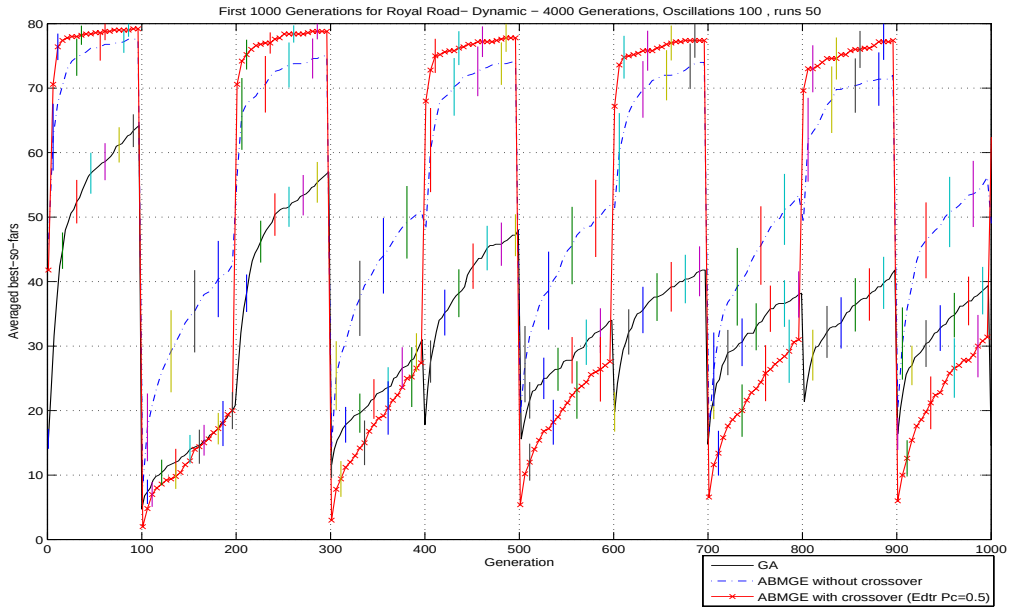

Figure 15: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on ORR, $p = 100$, for the first 1000 generations of a simulation of 50 runs and 4000 generations.

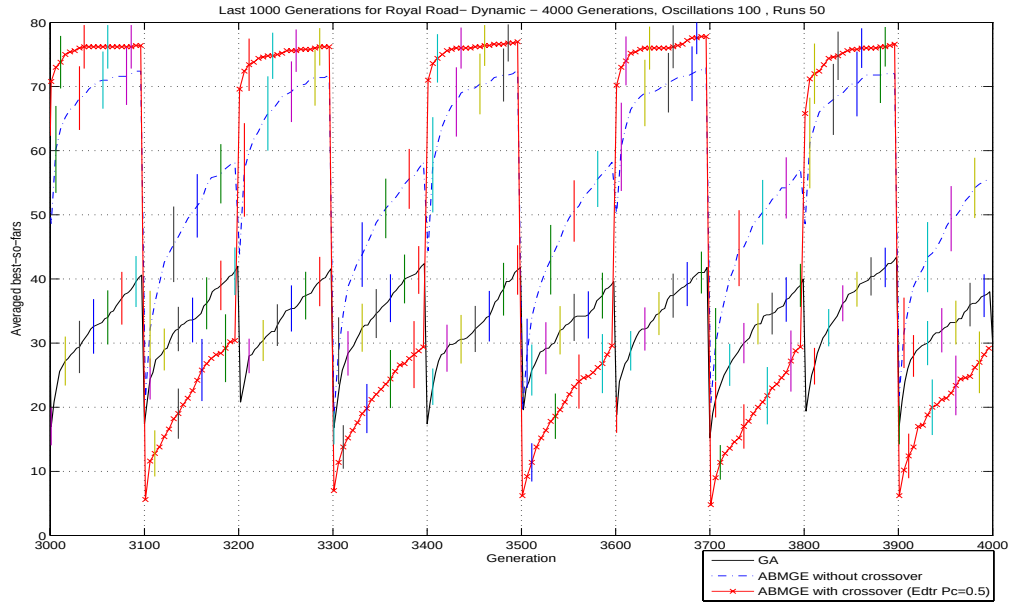

Figure 16: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on ORR, $p = 100$, for the last 1000 generations of a simulation of 50 runs and 4000 generations.

the GA ultimately converges to a population with a balanced combination of schemata of all 1's and schemata of all 0's.

As for the ABMGE, interestingly, the version with both editype crossover and mu-

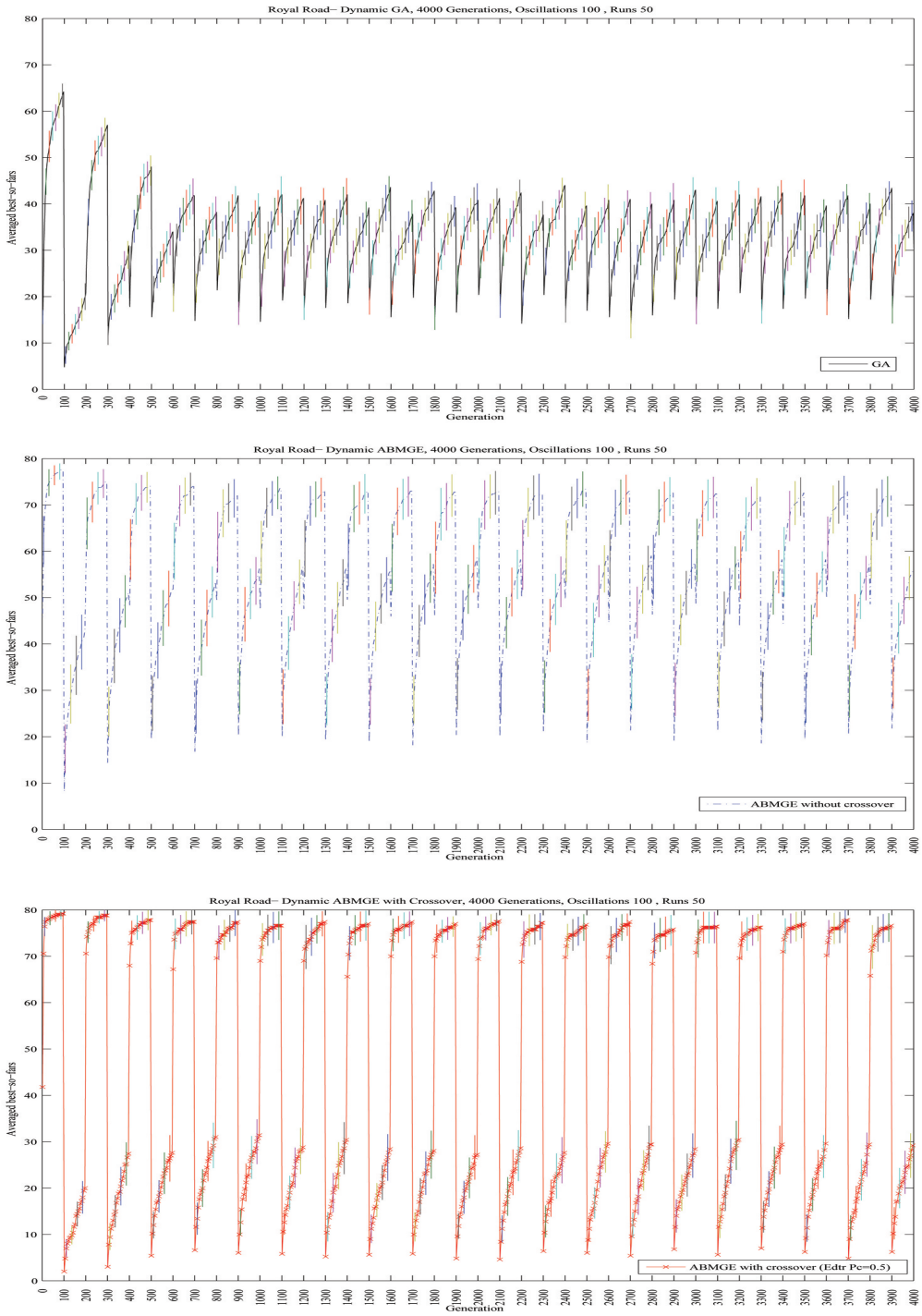

Figure 17: Performance of GA (a), ABMGE without crossover (b), and ABMGE with crossover ($P_{EdCross} = 0.5$) (c) on ORR, $p = 100$, for a simulation of 50 runs and 4000 generations.

