The secondary substrate problem in Co-Evolution and Developmental-Evolution

A Dissertation

Presented to
The Faculty of the Graduate School of Arts and Sciences
Brandeis University
Michtom School of Computer Science
Jordan B. Pollack, Advisor

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Shivakumar Viswanathan
May, 2007

This dissertation, directed and approved by Shivakumar Viswanathan's committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of:

DOCTOR OF PHILOSOPHY

Adam B. Jaffe, Dean of Arts and Sciences

Dissertation Committee:

Jordan B. Pollack, Chair

Timothy J. Hickey

Pengyu Hong

Una-May O'Reilly

Acknowledgments

The vigorous intellectual climate and camaraderie provided by the members of the Dynamical and Evolutionary Machine Organization (DEMO) lab formed the bedrock of this whole endeavor. My interest in developmental computation was a direct outcome of the exemplary work of Pablo Funes, Hod Lipson, Gregory Hornby and John Rieffel on evolutionary design. Sevan Ficici, Anthony Bucci, Edwin De Jong and Ari Bader-Natal educated me in the ways of coevolution; and Richard Watson and Keki Burjorjee taught me about the importance and subtleties of the representation problem in Evolutionary Computation. In an atmosphere filled with evolution-speak, Simon Levy, Ofer Melnik and Paul Chiusano provided a valuable perspective on neural networks and cognition. My deepest gratitude goes to these members of the DEMO lab. I am particularly indebted to Sevan Ficici, Ari Bader-Natal, Keki Burjorjee, Anthony Bucci, Richard Watson and Paul Chiusano for their critical and constructive feedback, extensive discussions, guidance and friendship.

This experience at the DEMO lab would have been impossible without Jordan Pollack. The tenacity of his quest for a non-cognitive solution to the AI problem; and his deep insights into the hard problems at the core of the AI, EC and ALife research programs, often described humorously, have been a great inspiration and have significantly shaped my research perspective. The technical formulation of developmental computation as a game is attributable to a key insight suggested by Jordan about the relevance of the Bellman equations to my attempts to formulate a computation model for developmental robustness.

A special thanks to Timothy Hickey, Pengyu Hong and Una-May O'Reilly for agreeing to be on my committee and for their patience with the numerous fits and starts leading up to the defense. I would in particular like to thank Una-May for her detailed feedback and for suggesting a lead that resulted in significant improvements to the mathematical formulation of the developmental model described here.

I am indebted to Myrna Fox, Ashley Boudo, Ruth Brigham, Jeanne DeBaie, Scott Buchanan, Saul Tejada and Julio Santana for their support and friendship; and Jacques Cohen and Robert Sekuler for their mentorship. I would especially like to thank Myrna Fox for her maternal concern, and for always saving the day in my numerous bureaucratic bumbles. I am deeply grateful to Judit Jané-Valbuena, Antonella Di Lillo, Teresa Broering, Phil Durbin, Radhika Subramanian, Chrisann Newransky, Helen and Stephan Kissler, Emily Lincoln, Matthew Tanner, Sierra Rothberg, Adam Beleveau and Aftab Pande for their irreplaceable friendship.

This dissertation may never have reached completion if it werent for the love and encouragement unmeasurable from my parents; sisters Sandya and Soumya; brothers-in-law Murali and Murari; my aunt Malathi Thiagarajan; and Peter and Doreen Mudry. And last and far from being the least, it is difficult to fully express my thanks to Kartik Chandran, Brenda Mudry and Amartya, for integrating me so thoroughly and completely into their lives through the ups and downs of these years in graduate school.

Abstract

The secondary substrate problem in Co-Evolution and Developmental-Evolution

A dissertation presented to the Faculty of
the Graduate School of Arts and Sciences of
Brandeis University, Waltham, Massachusetts

by Shivakumar Viswanathan

The performance of an Evolutionary Algorithm on a search problem is critically effected by the substrate used to encode the candidate solutions of the problem. In addition to the challenge of designing evolvable genetic substrates, *two-population competitive coevolutionary algorithms* (coEAs) and *developmental Evolutionary Algorithms* (devo-EAs) present another substrate-related design problem. Both involve an additional substrate with its own mechanism of change. In coEAs, test-cases are encoded with an independent genetic substrate having its own variation operators. In devo-EAs, phenotypes are composed of a distinct substrate with associated generative mechanisms capable of changing an individual's form and size during development. Though this "secondary" substrate is a distinctive feature of both algorithms, the design problem it poses remains poorly understood.

