On the Computational Power of Adaptive Systems

Carlos Cotta CCOTTAP@LCC.UMA.ES

Enrique Alba EAT@LCC.UMA.ES

José M. Troya TROYA@LCC.UMA.ES

Dept. of Lenguajes y Ciencias de la Computación

Campus de Teatinos (Room 2.2.A.6), E-29071-Málaga, SPAIN

Abstract

Recent research has demonstrated that no search algorithm is better than any other one when performance is averaged over all possible discrete problems. Hybridization (incorporation of problem-knowledge) is required to produce adequate problem-specific algorithms. This work explores the power of hybridization in the context of evolutionary algorithms. For this purpose, a framework for describing adaptive systems is presented. It is shown that, when hybridized, adaptive techniques are computationally complete systems with Turing capabilities. Moreover, evolutionary algorithms can be regarded as a kind of nondeterministic Turing machines.

Keywords: adaptive systems, problem solving, calculability, hybridization, termination criterion

1. Introduction

The paradigm of evolutionary computation (Evolutionary Programming [14], Genetic Algorithms [17], Evolution Strategies [22]) is a hot topic these days. The simplicity of these techniques and the implications of representing a computational metaphor of Nature have motivated the interest of many researchers. Nowadays it is possible to find a great amount of scientific papers and practical interdisciplinary applications of evolutionary algorithms [1]. Hence, they are considered robust and effective heuristics, easily applicable to any problem.

However, it is a well-known fact that robustness and efficiency are opposed. Traditionally, it has been thought that the quality of the solutions provided by an evolutionary algorithm is good when compared with non-robust specialized techniques [15] in spite of the loss of efficiency caused by their robustness. However, this has been shown to be a misconception as demonstrated in [16] and popularized with the so-called *No Free Lunch* Theorem [25]. These results set a lower bound for this robustness. Essentially, it is proved that any algorithm will yield the same expected performance for any chosen measure when performance is averaged over all possible discrete problems. As a result, no algorithm can be considered as a general and effective resolution tool if only a black-box evaluation function is provided.

Nevertheless, it should be noted that the mentioned limitation just applies to individual search algorithms. However, the evolutionary computation paradigm does not comprise a single technique but a large family of algorithms sharing a certain

underlying template. The question is whether there exists an intrinsic limitation on the applicability of this template. In other words, to determine whether it is possible to develop algorithms within the framework of evolutionary computation tailored to any domain. The answer to the latter question is yes, provided that enough problem-knowledge is included into this template, thus yielding a hybrid algorithm [9]. The first contribution of this work is to formalize this statement. In this sense, it is shown that the mechanism of hybridization is powerful enough to allow the definition of computationally complete systems when applied to the template of an evolutionary algorithm. More precisely, it is shown that a hybrid evolutionary algorithm can simulate the behavior of a Turing machine.

The remainder of this work is organized as follows. First, a brief overview of evolutionary algorithms is given in section 2. Then, a framework for hybridization in the context of adaptive systems is outlined in section 3. This framework is used to describe a hybrid evolutionary algorithm that simulates a Turing machine in section 4. Subsequently, evolutionary algorithms are shown to be a subclass of nondeterministic Turing machines in section 5. Finally, some conclusions are summarized in section 6.

2. Evolutionary Algorithms in Perspective

Evolutionary algorithms are complex adaptive systems based on the principles of natural evolution, namely adaptation and survival of the fittest. They maintain a population (pool) of candidate solutions whose adaptation is measured by means

of a *fitness* function. This population evolves through the application of several evolutionary operators: selection, reproduction and replacement. Reproduction can be usually divided in the successive application of some reproductive operators.

The several flavors of evolutionary algorithms differ in how individual solutions are represented and in the way they evolve. Traditionally, a well-known behavior has been assumed for reproductive operators. To be precise, solutions are considered to be decomposable in a list of features (*schemata* [17], *predicates* [23] or *formae* [19]), with independence of whether they are internally represented in the algorithm in this way or not. New tentative solutions are produced by modifying some of these features (mutation) or by combining the features of several solutions (recombination). In this sense, representation-independent operators have been defined with this behavior (e.g., *Binomial Minimal Mutation* [20], *Random Respectful Recombination* [19], etc.). In such a scenario, the ideal representation of solutions should induce a linearly separable mapping from solutions to their fitness, i.e., the fitness of a solution should be the sum of the contribution of each feature. Notice that, as pointed out in [21], if such a representation is possible, no complex adaptive system is required since a simple greedy hill-climber would perform optimally. With these premises, evolutionary algorithms should be appropriate in intermediate domains between linearly separable and fully epistatic regions.

However, since a given evolutionary algorithm (e.g., a generational genetic algorithm using one-point crossover and bit-flip mutation) is clearly better than random

search in separable domains (where salient features can be identified and propagated) and provides the same performance in random fitness landscapes (where no organization can be exploited), there must be problems in the intermediate regions in which it is outperformed by random search (Fig. 1). This fact is a direct consequence of the *No Free Lunch* Theorem. This theorem states that the performance of any two search algorithms is the same when averaged over all problems, assuming performance metrics that do not account for resampled points and a uniform distribution of problems. Plainly, the superiority (*lunch*) in a given domain of an algorithm $\mathcal{A}$ with respect to algorithm $\mathcal{B}$ has a price: worse overall performance in the remaining domains.

`Figure 1 near here`

There exists an ongoing debate about the implications of this result. Some authors consider that the assumptions on which it is based are unrealistic. The argument is twofold: first, the class of all problems is too broad; second, evolutionary algorithms are potentially retracing techniques. As to the first point, it should be noted that there exists no mathematical formulation of *real world* problems and hence no problem should be discarded in advance. Moreover, the intermediate regions (in which random search sometimes outperforms evolutionary algorithms) constitute the most interesting zone as pointed out in [8]. As to the second point, recent research [2] has shown that the effects of resampling are mostly significant in advanced stages of the search (where the algorithm is stagnated) and become negligible as the size of the problem grows.