tation performs best on the first fitness environment (SSR_1), and every time it repeats, but it performs very poorly on the second environment—even worse than the GA as generations progress. In contrast, the ABMGE version with editype mutation alone, is almost as good as the other version on the first environment, but much better on the second environment where it progressively improves its performance—well beyond that of the regular GA. Therefore, editype mutation alone seems to offer a much more flexible agent architecture in drastic environmental changes, far outperforming the traditional GA. Indeed, the ABMGE with editype mutation alone, is capable of producing agents which do well on both dramatically different fitness landscapes—as opposed to the GA agents which get worse and worse and settle to agents that are mediocre in both fitness landscapes.

While Editype crossover, as here implemented, does not allow agents to adapt to a drastic environmental change, unlike the GA, it is capable of recovering its performance once the environment changes back to the first state. This seems to indicate that, in this case, the editype preserves a kind of memory of the first environment it encounters, where it gets fixed. Whereas editype mutation alone provides flexibility as agents move from one environment to the next, editype mutation plus editype crossover does not render this flexibility, but it allows the performance on the first environment to remain high every time it repeats. In future work, we intend to investigate why this happens as well as other forms of editor variation.

5.2.2 Dynamic Schaffer Function DF_7

The ORR implements a drastically changing environment. In general, when we study dynamic environments, we consider less drastic changes. Angeline (1997) and Bäck (1998) reported a study of dynamic problems with three different modes of severity of changes. We use this idea to build a *dynamic version of the modified Schaffers function*, DF_7, based on the static Shaffer function F_7 (equation 3) used in section 5.1.4 but now controlled by a parameter s that specifies the *severity* of fitness change:

$$df_7(\overline{X}) = 2.5 - (X_1^2 + X_2^2)^{0.25}[sin^2(50(X_1^2 + X_2^2)^{0.1}) + 1] \tag{5}$$

where $X_i = x_i + \delta_s(t)$, $-1 \leq x_i \leq 1$ for $i = 1, 2$. t is used as an index for the environmental state, whenever the environment changes (e.g., every $p = 100$ generations), t is increased by 1. Finally, we implemented two methods to iteratively change the fitness function via the $\delta_s(t)$ update function, for a given constant severity s. The first method uses a linear update function (which we tested for $s = \{0.1, 0.5, 1\}$):

$$\begin{aligned} \delta_s(0) &= 0, \\ \delta_s(t) &= \delta_s(t-1) + s. \end{aligned} \tag{6}$$

The second method uses a jumping dynamics update function (which we tested for $s = \{0.1, 0.5\}$):

$$\begin{aligned} \delta_s(0) &= 0, \\ \delta_s(t) &= (-1)^t \cdot s \cdot (t+1)/2, \quad t \text{ odd} \\ \delta_s(t) &= (-1)^t \cdot s \cdot t/2, \qquad t \text{ even} \end{aligned} \tag{7}$$

A plot of this jumping dynamics update function is displayed in Figure 18 for severity $s = 0.5$. Here we only depict the results for the jumping dynamics update

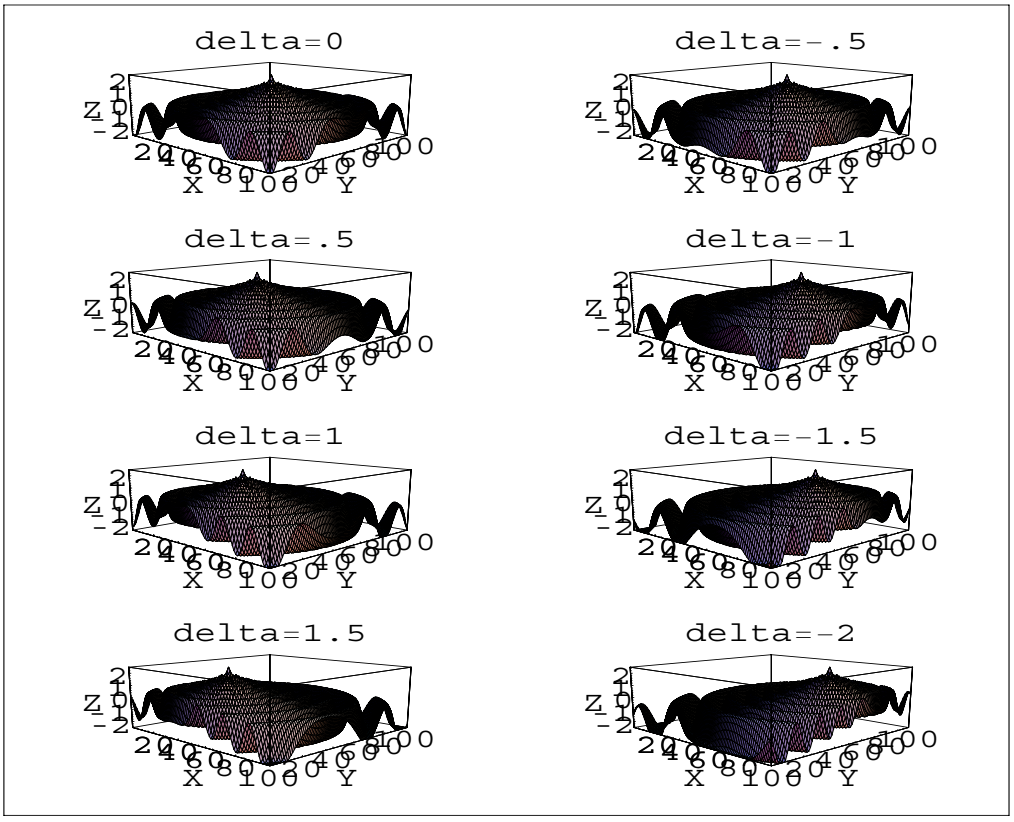

Figure 18: The jumping dynamic Schaffer function F_7 (severity = 0.5).

function with severity $s = 0.1$ [10]. But the results were quite comparable to what we obtained for other values of severity and update function tested. The ABMGE parameters of the dynamic environment simulations using DF_7 are the same as those used for the static F_7. We tried various editype mutation and crossover probabilities, for fitness function update periods: $p = \{50, 100, 200\}$. Figure 19 depicts the results for $P_{EdMut} = 0.05$ and $P_{EdCross} = 0.5$, editype editor family size $r \in \{1, \cdots, 20\}$, and period $p = 100$ for 1000 generations.

Both versions (with and without editype crossover) of the ABMGE significantly outperform the GA in this dynamic environment. However, given the confidence intervals, there is not a significative difference between the versions of the ABMGE. Therefore we cannot establish if editype crossover is beneficial as an editype variation mechanism on dynamic, multi-modal environments, or if mutation alone is preferable. But in all the tests we ran with the dynamic Schaffer DF_7 function, with the various parameters described above, the ABMGE significantly outperformed the GA.

5.2.3 Dynamic Optimal Control Function

In this subsection we test the ABMGE in a dynamic version of the optimal control problem studied in section 5.1.3. The constraints of the *dynamic optimal control* fitness function (DOC) are defined by equation 2 in section 5.1.3, except that here the control vari-

[10] Ramos et al. (2006) recently presented a study of this dynamic function with an evolutionary swarm algorithm.