This dissertation proposes novel formal models to characterize how the properties of the secondary substrate influences the performance devo-EAs and coEAs respectively.

Firstly, we propose a computational model for devo-EAs which shows that the point in time at which the development of a phenotype halts can introduce selection biases that can cause an empirically measurable retardation in the performance of a devo-EA. Furthermore, a Genotype-Phenotype map that is bias-free is formally equivalent to a Nash equilibrium in a non-cooperative multi-player game, where each genotype is a player, the possible halting points are strategies and the payoffs are related to the fitness function. We show that algorithmic solutions to find this Nash map are expensive without a suitable secondary substrate.

Secondly, we propose a novel search space model for Pareto coevolution that formally defines the evolvability properties required of the secondary substrate for pathology-free learning with a mutation-only coEA. With this model, we show that on boolean classification problems (a) the variational properties of the secondary substrate are a property of the problem class rather than tied to individual problems, and (b) the absence of coevolutionary pathologies does not imply success in finding high-quality solutions. Rather than being mysterious dynamical properties of coEAs, these findings are transparently explained using Machine Learning first principles.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The genetic theory of evolution, as formulated by the Modern Synthesis, applies to evolutionary adaptation occurring in all biological organisms. This generality has been an important inspiration for the Evolutionary Algorithm (EA) where the genetic logic of evolution is interpreted as a computational mechanism to "evolve" solutions to search problems irrespective of whether the entities represented by the "genomes" are computer programs, robot designs, neural networks or any other kind of systematically representable object [36, 26, 47, 7].

Despite the generality of EAs, a widely noted property of these algorithms is that their performance on a particular search problem is critically linked to the specific genetic data-structure used to represent the candidate solutions of the problem. The design problem posed by choosing a suitable genetic representation that enables effective evolution has been widely referred to as the *representation problem* [2, 75]. However, two popularly used biologically-inspired variants of the standard EA bring a novel added dimension to this classic representation problem in having an additional substrate as an integral part of their operation. These two algorithms are (a) the "Host-parasite" based *competitive co-evolutionary algorithms* (coEAs) [34, 61, 67, 16], and (b) *developmental-EAs* (devo-EAs) [45, 33, 67]. The additional substrate in coEAs is the genetic data-structure used to encode the "parasites" or tests; and in devo-EAs, it is the data-structure used to encode the "embryo".

The novel dimension presented by this additional substrate is that it is subject to operations of change that are *independent* of the variation operations acting on the genetic representations of the candidate solutions. In coEAs, the genetic encoding of the "parasites" is subject to independent mutation and crossover operations. The operations of change in devo-EAs are of a different flavor. Biological development is a process of change par excellence with the entity undergoing change being the "embryo" that goes from a single-celled entity to one consisting of millions of differentiated cells. Though not of the same order as biological development, this extensive change is a property of the artificial development phase in devo-EAs as well, where the embryonic substrate is subject to extensive modification during the developmental construction of the phenotype of each individual in every generation of the evolutionary process. For clarity, we will refer to this additional substrate in both coEAs and devo-EAs as the *secondary substrate* due to its auxiliary role with respect to the genetic representation of the candidate solutions (henceforth referred to as the *primary substrate*).

Given the integral role of this additional substrate in the problem-solving strategy of both coEAs and devo-EAs, it raises a basic question - *how do the properties of this secondary substrate and its associated operations of change influence the performance of (a) coEAs and (b) devo-EAs*? Despite the large body of research on both coEAs and devo-EAs, this critical concern remains poorly understood and has so far not been subject to focussed technical investigations.

The principled characterization of this *secondary substrate problem* posed by coEAs and devo-EAs is the subject of this dissertation. In this chapter, we provide a brief introduction to coEAs and devo-EAs, and an overview of the strategy adopted to study their respective secondary substrate problems.