In any case, notice that any evolutionary algorithm $\mathcal{A}$ is a particular instance of a general search template $\mathcal{S}$, i.e., $\mathcal{A} \triangleq \mathcal{S}(x_1, \cdots, x_n)$, where the parameters x_i represent elements such as representation, operators, evolution model, etc. An interesting point is how $\mathcal{S}(\cdot)$ compares to any $\mathcal{B}$. In other words, can we instantiate this template to perform adequately in arbitrary domains? The answer is yes, as demonstrated in the following sections.

3. A Framework for Hybrid Adaptive Systems

In its broadest sense, algorithmic hybridization is the incorporation of precise problem-knowledge into a general-purpose algorithm. From the point of view of evolutionary algorithms, two major mechanisms can be considered [6]: using problem-specific representation and operators (*strong* hybridization) or combining several algorithmic techniques (*weak* hybridization). This section presents a framework to describe hybrid systems using the first mechanism[1]. This framework is based on the formalism originally defined by Holland [17] and later extended by Daida *et al.* [7]:

Definition 1 (Adaptive System). An adaptive system is defined as a tuple $\mathcal{S} = (\mathcal{A}, \Omega, \tau, \mathcal{B}, \Omega_{\mathcal{B}}, \tau_{\mathcal{B}}, \mathcal{I})$ where

- $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots\}$ is the set of attainable structures.

1. We focus on strong hybridization for two reasons: first, both models of hybridization have been shown to be equally powerful [3]. Second, weak hybridization is a substantial topic for itself and hence we defer its study to a further work.

- $\Omega = \{\omega^1, \omega^2, \cdots\}$ is a set of operators $\omega^i : \mathcal{A} \to \mathcal{A}$ to modify structures in $\mathcal{A}$.
- $\tau : \mathcal{I} \times \mathcal{A} \to \Omega$ is an adaptive plan deciding which operator is to be applied on a structure $\mathcal{A}$ given input $\mathcal{I}$.
- $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \cdots\}$ is the set of artifacts used by $\mathcal{S}$.
- $\Omega_{\mathcal{B}} = \{\omega^1_{\mathcal{B}}, \omega^2_{\mathcal{B}}, \cdots\}$ is a set of operators $\omega^i_{\mathcal{B}} : \mathcal{B} \times \mathcal{A} \to \mathcal{B}$ to modify artifacts in $\mathcal{B}$ according to the internal structure.
- $\tau_{\mathcal{B}} : \mathcal{I} \times \mathcal{B} \to \Omega_{\mathcal{B}}$ is an adaptive plan analogous to τ.
- $\mathcal{I}$ is the set of possible inputs to the system from an environment E.

This definition requires some explanation. First of all, the adaptive system $\mathcal{S}$ is supossed to be embedded within an environment E which represents the target problem. The system has an internal structure $\mathcal{A}(t)$ at time t. This structure is usually decomposable into some substructures $\mathcal{A}^1_i(t)$ as shown in Fig. 2 (right), and undergoes adaptation via the adaptive plan τ. This plan determines structural modifications in response to the information $\mathcal{I}(t)$ supplied by the environment, usually interpretable as the *fitness* of an structure. More precisely, the adaptive plan selects an operator $\Omega_i(t) \in \Omega$ to modify $\mathcal{A}(t)$. The whole process is summarized in Fig. 2 (left). At this point, knowledge can be included defining appropriate internal structures ($\mathcal{A}_i$) and operators (ω^j), i.e., strong hybridization.

Figure 2 near here

Now, notice that there exists a parallelism between $(\mathcal{A}, \Omega, \tau)$ and $(\mathcal{B}, \Omega_{\mathcal{B}}, \tau_{\mathcal{B}})$. Roughly speaking, they can be seen as the *inside* domain and the *outside* domain respectively (see Fig. 2 -right-). Thus, the system manipulates as set of external artifacts $\mathcal{B}'(t) \subseteq \mathcal{B}$ embedded in the environment E. In many situations, the internal substructures correspond to internal representations (i.e., *genotypes*) of artifacts (i.e., *phenotypes*). This distinction is useful when several systems with different internal representations are combined by means of sharing some artifacts in $\mathcal{B}$, e.g., a heterogeneous parallel genetic algorithm [3].

Example 2.

The *threshold accepting* algorithm [12] is a very simple example of a system that exhibits a basic adaptive behavior. The functioning of this algorithm is similar to *simulated annealing* with the exception that the acceptance criterion is deterministic. To be precise a new solution is accepted whenever it improves the current solution or, at most, the fitness difference is not higher than a dynamic threshold value T. Assumming that a function $f : \mathcal{Z} \to \mathcal{Z}$ is to be minimized, the algorithm can be formulated as an adaptive system in which:

- The environment E comprises the function f. The set of inputs $\mathcal{I}$ from the environment is thus the image of $\mathcal{Z}$ under f.
- $\mathcal{A} = \mathcal{A}^1 \times \mathcal{M}$. The first term is $\mathcal{A}^1 = \{(\zeta, \xi, T) \mid \zeta, \xi, T \in \mathcal{G}\}$, where $\mathcal{G}$ is the set of discrete internal representations of whole numbers (e.g., standard binary

coding in complement to two). The first element of each tuple in $\mathcal{A}^1$ is the new value to be tested, the second one is the best-so-far generated value, and the third one is the current threshold value. As to $\mathcal{M}$, it represents the input history of the system, i.e., $\mathcal{M}(t) = \langle \mathcal{I}(1), \mathcal{I}(2), \cdots, \mathcal{I}(t-1) \rangle$.