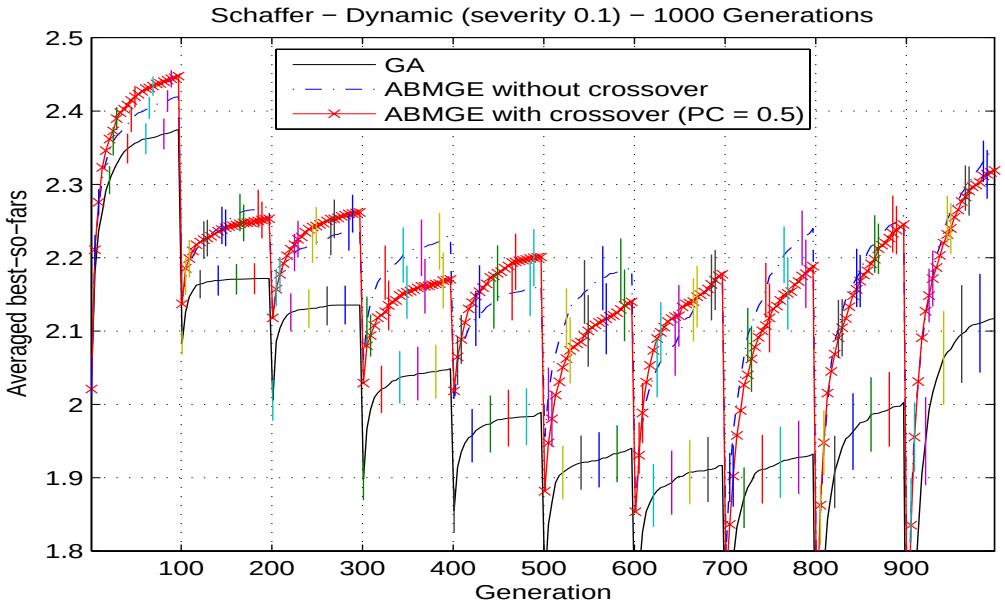

Figure 19: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on DF_7. Simulation depicted uses jumping dynamics update function with period $p = 100$.

ables are defined by: $U_i = u_i + \delta_s(t)$, $-1 \leq u_i \leq 1$ for $i = 1, 2$, where $\delta_s(t)$ is the fitness update function defined by either the linear or jumping dynamics of formulas 6 and 7 respectively. So far, we have only tested the DOC with the linear update function (6), which we tested for severity $s = \{0.1, 1\}$. A sketch of the DOC n for $\delta_s = 0$, 3, and 7 is illustrated on Figure 20.

In this subsection, each of the two variables is encoded by 50 bits, and thus each individual is a binary string of length 100. We again contrast the traditional GA with the ABMGE with and without editype crossover, using the same parameters as those used for the Dynamic Schaffer function in section 5.2.2, except editype editor family size $r \in \{1, \cdots, 10\}$. Figures 21 and 22 display the averaged best-so-far performance for severity $s = 0.1$ and $s = 1$, respectively.

Our results show that the genotype editing is again advantageous in tracking the extrema of the search space as they change in time. Indeed, both versions of the ABMGE quite significantly outperfom the GA. However, editype crossover in this dynamic environment is detrimental: the ABMGE without editype crossover (using editype mutation alone) is significantly better than the ABMGE with editype crossover.

5.2.4 Dynamic Schwefel's Function

Using the same methodology used for the DF_7 and DOC (sections 5.2.2 and 5.2.3), we also tested a dynamic version of the Schwefel function described in section 5.1.5. So far, we have only tested the *Dynamic Schwefel's Function* (DSF) with the linear update function (6), which we tested for severity $s = \{30, 50\}$ and update period $p = \{50, 100\}$. Here we only depict the results for $s = 50$ and $p = 50$ for 1000 generations with population size of 30 agents (Figure23). The results for other cases are quite similar, tough shorter update periods are slightly more favorable for the ABMGE. As it can be seen, the ABMGE significantly outperforms the GA in a majority of different fitness periods

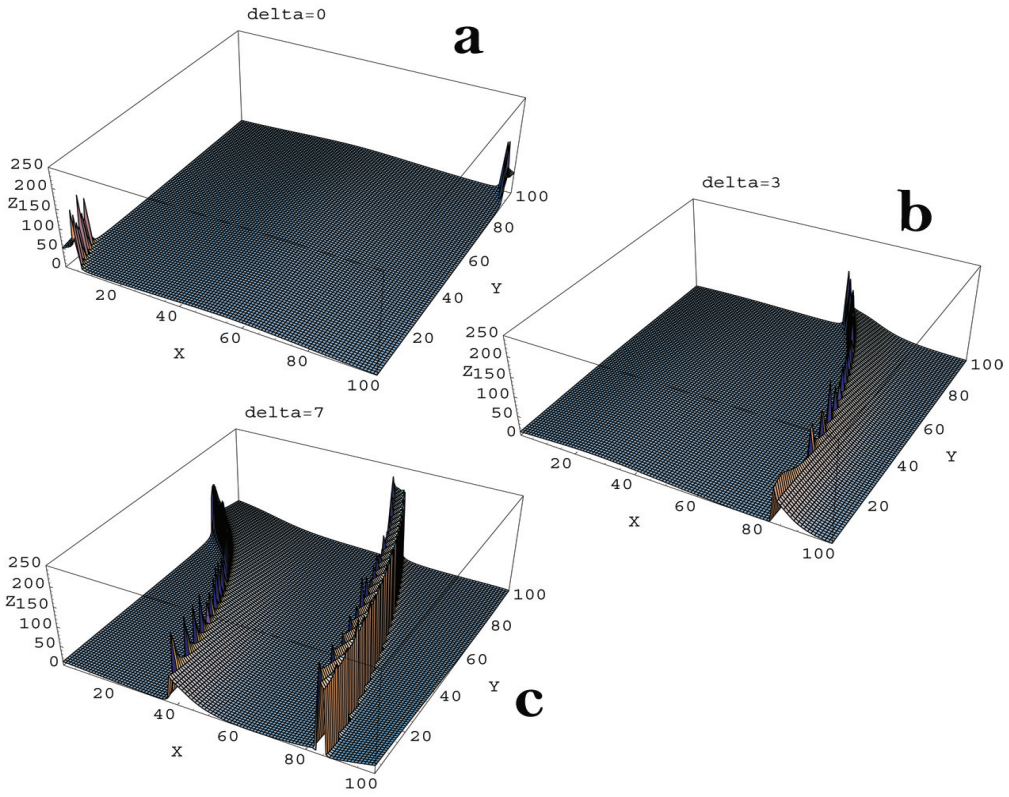

Figure 20: Linear dynamic optimal control function: a) $\delta_s = 0$, b) $\delta_s = 3$, and c) $\delta_s = 7$.

(14 out of 20), though the GA does outperform the ABMGE in a few (3 out of 20). Furthermore, there are 3 periods (out of 20) where neither one significantly outperforms the other. Finally, there seems to be no significant difference between the two types of ABMGE tested (with and without editype crossover).

6 Exploration vs. exploitation and genotype editing

To understand how genotype editing operates in the search process, we looked at the distribution of fitness values in a population for both the GA and the ABMGE (with mutation and without crossover). The study we present in this section looks only at the Dynamic Schwefel function tested in section 5.2.4 [11].

Figure 24 depicts the distribution of fitness values for the GA in the simulation depicted in section 5.2.4, for the first 100 generations, but only for a single run of the GA. The fitness update occurs between generation 49 and 50. This figure shows that just before the fitness function changes, the population has converged on almost a single value of fitness: most agents are identical and produce the same fitness value. As the fitness function changes (generation 50), the population of identical agents is now almost exclusively formed by very low fitness agents.