1.1 Background: The Evolutionary Algorithm

The typical context for the application of an EA arises when we seek an object that has a certain desired behavioral property, say Z, but where a computational method to conveniently and efficiently construct such an object directly based on the specification of Z is unavailable. *Search* is a general framing of such a problem as one of identifying such an object from a set of potential candidate solutions $\mathcal{S}$.

In the context of the evolutionary metaphor, the "fitness" is a measure defined on $\mathcal{S}$ that quantifies the extent to which a member of $\mathcal{S}$ has the property Z. In seeking to apply an Evolutionary Algorithms (EA) to a problem the only requirement that is usually made of such a fitness measure is that it is internally consistent. A convenient mathematical representation of such an internally consistent measure of relative suitability is as a function $f : \mathcal{S} \rightarrow \mathbb{R}$, where f, known as the *fitness function*, assigns a numerical value to every element of $\mathcal{S}$ expressing the magnitude of the element's suitability. The goal in using an EA is to find an object in $\mathcal{S}$ that maximizes this value given a particular fitness function.

The canonical form of the Darwinian Evolutionary Algorithm applied to such a search problem can be described as follows:

1. Draw an initial "population" of candidate solutions from $\mathcal{S}$

2. FITNESS EVALUATION: Evaluate all the individuals in the population using f

   - DEVELOPMENT: This stage is specific to methods where the data-structure encoding the candidate solutions is different from that which can be evaluated by f. This consequently involves an additional phase where the "evaluatable" versions of the entities are produced. This is akin to the genome/phenome distinction in biology, where genomes (encoded in DNA) are transmitted from the parents as part of a single-celled proto-offspring and the production of the adult phenome of the offspring involves a developmental phase.

3. STOPPING CONDITION: If some pre-chosen stopping condition is met then return the

highest fitness individual in the population and halt, else continue.

4. GENERATE A NEW POPULATION

   (a) SELECTION: Probabilistically pick individuals from the population to be "parents" (with replacement). In keeping with the Darwinian heuristic of "survival of the fittest", the probability of an individual A in the population being selected is greater than that of individual B iff $f(A) > f(B)$.

   (b) REPLICATION AND VARIATION: Generate "offspring" by making copies of the representations of the "parents" and apply *uninformed* variation operators to these copies. The variation is uninformed in the sense that the fitness or specific behavioral properties of the parents does not influence the specific manner in which variation operators are applied to the parents' representations. This is similar to the notion of random genetic mutations and crossover that occurs in biological evolution.

   (c) REPLACEMENT: Replace all/some of the individuals in the current population with these "offspring" to obtain the new population.

5. Goto step (2)

A schematic of the typical organization of these algorithmic processes occurring between two consecutive generations is shown in Figure 1.1. As shown in this figure, the genotypes are transmitted from one generation to another while the phenotypes are constructed repeatedly in every generation. In many cases the data-structure representing the phenotype and genotype are identical and the genotype/phenotype distinction serves as a way to distinguish between the two distinct process that the entities in the population are part of, namely (a) fitness evaluation operations (on the phenotype) and (b) selective replication and variation (on the genotype). The fitness of phenotypes is evaluated in every generation and no explicit fitness information is transmitted between generations.

While this extreme generality of EAs has led to their application to a wide variety of problems, a major challenge in applying EAs to specific real-world problems is dependent on