- Let $\eta : \mathcal{G} \to \mathcal{G}$ be a mutation operator. Let $\psi : \mathcal{G} \to \mathcal{G}$ be a threshold updating function. Then, $\Omega = \{\omega^1, \omega^2\}$, where

  - $\omega^1((\zeta, \xi, T), \langle \iota_i, \cdots, \iota_t \rangle) = ((\eta(\zeta), \zeta, \psi(T)), \langle \iota_i, \cdots, \iota_t \rangle)$
  - $\omega^2((\zeta, \xi, T), \langle \iota_i, \cdots, \iota_t \rangle) = ((\eta(\xi), \xi, \psi(T)), \langle \iota_i, \cdots, \iota_t \rangle)$

  i.e., ω^1 is an operator that accepts ζ as the new best generated value and generates a η-neighbor of ζ, and ω^2 is an operator that discards ζ and generates a new η-neighbor of ξ. Both operators update the threshold value.

- The adaptive plan $\tau : \mathcal{I} \times (\mathcal{A}^1 \times \mathcal{M}) \to \Omega$ is defined as

$$\tau(\iota_t, ((\zeta, \xi, T), \langle \iota_1, \cdots, \iota_{t-1} \rangle)) = \begin{cases} \omega^1 & \text{if } (\iota_t - \iota^*) < T \\ \omega^2 & \text{otherwise.} \end{cases} \tag{1}$$

  where $\iota^* = \min(\iota_k : 1 \leq k \leq t-1)$.

The definition of $(\mathcal{A}, \Omega, \tau)$ suffices to specify the internal behavior of the system. It is now necessary to indicate the external view of the system.

- $\mathcal{B} = \{(\theta, \mu) \mid \theta \in \mathcal{F}, \mu \in \mathcal{I}\}$, where $\mathcal{F}$ is the set of discrete external representations of whole numbers. Each tuple represents an element θ in the domain of

f and its image μ under f. A growth function $\gamma : \mathcal{G} \to \mathcal{F}$ (resp. an encoding function $\gamma^{-1} : \mathcal{F} \to \mathcal{G}$) is used to translate from the internal to the external (resp. from the external to the internal) representation.

- $\Omega_{\mathcal{B}} = \{\omega^1_{\mathcal{B}}, \omega^2_{\mathcal{B}}\}$ comprises two operators $\omega^i_{\mathcal{B}} : \mathcal{B} \times \mathcal{A} \to \mathcal{B}$ such that:
  - $\omega^1_{\mathcal{B}}((\theta, \mu), ((\zeta, \xi, T), \langle \iota_1, \cdots, \iota_t \rangle)) = (\theta, \mu)$, i.e., the artifact remains unchanged.
  - $\omega^2_{\mathcal{B}}((\theta, \mu), ((\zeta, \xi, T), \langle \iota_1, \cdots, \iota_t \rangle)) = (\gamma(\xi), \iota_t)$, i.e., the artifact is updated with a new value.
- $\tau_{\mathcal{B}} : \mathcal{I} \times \mathcal{B} \to \Omega$ is defined as

$$\tau(\iota_t, (\theta, \mu)) = \begin{cases} \omega^1_{\mathcal{B}} & \text{if } \iota_t > \mu. \\ \omega^2_{\mathcal{B}} & \text{otherwise.} \end{cases} \tag{2}$$

  i.e., the artifact is updated whenever a better value is found.

The system is initialized with $\mathcal{A}(0) = (\xi, \xi, T_0, \langle\rangle)$ and $\mathcal{B}(0) = (\xi, \mu^*)$, where ξ is a randomly generated value in the domain of f, $T_0 \geq 0$ is the initial threshold value and $\mu^* = \min(\mu \mid \mu \in \mathcal{I})$. With these settings ξ will be evaluated in the first step and $\mathcal{B}(1)$ will be updated with the correct fitness value.

Now, it is possible to define another system with the same external view but using different internal elements (e.g., using binary reflected Gray coding for whole numbers and a different mutation operator η'). These two systems can be subsequently combined by means of sharing an artifact. Such a combination is a weak hybrid model

that allows escaping from local optima in which any of the combined systems would stagnate when isolated [3].

This framework serves as a formalism for characterizing evolutionary algorithms. As an example, it has been used for defining a unified view of cellular and distributed genetic algorithms [4]. Moreover, it allows extracting some conclusions about the computational power of these algorithms in the presence of hybridization. To be precise, it will be shown in the next section that the template of an adaptive system can be instantiated with problem-dependent knowledge so as to allow performing an arbitrary computation (and hence an arbitrary search). Since any computation can be represented as a Turing machine (we consider this model of computation for the simplicity of the further development), it will be shown that an adaptive system can simulate the behavior of these devices.

4. An Adaptive Turing Machine

Before describing how an adaptive system can simulate the behavior of a Turing Machine (TM hereinafter), some basic elements must be previously defined.

Let M=(Σ, Q, s, δ) be a TM, i.e., a tuple comprising a set of symbols Σ, a set of states Q, an initial state s and a transition function δ. Let Q^h be $Q \cup \{h\}$, where h is a special state (the *halt* state). Let $\Sigma^{L,R}$ be $\Sigma \cup \{L, R\}$, where L and R represent possible moves of the TM head (to left and right, respectively).

Now, the transition function $\delta : Q \times \Sigma \to Q^h \times \Sigma^{L,R}$ can be decomposed in two orthogonal families of functions $(\delta_\Sigma^{a_1}, \delta_\Sigma^{a_2}, ..., \delta_\Sigma^{a_m})$ and $(\delta_Q^{q_1}, \delta_Q^{q_2}, ..., \delta_Q^{q_n})$ where $a_i \in \Sigma$ and $q_j \in Q$. Thus, the original transition function δ can be rewritten as $\delta(q, a) = \delta_\Sigma^a(q) = \delta_Q^q(a)$.

Let $\langle\cdot\rangle_k : Q \to \Sigma_\varrho^k$ be a function that encodes a state in exactly k symbols. Σ_ϱ is an alphabet from which these symbols are taken. Conversely, $\langle\cdot\rangle_k^{-1} : \Sigma_\varrho^k \to Q$ is a function that decodes a string of k symbols in the state it represents.