Figure 25, similarly, depicts the distribution of fitness values for the ABMGE (without crossover) for the same environment, again only for a single run of the ABMGE. In

[11] This dynamic function is the least favorable of those we tested for the ABMGE

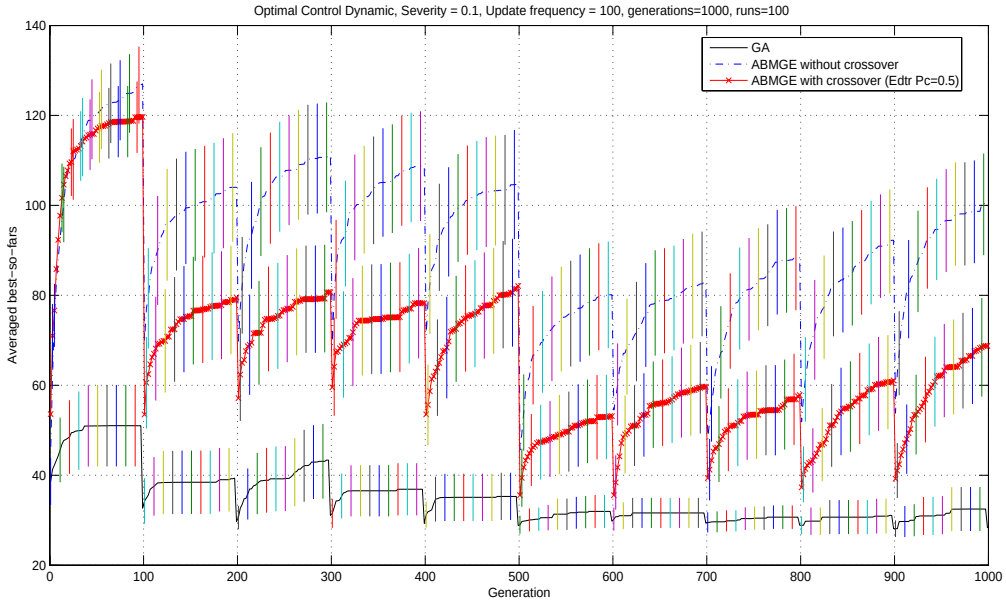

Figure 21: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on the DOC function. Simulation depicted uses a linear update function with period $p = 100$, severity $s = 0.1$, for 1000 generations.

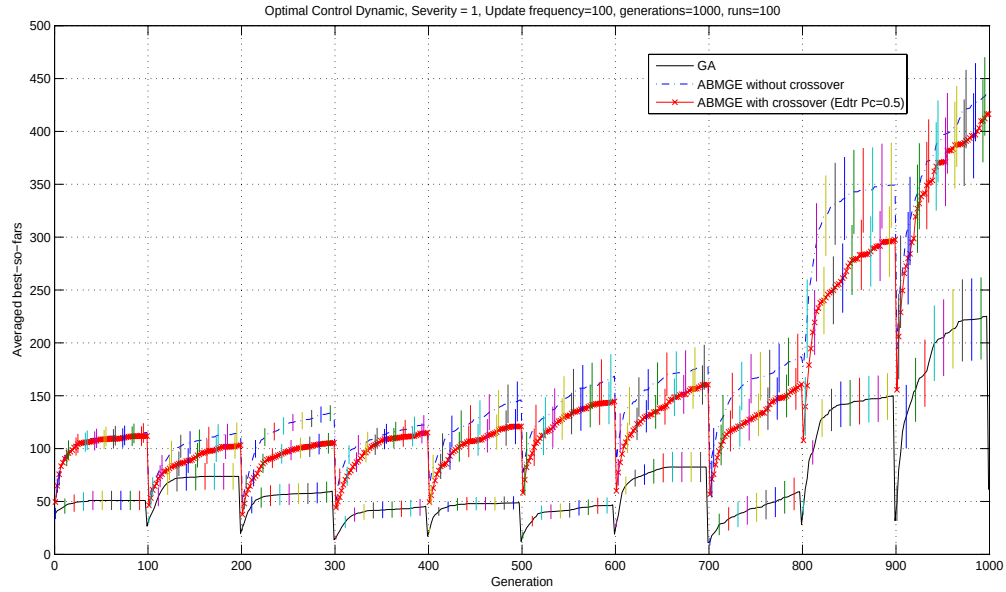

Figure 22: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on the DOC function. Simulation depicted uses a linear update function with period $p = 100$, severity $s = 1$, for 1000 generations.

this case, we can clearly see that the distribution of fitness values of the population of agents is much more spread out. Because genotype editing is a stochastic process, even identical agents (with the same codotype and editype) will produce different pheno-

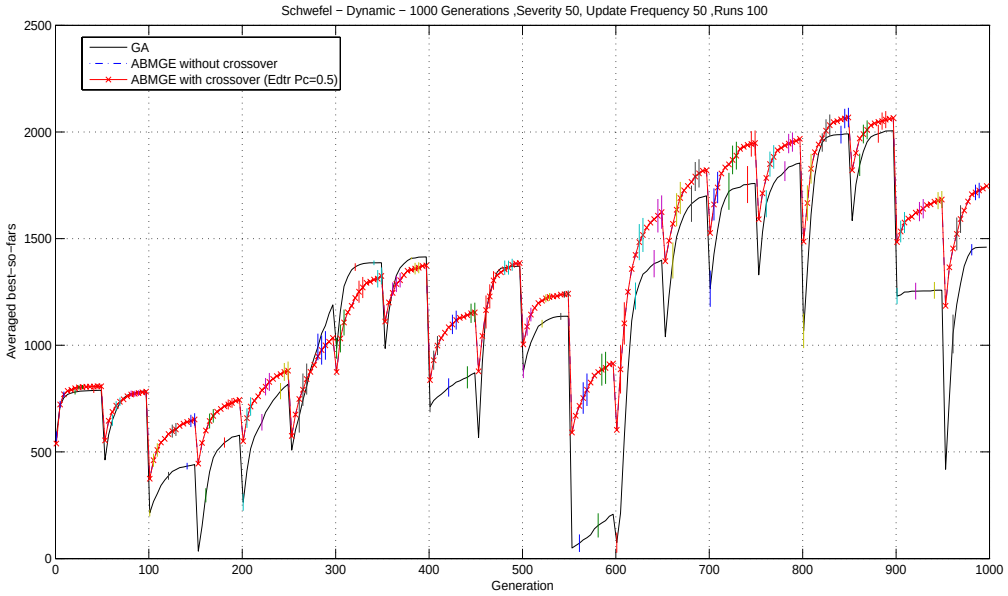

Figure 23: Performance of GA, ABMGE without crossover, and ABMGE with crossover ($P_{EdCross} = 0.5$) on the DSF. Simulation depicted uses a linear update function with period $p = 50$, severity $s = 50$, for 1000 generations.

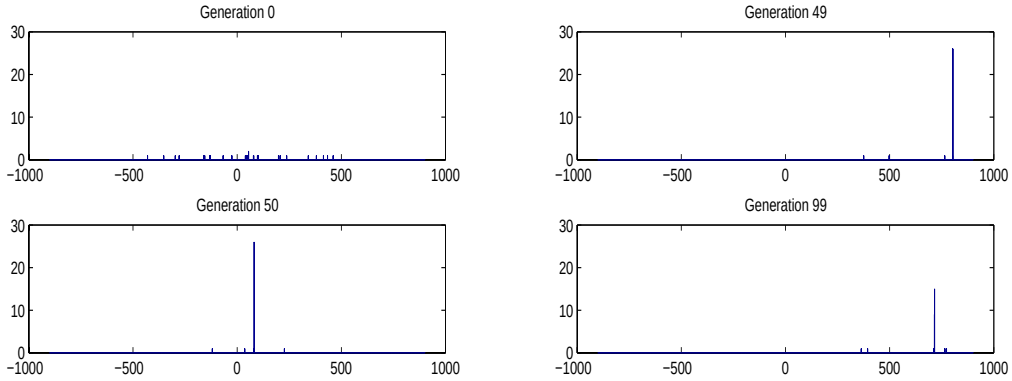

Figure 24: Fitness distribution for a single run of the GA on the DSF for 100 generations. Simulation depicted uses a linear update function with period $p = 50$, severity $s = 50$.

types. This way, and as initially predicted by Rocha (1995), we can see that genotype editing leads evolutionary agents to search their fitness space with a stochastic "cloud" of phenotype solutions, rather than a single phenotype. However, even though each agent can produce a different phenotype per generation, only one genotype is inherited and subjected to variation. If the genotype is repeated in the population, different phenotypes can exist in the population which are produced from the same genotype. This way, as can be seen in the figure, as the fitness function changes (generation 50), even in the new environment, the population of agents is capable of producing a few higher fitness phenotypes.