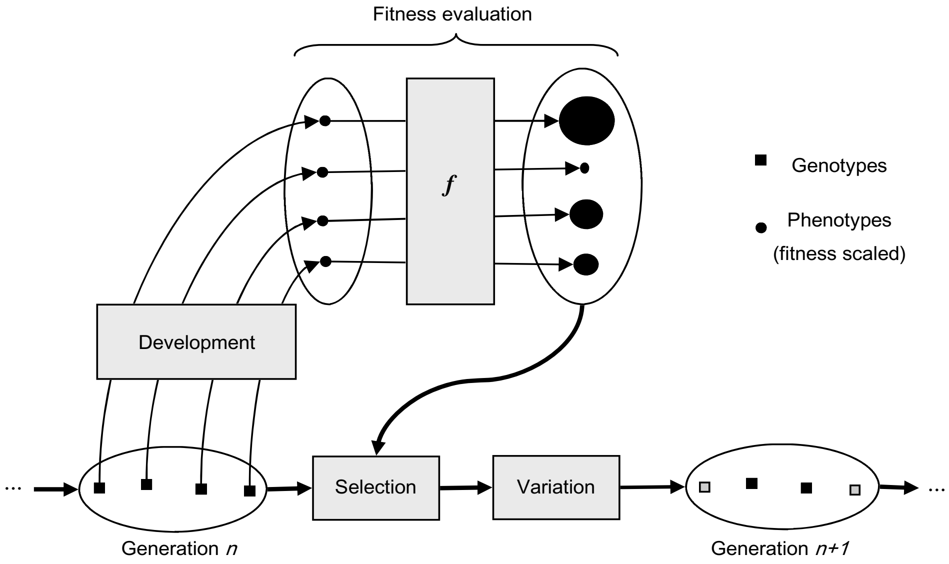


Figure 1.1: Schematic of the algorithmic processes occurring between generations

the ability of the engineer to (a) operationalize the evaluation of the desired property Z as a principled yet computationally practical fitness function, and (b) to pick a representation appropriate for the problem domain. Depending on the behavioral property of interest and the problem domain, these design issues can present non-trivial difficulties requiring significant domain knowledge, judgment and ingenuity on the part of the engineer.

A question both of theoretical and practical interest is whether the Evolutionary Computation framework could be extended in some way to mitigate these domain-specific difficulties while retaining (to the extent possible) the spirit of generality of the EA. Two innovative biologically inspired extensions that have been proposed to this end have been *competitive coevolution* [34, 67, 16, 62] to address issue (a) above, and *developmental representations*[45, 33, 67, 37, 11] to address issue (b). The innovation in both cases has been to frame fitness evaluation and the structuring of the representation as computational "problems" that could be addressed by mechanisms naturally associated with biological evolution.

1.2 Coevolution and development

Both *competitive coevolution* and *development* are independent areas of active research and each has come to be associated with a large and diverse family of techniques. Unlike the canonical EA, however, a precise definition of the structure of these algorithms is a slippery issue in both cases. This concern will feature prominently in our subsequent analysis so, for the time being, the classic form of these algorithms is described below.

1.2.1 Competitive coevolution

Two-population competitive coevolution [34, 67, 16, 62] is an evolutionary problem-solving approach distinguished by the feature that the fitness evaluation algorithm is itself another Evolutionary Algorithm. This fitness evaluation strategy has found a natural application to the automated generation of programs for complex tasks [34, 48, 67, 22, 69] where the behavior of the desired solution is defined in terms of its behavior over a number of discrete "tests". This includes game learning [62, 53], classification [56, 39, 42, 24], and the design of robotic controllers [25, 67].

In these test-based problems the number of such tests can be large, often of the order of the size of the candidate solution space itself. The sheer number of tests makes the naive strategy of exhaustively evaluating each candidate solution in the population against all the tests highly impractical. Typically, addressing this situation can require significant domain knowledge to identify a suitable subset of tests to be used as a training set. To address this difficulties, in a coevolutionary approach the problem of picking a parsimonious subset of suitable tests is posed as a search problem. The mechanism of choice to address this search problem is an EA. Rather than being a search problem that is addressed offline independent of the main evolutionary algorithm, this second EA operates over the space of test-cases with the goal being to adaptively find subsets of tests to evaluate the main evolving population of candidate solutions *at run time*.

The qualifier "competitive" comes from the natural phenomenon inspiring this approach,