Definition 3 ($\diamond$ operator). The infix $\diamond$ operator is defined as

$$\diamond : (\Sigma \to (Q^h \times \Sigma^{L,R})) \times (\Sigma^* \times \Sigma \times \Sigma^*) \to (\Sigma^* \times \Sigma \times \Sigma^*)$$

with the following semantics:

$$\delta_Q^i \diamond u_1 \underline{a} u_2 = \begin{cases} u_1 \underline{a'} u_2 & \text{if } \pi^2(\delta_Q^i(a)) = a', a' \in \Sigma \\ u_1' \underline{x} a u_2 & \text{if } \pi^2(\delta_Q^i(a)) = L, u_1' x = u_1 \\ u_1 a \underline{x} u_2' & \text{if } \pi^2(\delta_Q^i(a)) = R, x u_2' = u_2 \end{cases} \tag{3}$$

where π^j represents the projection of the jth argument.

This operator can be seen as the application of a transition rule to a string of symbols that represents the contents of the tape. For simplicity, the tape is considered two-way infinite. New blank-symbols # are inserted whenever $u_1 = \varepsilon$ or $u_2 = \varepsilon$.

After having stated the basic components, an adaptive system can be defined to simulate the behavior of any TM. This adaptive system is placed in an environment

E that determines the computation to be done.

Lemma 4 (simulation of a TM). Let M be a TM. There exists an adaptive system $\mathcal{S}$ that simulates M.

Proof of lemma 4: Let M=(Σ, Q, s, δ). Define $(\mathcal{A}, \Omega, \tau, \mathcal{B}, \Omega_{\mathcal{B}}, \tau_{\mathcal{B}}, \mathcal{I})$ such that:

- $\mathcal{A} = \mathcal{A}^1 \times \mathcal{M}$, where $\mathcal{A}^1 = \{\langle q \rangle_k \mid q \in Q^h\}$ and $\mathcal{M}$ is the input story of the system. Thus, $\mathcal{A}^1(t)$ is the internal state of the TM at time t.

- $\Omega = \{\omega^i \mid i \in \Sigma\}$ such that $\omega^i(\mathcal{A}_j) = \langle \pi^1(\delta^i_\Sigma(\langle \mathcal{A}_j \rangle^{-1}_k)) \rangle_k$, i.e., each operator ω^i decodes the internal state, applies a certain transition rule to obtain a new state, and finally encodes it back.

- $\tau : \mathcal{I} \times \mathcal{A} \to \Omega$ is defined as $\tau(\iota(t), \mathcal{A}(t)) = \omega^{\iota(t)}, \forall \mathcal{A}(t) \in \mathcal{A}$.

- $\mathcal{B} = (\Sigma^* \times \Sigma \times \Sigma^*)$. This artifact-space represents the possible configurations of the tape.

- $\Omega_{\mathcal{B}} = \{\omega_{\mathcal{B}}\}$ such that $\omega_{\mathcal{B}}(\mathcal{B}_i, \mathcal{A}_j) = \omega_{\mathcal{B}}(\mathcal{B}_i, (\mathcal{A}^1_j, \langle \iota_i, \cdots, \iota_{tj} \rangle)) = \delta^q_Q \diamond \mathcal{B}_i$, where $q = \langle \varpi^{-1}(\mathcal{A}^1_j) \rangle^{-1}_k$, and $\varpi \equiv \omega^{\iota_{tj}}$. This operator manipulates the external artifact (the tape), updating the cell under the imaginary TM head according to the transition function. For that purpose, it applies a transition rule δ^q_Q to the tape, where q is the previous state (this is required because, for technical reasons, the system first updates its internal structure -i.e., state- and then the artifacts).

- $\tau_{\mathcal{B}} : \mathcal{I} \times \mathcal{B} \to \Omega_{\mathcal{B}}$ is defined as $\tau_{\mathcal{B}}(\iota(t), \mathcal{B}(t)) = \omega_{\mathcal{B}}, \forall \mathcal{B}(t) \in \mathcal{B}, \forall \iota(t) \in \mathcal{I}$.

- $\mathcal{I} \equiv \Sigma$, i.e., the system receives the symbol under the imaginary TM head as an input.

Initially, $\mathcal{A}^1(0) = \langle s \rangle_k$ (i.e., the encoding of the initial state) and $\mathcal{B}(0) = \#u\underline{\#}$ (i.e., the input for the function to be computed). Subsequently, the adaptive plan iteratively modifies the internal structure (state) according to the inputs from the environment (the symbol read), appropriately updating the external artifact (the tape).

This process is repeated until the computation is complete. Notice that, in principle, an adaptive system has no halting state. On the contrary, it keeps carrying its adaptive plan as long as it receives an input from the environment. However, this does not pose any problem for the purposes of the simulation since it can be seen that, if the TM halts at time t, $\mathcal{A}(t+k) = \mathcal{A}(t), \forall k \geq 0$ and $\mathcal{B}(t+k) = \mathcal{B}(t), \forall k \geq 0$. On the other hand, it is easy to extend the artifact space so as to include an indication that the TM has halted. Stimuli from the environment can then be stopped.

Proof of lemma 4 □

5. Evolutionary Algorithms as Nondeterministic TMs

Lemma 4 shows that it is possible to define an algorithm within the framework of adaptive systems capable of performing an arbitrary computation. For that purpose, and as it been shown, it is generally required to use a family of operators to be applied in different stages of the computation. However, traditional evolutionary

algorithms usually use just two operators (crossover and mutation) or even just one (a mutation operator). In this scenario, these algorithms can be seen as a kind of nondeterministic TM (NTM hereinafter). To see this, consider the following example:

Example 5.