We can see that, with its stochastic phenotypic diversity, the ABMGE is perma-

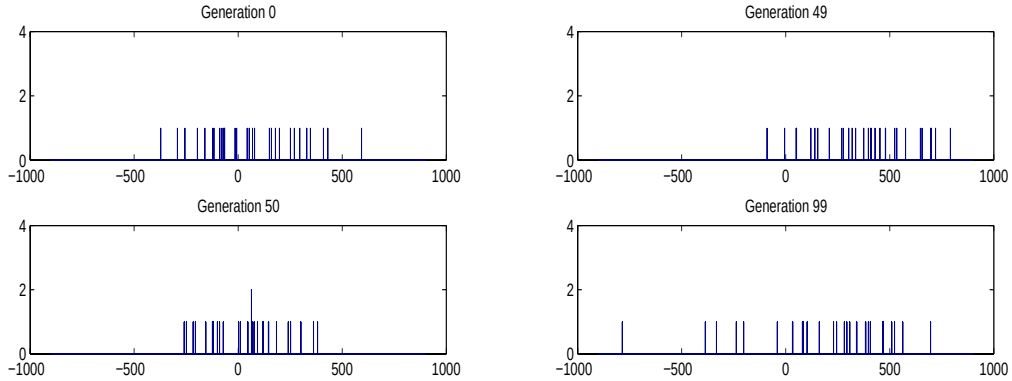

Figure 25: Fitness distribution for a single run of the ABMGE on the DSF for 100 generations. Simulation depicted uses a linear update function with period $p = 50$, severity $s = 50$.

nently exploring more of the search space than the GA. Evolution naturally leads the search to *exploit* areas of the codotype space which tend to result in good phenotypes, but given the stochastic "expression" of phenotypes, one single genotype (codotype plus editype) can *explore* different areas of the phenotype fitness space. Indeed, because the same codotype can result in different phenotypes, the ABMGE can exploit a fixed, good area of the codotype space, while producing different phenotypes—which is only possible because the ABMGE instantiates an indirect, stochastic genotype/phenotype mapping. Thus, the ABMGE seems to achieve a better exploitation/exploration balance than the GA—which is particularly useful in changing environments. This can be clearly seen when we calculate the distribution of fitness values for both algorithms, for all the 100 runs of our simulation in section 5.2.4. Figure 26 depicts the distribution of fitness values for 100 runs of the GA for 200 generations, and figure 27 depicts the same for the ABMGE. It is obvious from these figures that the ABMGE explores more of the fitness space due to its stochastic phenotype "expression"[12]. Furthermore, as can be seen in figure 23, the ABMGE statistically and significantly tends to find solutions with higher fitness.

7 Editing and Variation

Our study is an effort to understand the effect of genotype editing on equivalent genotypes. In other words, our approach was to investigate the behavior of genotypes with and without editypes by fixing all codotype parameters. However, it could be the case that the observed benefits of genotype editing result simply from the benefits of additional variation. As we discussed in section 6, it is clear that the search process instantiated with genotype editing is quite distinct from mere additional variation. While edits allow a wider exploration of the search space, the various phenotypic solutions produced by genotype editing are not directly inherited. It is the unedited codotype that is inherited (together with its editype).

In the ABMGE, if the phenotypes stochastically produced from the same genotype tend to be weak solutions, the genotype will disappear from the population. But, if the genotype tends to lead to good phenotypes, even producing a few weak phenotypic

[12]Videos of the distributions of fitness values as they change in time, for the simulations depicted in the figures in this section and for other simulations, are available online at http://informatics.indiana.edu/rocha/editing.

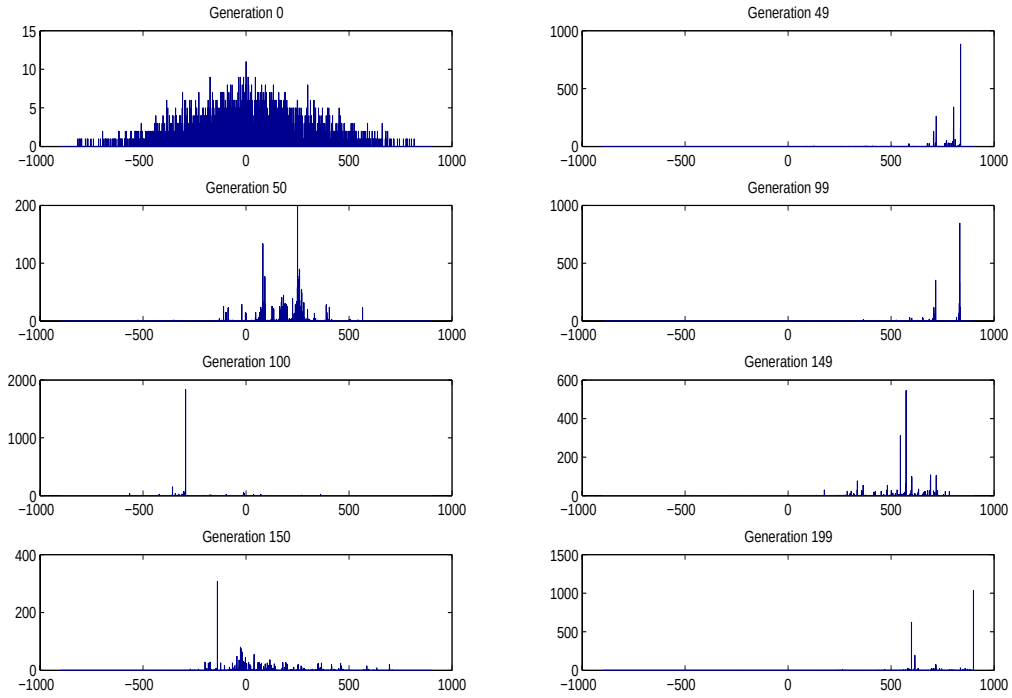

Figure 26: Fitness distribution for 100 runs of the GA on the DSF for 200 generations. Simulation depicted uses a linear update function with period $p = 50$, severity $s = 50$.

variations, the genotype may remain in the population—especially if it is repeated. In this sense, we think of genotype editing as a stochastic production of various phenotypes, which are nonetheless produced from and constrained by a single genotype. In contrast, any variation of the codotype is inherited. In a GA without editype, any variation is inherited leading to a parallel, unconstrained search. Therefore, in an evolutionary search, excessive variation is typically deleterious.

In any case, to address the question of whether the benefits of editing are similar to increased variation, we conducted some preliminary tests that we summarize here. We present this study solely for the static and dynamic Small Royal Road test function. We plan to expand on this topic in future work, where we will expand the study with additional fitness functions and larger search spaces.

When we originally selected the appropriate codotype mutation rate used in the Small Royal Road (section 5.1.1) tests, we selected the best variation parameters we found for the GA: $P_{Cross} = 0.7$ and $P_{Mut} = 0.005$. Figure 28 depicts three runs of the GA with three distinct values of P_{Mut} as well as the same run of the ABMGE without editype crossover shown in (section 5.1.1). All other parameters remain the same. As it can be clearly seen, the best performance of the GA is observed with the parameters we used, which lead to a lower performance than the ABMGE in the same circumstances.