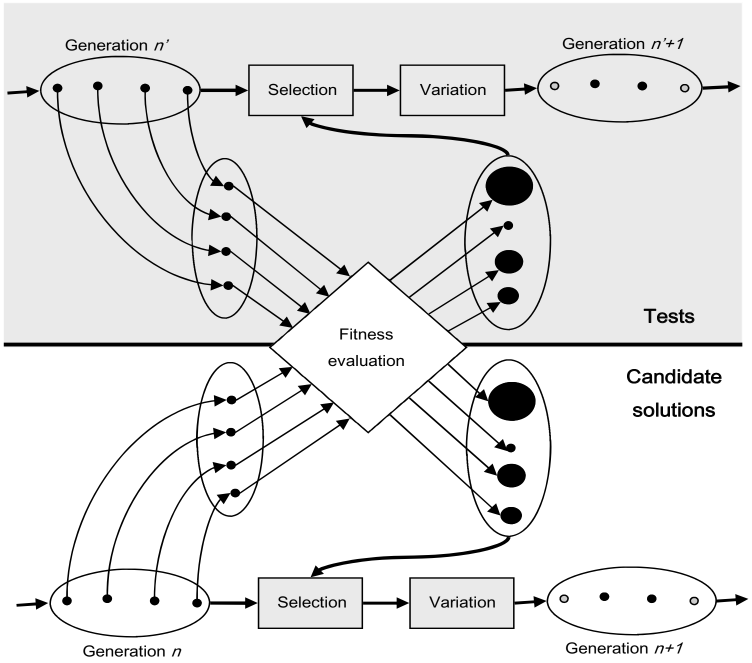


Figure 1.2: Schematic of the algorithmic processes occurring between generations

namely, the "arms races" that are known to occur between reciprocally evolving (i.e. coevolving) biological species that occupy the same ecological niche and have an adversarial or "competitive" relationship with each other [18, 71]. An example is the evolutionary relationship between a virulent species of parasite and its host species. The parasites are under constant selection pressure to adapt to beat the host's immune system, and the hosts are concurrently under selection pressure to develop defenses to ward off the parasites.

In the canonical algorithmic interpretation of such a competitive coevolutionary relationship, the protocol and associated logic of assigning fitness to the individuals in each population is explicitly designed to induce such a reciprocal competition where an individual in each population receives a fitness proportional to its ability to "beat" the current individuals in the other population. For brevity, we will refer to this pair of concurrently operating EAs as being a *coEA*. A schematic of the general dynamic of a canonical coEA is shown in Figure 1.2.

A variant of this scheme is *single-population* coevolution [6, 58, 15, 3] where there is no independent test population. Instead each individual is evaluated by using all the other individuals in the same population as tests. This is applicable only in a limited number of domains such as games where every player can directly interact with another player. This can be considered to be an evolutionary interpretation of learning by self-play [64, 70, 79]. In more biological terms, this is also known as frequency dependent selection and formed the basis for the seminal work of Maynard Smith on Evolutionary Game Theory[52]. An alternate coevolutionary paradigm where this reciprocal relationship is interpreted in explicitly non-adversarial terms is "cooperative" coevolution [59]. While these are important alternatives among others, our focus will be restricted to the two-population competitive case.

1.2.2 Development

A challenging search problem in automated design is the scenario where it is expected that suitable solutions would likely be composed of a large number of elements in complicated configurations but where the general form of these solutions is not known a priori. One difficulty is posed by how a large and diverse class of interesting candidate solutions could be conveniently represented. Another closely related difficulty is determining the appropriate search algorithm that could effectively address such a search problem as (a) the size of the search space is very large and (b) the amount of principled domain knowledge available to carefully engineer the algorithm is limited. Evolutionary Algorithms (EAs) using generative representations of the candidate solutions have emerged as promising algorithms suited to address such problems [45, 33, 67, 37].

Rather than being a "representation" in the sense of a specific data-structure such as a bit-string as in Genetic Algorithms [30], or S-expressions as in Genetic Programming [47] or real-valued vectors as in Evolutionary Strategies [7], a generative representation typically refers to the case where the genetic data-structure is interpreted as a procedure (or a set of rules) that is recursively applied to generate the evaluatable candidate solution. In the context of EAs, these representations are "developmental" by analogy to the biological process of development associated with the reproduction of multicellular organisms from