Let $f : \mathcal{D} \to \Re$ be a fitness function ranging over a finite set $\mathcal{D}$. Let $\mathcal{S}$ be an evolutionary algorithm. For the sake of simplicity, it is assumed that $\mathcal{S}$ only uses a stochastic mutation operator η. Now consider the following adaptive system (only the internal view is described):

- $\mathcal{A} = \mathcal{A}^1 \times \mathcal{M}$, where $\mathcal{A}^1 = \{\langle(\zeta_1, \mu_1), \cdots, (\zeta_k, \mu_k)\rangle \mid k \leq |\mathcal{G}|,\ (\zeta_i, \mu_i) \in (\mathcal{G} \times \mathcal{R}),\ \zeta_i \neq \zeta_j,\ 1 \leq i, j \leq k\}$, i.e., $\mathcal{A}^1$ comprises all sequences of tuples (ζ_i, μ_i) without any ζ_i duplicated. The first element of each tuple is the internal representation of a value in $\mathcal{D}$ and the second element is the internal representation of its image under f. As to $\mathcal{M}$, it is the input story of the system as in Example 2.

  Initially, $\mathcal{A}(0) = (\langle(\zeta^*, \mu^*)\rangle, \langle\rangle),\ \zeta^* \in \mathcal{G},\ \mu^* \in \mathcal{R}$.

- $\mathcal{I} = \{\langle\mu_1, \cdots, \mu_k\rangle \mid k \leq |\mathcal{G}|,\ \mu_i \in \mathcal{R},\ 1 \leq i \leq k\}$, i.e., the inputs from the environment are sequences of fitness values.

- $\Omega = \{\omega\}$, such that $\omega(\mathcal{A}_j) = \omega(\mathcal{A}^1_j, \mathcal{M}_j) = (\omega_\eta(\omega_{up}(\mathcal{A}^1_j, \mathcal{M}_j)), \mathcal{M}_j)$, where the functions ω_{up} and ω_η are defined as follows:

– $\omega_{up} : \mathcal{A}^1 \times \mathcal{M} \to \mathcal{A}^1$ is recursively defined as:

$$\omega_{up}(\langle(\zeta_i, \mu_i), \cdots, (\zeta_k, \mu_k)\rangle, \langle \iota_1, \cdots, \iota_t\rangle) =$$

$$\begin{cases} \langle(\zeta_i, \mu')\rangle :: \omega_{up}(\langle(\zeta_{i+1}, \mu_{i+1}), \cdots, (\zeta_k, \mu_k)\rangle, \langle \iota_1, \cdots, \iota'_t\rangle) & \text{if } \iota_t = \langle \mu' \rangle :: \iota'_t. \\ \langle(\zeta_i, \mu_i), \cdots, (\zeta_k, \mu_k)\rangle & \text{if } \iota_t = \langle\rangle. \end{cases}$$

where :: represents a sequence concatenation operation. This operator simply updates the fitness values of the elements that were created in the previous step, using the values provided by the environment.

– $\omega_\eta : \mathcal{A}^1 \to \mathcal{A}^1$ is defined as $\omega_\eta(\langle(\zeta_1, \mu_1), \cdots, (\zeta_k, \mu_k)\rangle) = \langle(\eta(\zeta_i), \mu^*) \mid \eta(\zeta_i) \neq \zeta_j, 1 \leq i, j \leq k, \mu^* \in \mathcal{R}\rangle :: \langle(\zeta_1, \mu_1), \cdots, (\zeta_k, \mu_k)\rangle$.

This function generates new non-duplicated η-neighbors of the elements in the current population.

- $\tau : \mathcal{I} \times \mathcal{A} \to \Omega$ is defined as $\tau(\iota(t), \mathcal{A}(t)) = \omega, \forall \iota(t) \in \mathcal{I}, \forall \mathcal{A}(t) \in \mathcal{A}$, i.e., selects the only available operator.

This adaptive system can be clearly identified as a NTM. Its internal structure comprises a population of substructures (individuals). In each step of the adaptive plan, these individuals are evaluated and new individuals are created and inserted in the population in parallel. Hence, only a polynomial number of adaptive steps is required to find the optimal value of f (as long as only a polynomial number of η-mutations be required to reach this optimal value from $\zeta^* \in \mathcal{G}$). The clear drawback is the size of the population, which will exponentially grow.

Obviously, the system defined in Example 5 is unrealistic due to the generally exponential growth of the population. Nevertheless, this system serves as a previous step for defining a more adjusted model of evolutionary algorithms. To be precise, the following amendments must be made:

- $\mathcal{A}^1 \equiv (\mathcal{G} \times \mathcal{R})^n$, i.e., the population is composed of exactly n pairs (ζ, μ) instead of being any finite sequence of such pairs. Also, $\mathcal{M} \equiv \mathcal{R}^\lambda$, $\lambda \leq n$, i.e., the fitness values of a subset of individual in the population.

  Initially, $\mathcal{A}(0)$ is randomly chosen as $(\langle(\zeta_1^*, \mu_1^*), \cdots, (\zeta_n^*, \mu_n^*)\rangle, \langle\rangle), \zeta_i^* \in \mathcal{G}, \mu_i^* \in \mathcal{R}, 1 \leq i \leq n$.

- $\Omega = \{\omega\}$, with ω and ω_{up} being defined as above. However ω_η is now defined as $\omega_\eta(\langle(\zeta_1, \mu_1), \cdots, (\zeta_n, \mu_n)\rangle) = \langle(\xi_1, \mu^*), \cdots, (\xi_\lambda, \mu^*) \mid \exists \zeta_j \in P, \eta(\zeta_j) = \xi_i\rangle :: P$, where $\mu^* \in \mathcal{R}$ and $P = \omega_\varrho(\langle(\zeta_1, \mu_1), \cdots, (\zeta_n, \mu_n)\rangle)$.

  As it can be seen, a new internal function ω_ϱ is used. This is a function defined as $\omega_\varrho : (\mathcal{G} \times \mathcal{R})^n \to (\mathcal{G} \times \mathcal{R})^{n-\lambda}$ commonly known as *replacement* function. Its purpose is to make room in the population for the individuals that will be created in each step. Subsequently, the new individuals are created by applying η to individuals in this reduced population P.