In the dynamic environment of the Oscillatory Small Royal Road (ORR)(section 5.2.1), we observed that a higher (but not excessive) mutation rate was beneficial for the GA, though still inferior to the ABMGE. Recall that the ORR alternates between two maximally different Small Royal Road functions, SRR_1 and SRR_0, every period of p generations (see section 5.2.1). In figures 29 and 30 we use the ORR with $p = 100$

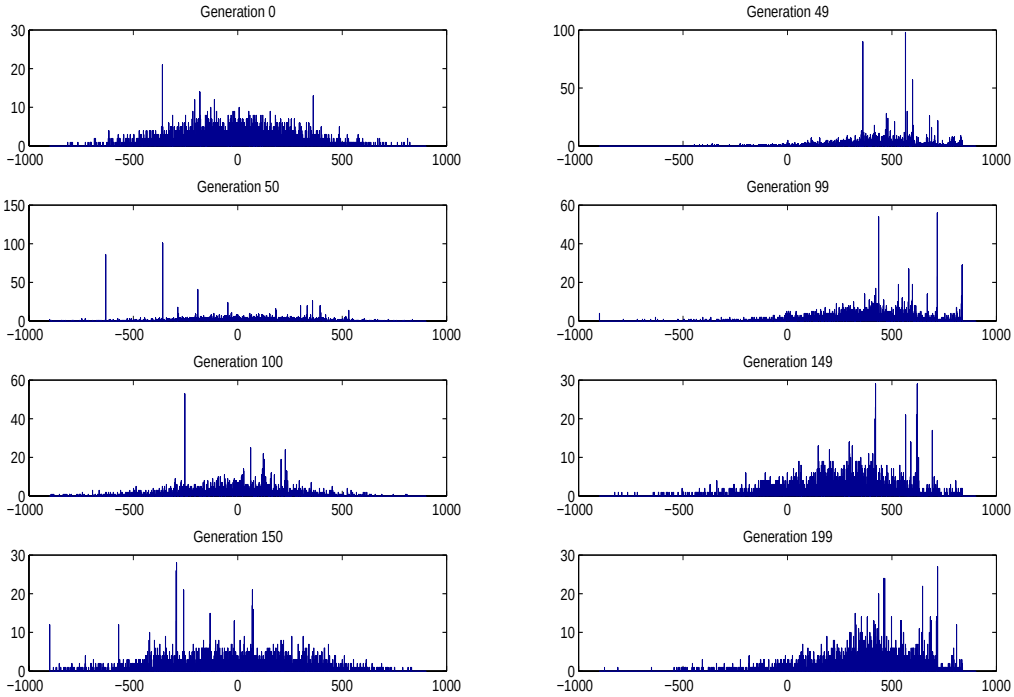

Figure 27: Fitness distribution for 100 runs of the ABMGE on the DSF for 200 generations. Simulation depicted uses a linear update function with period $p = 50$, severity $s = 50$.

to contrast the performance of the ABMGE without editype crossover, which we had already reported in section 5.2.1, with two versions of the GA with distinct mutation rates: $P_{Mut} = 0.005, 0.05$ [13].

Figure 29 depicts the first 1000 generations and figure 30 depicts the last 1000 generations. It is clear that the ABMGE far outperforms both versions of the GA in the first period (100 generations), and every time the first fitness function (SRR_1) repeats. We can also see in figure 29 that the GA with lower mutation is better than the one with higher mutation in the first period, but as the environment oscillates, higher mutation is beneficial for the GA, allowing the population to recover more effectively. Indeed, in the first few oscillations, the GA with higher mutation even outperforms the ABMGE in the second Royal Road environment (SRR_0). But as the evolutionary process continues, the ABMGE catches the higher mutation GA in its most adverse oscillation period (SRR_0)—see figure 30. This shows that the co-evolution of codotype and editype in the ABMGE allows the population of agents of the ABMGE to perform best in both states of the oscillating environment. A higher mutation is beneficial for the GA, but (in this environment) every time the environment oscillates, its agents have to cope with the new environment all over again. In contrast, the ABMGE recovers very quickly every time its first fitness function presents itself, and is capable of eventually responding well to the second fitness function. This seems to indicate that the introduction of an editype, in this fitness environment, grants a kind of evolved memory of previous environments which the GA does not attain.

We will explore the results described in this section much further in subsequent

[13] The results for $PMut = 0.5$ where quite inferior, so we do not display or discuss them.

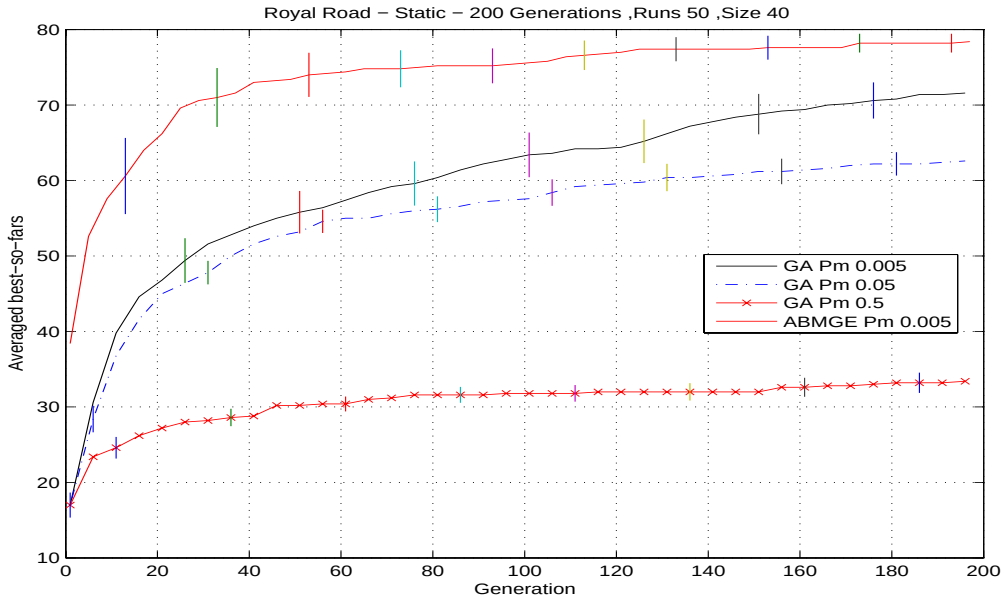

Figure 28: ABMGE with $p_{Mut} = 0.005$ contrasted with three GA with $p_{Mut} = 0.005$, 0.05, and 0.5, on SRR_1

work. However, it is clear from this preliminary study of varying mutation rates, that the evolutionary search provided by the ABMGE is not simply a case of increased variation. The inclusion of an editype, in addition to the constrained stochastic search described in section 6, seems to provide a means for agents to cope better in dynamic environments, seemingly by establishing an evolvable memory of previous environments as discussed in this section.

8 Discussion

We introduced an evolutionary model of genotype editing based on agents endowed with coding and non-coding portions of their artificial genome: a codome and an editome. The non-coding editome is used to alter encoded solutions ontogenetically, without inheritance. We furthermore tested different forms of variation on the editome, independently from the codome, on several static and dynamic environments, and showed that genotype editing almost always significantly outperforms the traditional GA which does not include an editome.

The comparison of our algorithm with a simple GA is not meant to imply that our method is the best possible evolutionary optimization algorithm for the fitness functions tested. Our goal is above all to understand how genotype editing works and what kind of search process it leads to. Therefore, we compared the ABMGE to the canonical genetic algorithm. We show that the ABMGE, under most circumstance tested, improved upon the GA. This means that genotype editing generally improves on the canonical evolutionary search process, not that it is the best algorithm for the functions tested. Indeed, other authors have tested different algorithms (e.g. particle swarms) on some of our test functions with better results than the ABMGE (Ramos et al. (2006)).