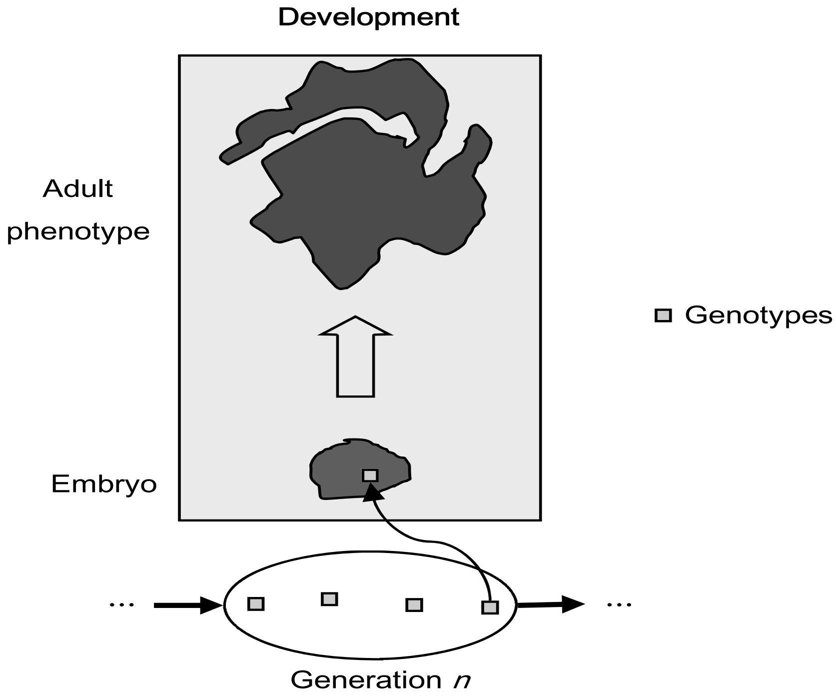


Figure 1.3: Schematic of the developmental process

single cells in each generation. The genetic representation in this case is equivalent to a "developmental program" and the evaluatable data-structure is the "phenotype". Figure 1.3 is a cartoon of the internals of this stage previously represented as a box in Figure 1.1.

For the sake of brevity, we will refer to the combination of an EA and such a developmental representation as a *Developmental EA* or a *devo-EA*.

In the theoretical models of Evolutionary Computation (EC), the formalism used to describe this relation between the two entities is the Genotype-Phenotype map $\psi : \mathcal{G} \to \mathcal{P}$, where $\mathcal{G}$ is the set of genotypes and $\mathcal{P}$ is the set of phenotypes. Due to the extreme generality of this formalism and its inability to distinguish developmental representations from any other, finding a suitable nomenclature has been a research theme and a variety of taxonomic classifications have been proposed based on differing criteria. Angeline [4] distinguishes between *translative*, *generative* and *adaptive* encodings; Kumar and Bentley [49] differentiate between *explicit* and *implicit* generative encodings. Hornby and Pollack [37] propose the use of constructs from high level programming language such as iteration and procedure calls for categorization. Stanley and Miikulainen [68] suggest a list of biologically inspired behavioral capabilities as a unified way to compare developmental representations independent of whether they are cellular based or grammar based.

Though we will introduce a specific mathematical notion of development in later chapters, for the purposes of this general introduction the notion of a developmental representation used here is one where:

1. the genome is interpreted as a set of rules,
2. the genome and phenome are both logically and physically distinct and are encoded with different substrates, and
3. the process of generating the phenome can be described as “(a) a process of (b) more or less gradual (c) change, (d) resulting in [...] one or more qualitatively different stages for which (e) the prior stages are necessary conditions” [73].

1.3 Approach

1.3.1 Theories of fitness evaluation

We posit that both coEAs and devo-EAs modify the conceptual structure of the problem of evolutionary search by actively involving a *theory of fitness evaluation*, explicitly in the case of coEAs and more implicitly in devo-EAs.

In the canonical formulation of an evolutionary search problem, the computation associated with fitness evaluation is usually characterized in functional terms as the *fitness function*, $f : \mathcal{P} \to \mathbb{R}$, where $\mathcal{P}$ is the set of evaluatable candidate solutions or phenotypes. Similarly, the computation associated with representation rewriting, developmental or otherwise, is characterized as the *Genotype-Phenotype map*, $\psi : \mathcal{G} \to \mathcal{P}$, where $\mathcal{G}$ is the set of genotypes and $\mathcal{P}$ is the set of phenotypes.