  Although not specified here, it is also common to consider another internal function $\omega_\sigma : (\mathcal{G} \times \mathcal{R})^{n-\lambda} \to (\mathcal{G} \times \mathcal{R})^\lambda$ known as *selection* function. Thus, the

new individuals are created from $\omega_\sigma(P)$ instead of from P. For simplicity, it can be assumed that $\omega_\sigma(P)$ is a uniform random sampling of P.

With the above modifications, the adaptive system resembles some commonly used models of evolutionary computation such as evolution strategies or evolutionary programming. As it can be seen, one of the most important modifications is the use of the replacement function ω_ϱ in order to keep the population with constant size. Traditionally, this function can be found in two modalities:

a) *Plus* strategy: Let $\{i_1, i_2, \cdots, \imath_n\}$ be an index list of the individuals in the population sorted by increasing fitness values (minimization is assumed), i.e, (ζ_{i_1}, μ_{i_1}) is the best individual in the population and (ζ_{i_n}, μ_{i_n}) is the worst. Then, $\omega_\varrho(\langle(\zeta_1, \mu_1), \cdots, (\zeta_n, \mu_n)\rangle) = \langle(\zeta_{i_1}, \mu_{i_1}), \cdots, (\zeta_{i_m}, \mu_{i_m})\rangle$ where $m = n - \lambda$, i.e., the best m individuals of the population are kept.

b) *Comma* strategy: Let $\{i_1, i_2, \cdots, \imath_\lambda\}$ be an index list of the first λ individuals in the population sorted by increasing fitness values, i.e, a sorted list of the individuals that were created in the last step. Then, $\omega_\varrho(\langle(\zeta_1, \mu_1), \cdots, (\zeta_n, \mu_n)\rangle) = \langle(\zeta_{i_1}, \mu_{i_1}), \cdots, (\zeta_{i_m}, \mu_{i_m})\rangle$ where $m \leq \lambda$, i.e., the best newly-created m individuals are kept.

With any of these two replacement strategies, the system can be generally seen as a restricted NTM in which not every transition (i.e., η-mutation) is explored and some configurations (i.e., points in $\mathcal{D}$) get lost. In this scenario, the obvious goal is

to choose both η and ω_ϱ so as to maximize the probability of reaching the optimal solution starting from a random population. Unfortunately, finding this optimal solution is not always possible. In many situations the system reaches an internal state characterized by either of the two following scenarios:

a) $\omega_\varrho(\mathcal{A}^1(t)) = \omega_\varrho(\langle(\zeta_1,\mu_1),\cdots,(\zeta_n,\mu_n)\rangle) = \langle(\zeta_{\lambda+1},\mu_{\lambda+1}),\cdots,(\zeta_n,\mu_n)\rangle,\ \forall\zeta_i = \eta(\zeta_j),\ 1 \leq i \leq \lambda < j \leq n$. This situation may take place when using a plus strategy and implies that the whole population is at a local optimum within the combined landscapes [18] of the η and ω_ϱ operators. No new individual will be accepted in the population beyond this point.

b) $\omega_\varrho(\mathcal{A}^1(t)) = \omega_\varrho(\langle(\zeta_1,\mu_1),\cdots,(\zeta_n,\mu_n)\rangle) \subseteq \cup_{k<t}\mathcal{A}^1(k),\ \forall\zeta_i = \eta(\zeta_j),\ 1 \leq i \leq \lambda < j \leq n$. This situation may take place when using a comma strategy and implies that the whole population will oscillate across a closed subset of the search space. As in the previous situation, no new individual (i.e., an individual that has not been generated in any previous step) will be accepted in the population beyond this point.

In any of these two scenarios, it is convenient to stop the execution of the adaptive system since no improvement can be expected. A probabilistic analysis for this purpose is presented below, given that a plus strategy is used.

Example 6.

Consider an adaptive system as described above. More precisely, assume that

- $\mathcal{G} \equiv \Sigma^{\ell}$, i.e., the system manipulates strings of ℓ symbols from an alphabet Σ.

- η is a stochastic mutation operator that randomly modifies one of the symbols in a string. More strictly, η can be described as a function $\eta : \mathcal{G} \times \mathcal{G} \to [0, 1]$ defining a probability distribution over all pairs of strings. In this case,

$$\eta(\zeta, \zeta') = \eta(s_1 \cdots s_\ell, s'_1 \cdots s'_\ell) = \begin{cases} \frac{1}{\ell \cdot |\Sigma - 1|} & \text{if } \exists! i, 1 \leq i \leq \ell : s_i \neq s'_i. \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

  i.e., all mutations have the same probability.

Initially, suppose $\lambda = 1$. Hence, the number of substructures in $\mathcal{A}$ is $n = m + 1$. According to the definition of the ω_ϱ operator for a plus strategy:

$$\mathcal{A}(t+1) \neq \mathcal{A}(t) \Longrightarrow \exists j, 2 \leq j \leq m+1 : \mu_1 \succ \mu_j \tag{5}$$

i.e., the newly created string is accepted only if it is better than a preexisting string. Then, the probability of not producing any better string after r steps of the adaptive system is

$$p(E_r) = \sum_{i=0}^{m \cdot (|\Sigma|-1) \cdot \ell} p(E_r / R_i^m) = \tag{6}$$

$$\sum_{i=0}^{m \cdot (|\Sigma|-1) \cdot \ell} \sum_{j=1}^{m} p_s(\zeta_j) \cdot \sum_{k=0}^{\min(i, (|\Sigma|-1) \cdot \ell)} p(R_{k,j}^m / R_i^m) \cdot \left(\frac{(|\Sigma| - 1) \cdot \ell - k}{(|\Sigma| - 1) \cdot \ell} \right)^r \tag{7}$$

where the following notation is used:

$p(E_r)$: probability of not producing any better string in r steps of the system.