In static environments, the ABMGE outperformed the GA in every function we tested. When it comes to the editype crossover mechanism, it is clearly an advantage

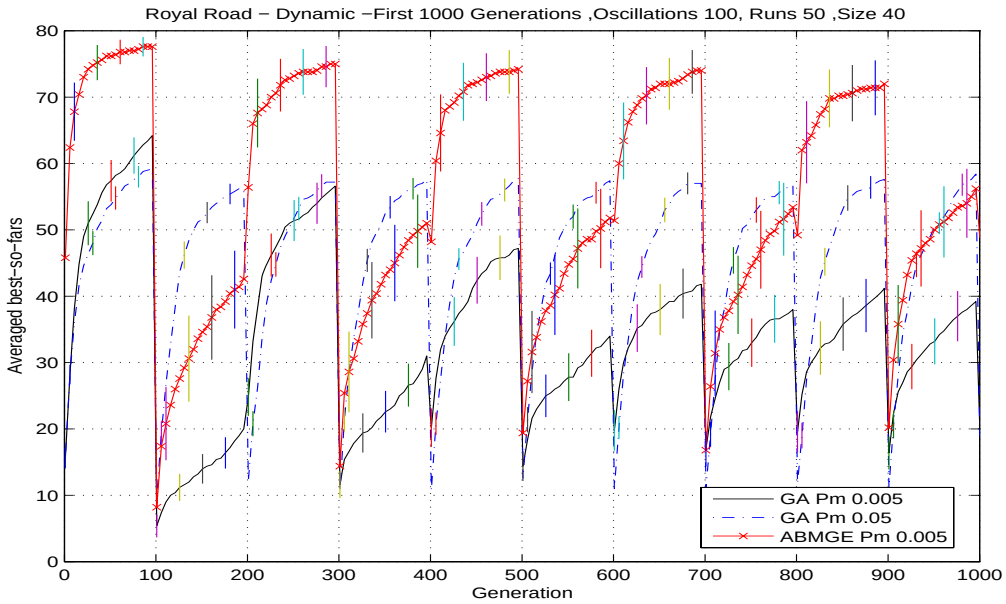

Figure 29: Performance of GA and ABMGE without crossover on ORR, $p = 100$, for the first 1000 generations of a simulation of 50 runs and 4000 generations. ABMGE with $p_{Mut} = 0.005$ contrasted with two GA with $p_{Mut} = 0.005$ and 0.05

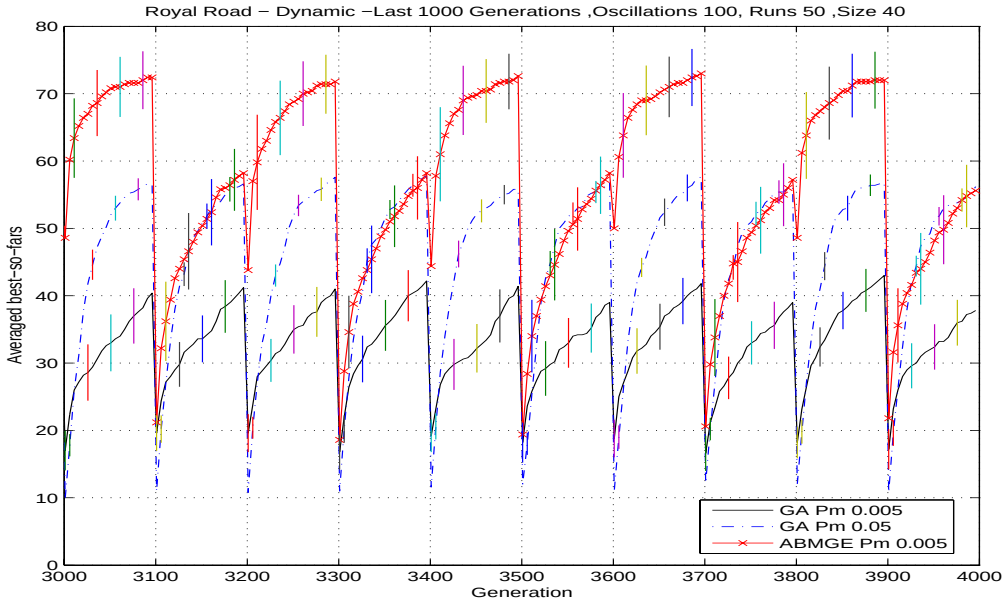

Figure 30: Performance of GA and ABMGE without crossover on ORR, $p = 100$, for the last 1000 generations of a simulation of 50 runs and 4000 generations. ABMGE with $p_{Mut} = 0.005$ contrasted with two GA with $p_{Mut} = 0.005$ and 0.05

in the amenable fitness environments defined by the Small Royal Road (section 5.1.1) and the De Jong (section 5.1.2) functions. It was also advantageous in the static, multi-modal Schaffer function (section 5.1.4). However, its performance was statistically in-

distinguishable from the ABMGE with editype mutation alone for the other functions tested: Optimal Control (section 5.1.3) and Schwefel (section 5.1.5). In any case, in static environments, editype crossover was never detrimental.

In dynamic environments, the ABMGE also globally outperformed the GA in every function tested. Indeed, the ABMGE outperformed the GA in every period of these simulations, except in the case of the dynamic Schwefel function 5.2.4 where the GA outperformed the ABMGE in a very small number of fitness periods. However, even in this case, the ABMGE outperformed the GA in a large majority of periods. Thus we can conclude that even in the case of the dynamic Schwefel function, the ABMGE globally outperformed the GA.

Furthermore, in dynamic environments, the difference between the ABMGE and the GA tends to get more pronounced. This is clear in the case of the Oscillating Royal Road function (sections 5.2.1 and 7), where the two oscillating fitness functions are maximally incompatible. While the GA settles to agents that are mediocre in both oscillating environment, the ABMGE (without editype crossover) uses genotype editing to produce phenotypes that remain excellent in the first fitness environment encountered (and its subsequent re-appearances), and are quite good in the second environment. On the other dynamic functions tested, the changing environments are not as incompatible as is the case of the ORR, but here too, genotype editing as implemented by the ABMGE provides an evolutionary advantage in tracking the changing extrema.

Finally, in dynamic environments, while the ABMGE with editype crossover clearly outperformed the GA, it never provided a clear advantage over the ABMGE with editype mutation alone. Therefore, our implementation of editype crossover is not useful in dynamic environments. One interesting result with editype crossover is the case of the Oscillating Royal Road function (section 5.2.1). While this version of the ABMGE performs rather poorly on the second fitness function (even worse than the GA), every time the environment switches back to the first fitness function, the agents regain very high fitness values (even better than the ABMGE without editype crossover). This means that the ABMGE with editype crossover, in this dynamic environment, is capable of recuperating the solutions it reached in the first environment the first time around. While it does not show the flexibility of the ABMGE with editype mutation alone in adapting to changing environments (see section 7), it does preserve a very stable and effective memory of the first environment found (which the GA does not).

Thus, we conclude that the co-evolving genotype editing mechanism offers a significant evolutionary advantage. This advantage is particularly interesting in dynamic environments as agents become better equipped to deal with changing environments. Indeed, they both react quicker to the change and produce fitter agents. While our highly idealized models do not capture the reality of Biology, they do imply that the process of RNA editing in nature is, likewise, advantageous in evolution. Indeed, our results emphasize the importance of genetic regulation by non-coding genetic components. The stochastic regulation via genotype editing we tested here, instantiates an indirect genotype/phenotype mapping capable of enhanced exploration of the fitness space, without loosing the ability to exploit good areas of the genotype space.