$p_s(\zeta_j)$: probability of selecting the jth individual, e.g., $\frac{1}{m}$ assuming random selection.

$p(R_i^m)$: probability that a total number of i mutations produce a better string in a population of m individuals.

$p(R_{k,j}^m)$: probability that k mutations on ζ_j produce a better string.

Subsequently, and since $p(E_r/R_0^m) = 1$, it is easy to see that $p(R_0^m/E_r) = p(R_0^m)/p(E_r)$. The key point in (6) is the calculation of $p(R_i^m)$. For that purpose, the easiest option is to assume a uniform distribution over all target functions and populations (in the line of the assumptions of the No Free Lunch theorem). Thus,

$$p(R_i^m) = \frac{\begin{pmatrix} m\cdot(|\Sigma|-1)\cdot\ell \\ i \end{pmatrix}}{\sum_{j=0}^{m\cdot(|\Sigma|-1)\cdot\ell}\begin{pmatrix} m\cdot(|\Sigma|-1)\cdot\ell \\ j \end{pmatrix}} = \frac{\begin{pmatrix} m\cdot(|\Sigma|-1)\cdot\ell \\ i \end{pmatrix}}{2^{n_c\cdot(|\Sigma|-1)\cdot\ell}} \tag{8}$$

Then, $p(R_{k,j}^m/R_i^m)$ can be calculated as

$$p(R_{k,j}^m/R_i^m) = \frac{p(R_{k,j}^m \cap R_i^m)}{p(R_i^m)} = \frac{\begin{pmatrix} (|\Sigma|-1)\cdot\ell \\ k \end{pmatrix}\cdot\begin{pmatrix} (m-1)\cdot(|\Sigma|-1)\cdot\ell \\ i-k \end{pmatrix}}{\begin{pmatrix} m\cdot(|\Sigma|-1)\cdot\ell \\ i \end{pmatrix}} \tag{9}$$

Finally, notice that the more general situation in which $\lambda > 1$ and no improvement has been observed in the last r steps is equivalent to use $\lambda' = 1$ and $r' = \lambda \cdot r$.

As an example of the application of this estimation, consider the Schwefel [24] function, a nonlinear minimization problem whose functional form is as follows:

$$F(\vec{x}) = nV + \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|}) \qquad x \in [-512, 511] \tag{10}$$

where $V = -\min\{-x \sin(\sqrt{|x|}) : x \in [-512, 511]\}$. Now, consider a system using an integer representation in which $\lambda = 10$, $m = 1$ and η is a mutation operator that increases or decreases by 1 a randomly selected vector position (performing wrapping if necessary).

Figure 3 near here

Fig. 3 (left) shows an experimental example on a 10-dimensional Schwefel function. It can be seen that the system enters a stationary state in the interval between 4500 and 5000 function evaluations. This is accurately detected with the above estimation as shown in Fig. 3 (right).

In many situations, the above estimation may be very conservative. Some strategies for coming up to a better estimation of $p(R_i^m)$ using knowledge of the target function are described in [3]

6. Concluding Remarks

This work fits in the context of an old debate about the nature of evolutionary algorithms. Traditionally, they have been considered as function optimizers. They can be certainly used for that purpose (specifically evolution strategies and evolutionary programming, whose asymptotic convergence to the optimum is guaranteed). How-

ever, they are a more general tool for the resolution of sequential decision problems [10, 11]. In fact, the template of an evolutionary algorithm allows the definition of a computational model with Turing capabilities, as demonstrated in section 4.

In the light of these results, characterizations of evolutionary algorithms in which their applicability is restricted to certain domains have to be revisited. As a matter of fact, such a limitation arises from implicit assumptions about the functioning of the evolutionary operators. However, nothing precludes incorporating as much problem-knowledge as needed in these operators, as it has been empirically shown in the last years. Of course, there still remains the problem of adequately incorporating this knowledge. This step is inherently hard as shown by [16]. This implies that finding the optimal hybrid algorithm, that is, determining the best representation and operators (even in an approximate sense), can be as hard as solving the target problem. Hence, only heuristic guidelines can be used for this purpose (see [9, 20, 5, 6]).

In this sense, and as the TM metaphor suggests, an operator pool comprising several reproductive operators may be required since different structures may request a different treatment. A decision apparatus (analogous to the transition function) must decide how to make each individual solution evolve. This approach is usually followed in Evolutionary Programming (which seems to be an appropriate technique for problems with high interdependencies [13]) and has been also proposed by Davis [9] in the field of Genetic Algorithms. Consider that a TM may constitute an extreme

case in which a different reproductive operator (i.e., transition rule) can be defined for each configuration. Obviously, not every problem require this.

References

[1] T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.

[2] C. Cotta. On resampling in nature-inspired heuristics. In V. Botti, editor, *Proceedings of the Seventh Conference of the Spanish Association for Artificial Intelligence*, pages 145–154, 1997. In Spanish.

[3] C. Cotta. *A Study of Hybridization Techniques and their Application to the Design of Evolutionary Algorithms.* PhD thesis, University of Málaga, 1998. In Spanish.

[4] C. Cotta, E. Alba, and J.M. Troya. A study of the power and robustness of parallel evolutionary algorithms. *Inteligencia Artificial*, 5:6–13, 1998. In Spanish.

[5] C. Cotta and J.M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6 (1):25–44, 1998.

[6] C. Cotta and J.M. Troya. On decision-making in strong hybrid evolutionary algorithms. In A.P. del Pobil, J. Mira, and M. Ali, editors, *Tasks and Methods in Applied Artificial Intelligence*, volume 1415 of *Lecture Notes in Computer Science*, pages 418–427. Springer-Verlag, Berlin, 1998.

[7] J.M. Daida, S.J. Ross, and B.C. Hannan. Biological symbiosis as a metaphor for computational hybridization. In L.J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 328–335, San Francisco CA, 1995. Morgan Kaufmann.

[8] R. Das and D. Whitley. The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 166–173, San Mateo CA, 1991. Morgan Kauffman.

[9] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold Computer Library, New York, 1991.

[10] K.A. De Jong. Are genetic algorithms function optimizers? In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature 2*, pages 3–13, Amsterdam, 1992. Elsevier Science Publishers B.V.

[11] K.A. De Jong. Genetic algorithms are not function optimizers. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5–17, San Mateo CA, 1993. Morgan Kauffman.

[12] G. Ducek and T. Scheuer. Threshold accepting: a general purpose optimization algorithm. *Journal of Computational Physics*, 90:161–175, 1990.

[13] D.B. Fogel. Evolutionary programming and evolution strategies tutorial. Stanford, CA, 1996. Tutorial at the 1996 Genetic Programming Conference.

[14] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution.* Wiley, New York, 1966.

[15] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading MA, 1989.

[16] W.E. Hart and R.K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195, San Mateo CA, 1991. Morgan Kaufmann.

[17] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Harbor, 1975.

[18] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, 1995.

[19] N.J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205, 1991.

[20] N.J. Radcliffe. Fitness variance of formae and performance prediction. In L.D. Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72, San Mateo CA, 1994. Morgan Kauffman.

[21] N.J. Radcliffe and P.D. Surry. Fundamental limitations of search algorithms: Evolutionary computing in perspective. In J. Van Leeuwen, editor, *Computer*

Science Today: Recent Trends and Developments, volume 1000 of *Lecture Notes in Computer Science*, pages 275–291. Springer-Verlag, 1995.

[22] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Verlag, Stuttgart, 1973.

[23] M.D. Vose and G.E. Liepins. Schema disruption. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–243, San Mateo CA, 1991. Morgan Kauffman.

[24] D. Whitley, S. Gordon, and K. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Y. Davidor, H. Schwefel, and R. Männer, editors, *Parallel Problem Solving From Nature 3*, pages 6–15. Springer-Verlag, 1994.

[25] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1 (1):67–82, 1997.

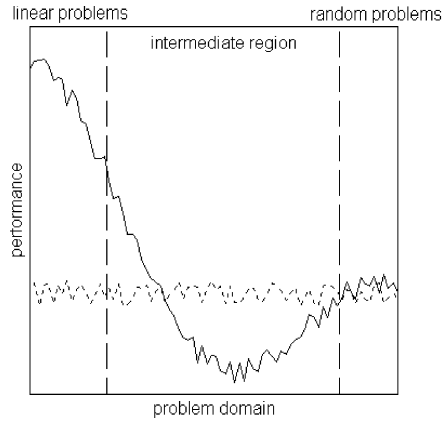

Figure 1: Performance comparison of two algorithms $\mathcal{A}$ (solid line) and $\mathcal{B}$ (dotted line). They exhibit the same performance in the random region and $\mathcal{A}$ is better in the linear region. Hence, there exist problems in the intermediate region in which $\mathcal{B}$ must outperform $\mathcal{A}$.

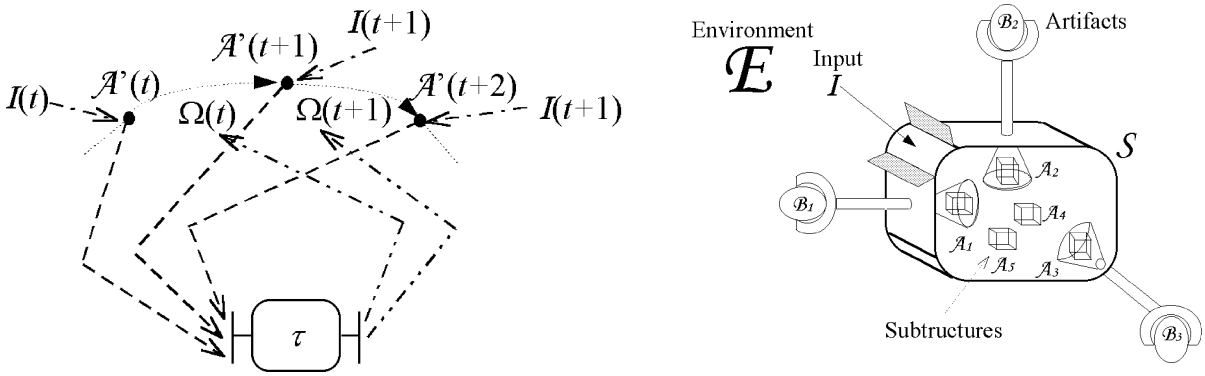

Figure 2: Outline of the functioning of the adaptive plan (left) and imaginary aspect of an adaptive system $\mathcal{S}$ (right).

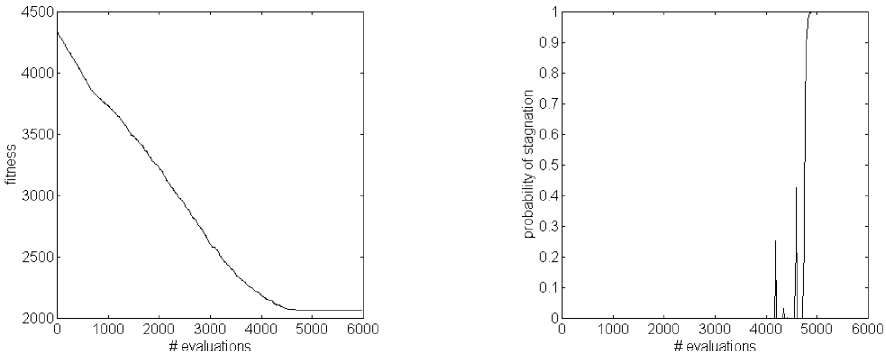

Figure 3: Evolution of fitness in an experimental run of a (1+10) algorithm on a 10-dimensional Schwefel function (left) and evolution of the stagnation probability in the same run (right).