In conclusion, the performance of our agent-based algorithms positions them as promising novel methods for evolutionary computation and machine learning. We have thus advanced the understanding of the evolutionary role of RNA Editing, and we have developed a new biologically-inspired algorithm with powerful search capabilities—meeting the two goals we set up for this project.

There are many other aspects of the ABMGE that one can study to further enhance the power of this agent-based evolutionary algorithm. Naturally, we intend to apply them to additional problems, but we also intend to study the influence of other forms of editor variation, and allow the variation of other editor parameters such as editor length, function, concentration, etc. There is also much to investigate regarding dynamic environments. Not only do we expect to study other dynamic landscapes, but we need to investigate more deeply the influence of the changing period on performance. Finally, we intend to investigate more biologically realistic ways of implementing an editype and a codotype, namely by exploring and expanding various artificial genome models in the literature.

9 Acknowledgements

The Indiana University's supercomputing facilities used in our analysis are funded in part by NSF under Grant No. 0116050 and Grant CDA-9601632. We are grateful to IU's Research and Technical Services for technical support. We are also grateful to the FLAD Computational Biology Collaboratorium at the Gulbenkian Institute in Oeiras, Portugal, for hosting and providing facilities used to conduct part of this research.

References

Angeline, P. J. (1997). Tracking extrema in dynamic environments. In *Proceedings of the Sixth International Conference on Evolutionary Programming*, pages 335–345.

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

Bäck, T. (1998). On the behavior evolutionary algorithms in dynamic environments. In *IEEE International Conference on Evolutionary Computation*, pages 446–451.

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

Benne, R., Van den Burg, J., Brakenhoff, J., Sloof, P., and Tromp, M. (1986). Major transcript of the frameshifted coxII gene from trypanosome mitochondria contains four nucleotides that are not encoded. *Cell*, 46(6):819–826.

Blanc, V. and Davidson, N. O. (2003). C-to-U RNA editing: Mechanisms leading to genetic diversity. *Journal of Biological Chemistry*, 278(3):1395–1398.

Chilibeck, K., Wu, T., Liang, C., Schellenberg, M., Gesner, E., Lynch, J., and Macmillan, A. (2006). FRET analysis of In Vivo dimerization by RNA editing enzymes. *J Biol Chem*, 281(24):16530–16535.

De Jong, K. (1975). *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor.

Dellaert, F. and Beer, R. (1994). Toward an evolvable model of development for autonomous agent synthesis. In Brooks, R. and Maes, P., editors, *Artificial Life IV*. MIT Press Cambridge.

Ficici, S. and Pollack, J. B. (2003). A game-theoretic memory mechanism for ecevolution. In *Proc. of 2003 Genetic and Evolutionary Computation Conference*, pages 286–297.

Forrest, S. and Mitchell, M. (1993). Relative building block fitness and the building block hypothesis. *Foundations of Genetic Algorithms*, 2:109–126.

Furey, T. S., Diekhans, M., Lu, Y., Graves, T. A., Oddy, L., Randall-Maher, J., Hillier, L. W., Wilson, R. K., and Haussler, D. (2004). Analysis of human mRNAs with the reference genome sequence reveals potential errors, polymorphisms, and RNA editing. *Genome Ressearch*, 14(10B):2034–40.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundation of Genetic Algorithms*, pages 69–93. Morgan Kaufmann.

Hager, W. W. and Pardalos, P. M. (1998). *Optimal Control: Theory, Algorithms and Applications*. Kluwer Academic Publishers.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, 1:495–502.

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

Hoopengardner, B., Bhalla, T., Staber, C., and Reenan, R. (2003). Nervous system targets of RNA editing identified by comparative genomics. *Science*, 301(5497):832–836.

Huang, C.-F. (2002). *A Study of Mate Selection in Genetic Algorithms*. Doctoral dissertation. Ann Arbor, MI: University of Michigan, Electrical Engineering and Computer Science.

Huang, C.-F. and Rocha, L. M. (2003). Exploration of RNA editing and design of robust genetic algorithms. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pages 2799–2806. IEEE Presss.

Huang, C.-F. and Rocha, L. M. (2004). A systematic study of genetic algorithms with genotype editing. In *Proc. of 2004 Genetic and Evolutionary Computation Conference*, volume 1, pages 1233–1245.

Huang, C.-F. and Rocha, L. M. (2005). Tracking extrema in dynamic environments using a co-evolutionary agent-based model of genotype edition. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 545–552, New York, NY, USA. ACM Press.

Kargupta, H. (1996). The gene expression messy genetic algorithm. In *International Conference on Evolutionary Computation*, pages 814–819.

Maas, S., Rich, A., and Nishikura, K. (2003). A-to-I RNA editing: Recent news and residual mysteries. *Journal of Biological Chemistry*, 278(3):1391–1394.

Mattick, J. S. (2003). Challenging the dogma: the hidden layer of non-protein-coding RNAs in complex organisms. *BioEssays*, 25:930–939.

Mittaz, L., Antonarakis, S. E., Higuichi, M., and Scott, H. S. (1997). Localization of a novel human RNA-editing deaminase (h(red)2 or ADARB2) to chromosome 10p15. *Human Genetics*, 100:398–400.

Panait, L., Wiegand, R. P., and Luke, S. (2004). A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In *Proc. of 2004 Genetic and Evolutionary Computation Conference*, volume 1, pages 573–584.

Pollack, R. (1994). *Signs of Life: The Language and Meanings of DNA*. Houghton Mifflin.

Potter, M. and De Jong, K. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Ramos, V., Fernandes, F., and Rosa, A. (2006). On self-regulated swarms, societal memory, speed and dynamics. In Rocha, L. M., Yaeger, L. S., Bedau, M. A., Floreano, D., Goldstone, R. L., and Vespignani, A., editors, *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 393–399. MIT Press.

Reil, T. (1999). Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 457–466, London, UK. Springer-Verlag.

Rocha, L. M. (1995). Contextual genetic algorithms: Evolving developmental rules. In Moran, F., Moreno, A., Guervos, J. J. M., and Chacon, P., editors, *ECAL 1995: Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995*, volume 929 of *Lecture Notes in Computer Science*, pages 368–382. Springer.

Rocha, L. M. (2001). Evolution with material symbol systems. *Biosystems*, 60(1-3):95–121.

Rocha, L. M. and Huang, C.-F. (2004). The role of RNA editing in dynamic environments. In *The Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 489–494. MIT Press.

Rocha, L. M., Maguitman, A., Huang, C.-F., Kaur, J., and Narayanan, S. (2006). An evolutionary model of genotype editing. In Rocha, L. M., Yaeger, L. S., Bedau, M. A., Floreano, D., Goldstone, R. L., and Vespignani, A., editors, *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 105–111. MIT Press.

Schwefel, H.-P. (1981). *Numerical optimization of computer models*. Chichester: Wiley & Sons.

Simpson, L. (1999). RNA editing-an evolutionary perspective. In Atkins, J. and Gesteland, R., editors, *The RNA World*, pages 585–608. Cold Spring Harbor.

Simpson, L. and Emerson, R. B. (1996). RNA editing. *Annual Review of Neuroscience*, 19:27–52.

Stuart, K. (1993). RNA editing in mitochondria of african trypanosomes. In Benne, R., editor, *RNA Editing : The Alteration of Protein Coding Sequences of RNA*, pages 26–52. Ellis Horwood Publishers.

Stuart, K., Allen, T., Kable, M., and Lawson, S. (1997). Kinetoplastid RNA editing: complexes and catalysts. *Curr Opin Chem Biol*, 1:340–346.

Sturn, N. R. and Simpson, L. (1990). Kinetoplast DNA minicircles encode guide RNA's for editing of cytochrome oxidase subunit III mRNA. *Cell*, 61:879–884.

Wang, Q., Khillan, J., Gadue, P., and Nishikura, K. (2000). Requirement of the RNA editing deaminase ADAR1 gene for embryonic erythropoiesis. *Science*, 290(5497):1765–1768.