As the structure of the primary problem-relevant search space is determined by the genetic representation and its variation operators, the theoretical concepts of evolutionary computation have traditionally been indifferent to the details of the specific algorithms used to realize the fitness function and the Genotype-Phenotype map. This is to the extent that f and ψ are effectively treated in their composed form:

$$(f \circ \psi) = \underline{f} : \mathcal{G} \rightarrow \mathbb{R} \tag{1.3.1}$$

We will refer to $\underline{f}$ as the *compound fitness function* to distinguish it from the basic fitness function f. This lack of a theoretical commitment to a specific algorithmic conception of *how* the fitness function and the Genotype-Phenotype map are realized has made EAs a very general class of search algorithms suited for black-box optimization. However, both coEAs and devo-EAs deviate from this generality by explicitly adopting additional theoretical commitments about these algorithmic processes and their relevant domains. These additional commitments can be broadly summarized as follows.

Competitive fitness evaluation is premised on the explicit knowledge that the overall fitness of a candidate solution is a property of its behavior on a set of discrete "tests"; and the premises that:

[**C-1**] "Intelligently" drawn samples from this test set are sufficient to evaluate and evolve suitable candidate solutions for the search task.

[**C-2**] A competition based rationale is a suitable basis to perform the required "intelligent" sampling over the entire evolutionary search process.

[**C-3**] This "intelligent" sampling can be suitably realized with an EA.

The developmental protocol is premised on the explicit knowledge that the phenotypes have a component based internal structure and the general premises that:

[**D-1**] A suitable space of problem-relevant phenotypes can be algorithmically generated by the developmental system.

[**D-2**] The evaluation of the phenotypes obtained from the developmental process is a suitable basis to attribute fitness to the corresponding genotypes to search the space defined by $\mathcal{G}$ using an EA.

In this broad sense both coEAs and devo-EAs share a basic similarity in involving a specific theory about the algorithmic properties of $\underline{f}$. Coevolutionary Algorithms are based on a theory about the algorithmic realization of the actual fitness function f. Developmental-EAs are explicitly based on a theory about the algorithmic realization of the Genotype-Phenotype map ψ and in conjunction with [**D-2**] above involve an additional theoretical commitment to the nature of the relationship between this Genotype-Phenotype map and the actual fitness function f.

The popularity and widespread use of coEAs and devo-EAs would suggest that these premises are indeed satisfied. However, here we examine how faithfully and robustly they can *in fact* be realized in terms of the properties of the secondary substrate and its relationship to the problem structure.

1.3.2 Overview

In a standard EA, the progress of adaptation is determined by the likelihood of *favorable* (or adaptive) phenotypic variation arising in the population. The origin of such adaptive phenotypic variation depends on whether and how the variability of the phenotypes is related to the problem structure [75]. As genetic variation is the only source of heritable phenotypic variation in the canonical EA, adaptive phenotypic variability is effectively determined by the properties of the genomic data-structure or substrate [2, 75, 78, 60]. In general, a genetic representation that enables effective adaptation on a problem under the selection and variation operations of the EA is said to be *evolvable* [2, 75].

Broadly, to address the question of what makes a genome evolvable requires a precise notion of the problem and its "structure", and a way of establishing the correspondence of this structure to the properties of the genome and its associated variation operators on one hand, and the algorithmic dynamics of the evolving population on the other. Our approach is to apply a similar rationale to analyze the secondary substrate in coEAs and devo-EAs.

In the case of devo-EAs, we focus on the soundness of the premise [**D-2**] by considering it to be a hypothesis that requires empirical and theoretical validation. In devo-EAs, researchers often describe development as an “unfolding” of form as a result of genetic and environmental interactions. This non-quantitative description however provides no sense for how “hard” or “easy” it is for development to be effectively possible for *all* the genotypes, and the suitability of the secondary substrate for this purpose. In Chapter 2, we propose a computational model of development that allows the posing of quantitative questions about the controllability of development and its relation to overall performance. Specifically, we consider the question of how the choice of time at which development halts can impact the performance of the EA. This question is posed as the IDEAL DELIVERY problem, i.e. what the ideal time for development to halt and “deliver” the phenotype into the evolving population.

In the case of coEAs, we focus on whether premise [**C-3**] can be satisfied in such a way that the EA operating on the test-space can actually exhibit performance better than random search. For this to be possible, as in standard EAs as well, it requires the existence of some correlation between the properties of the genetic substrate and the structure of the problem. The obstacle here is the absence of a precise specification of the problem that the evolution of the “parasites” is expected to solve that is concrete enough to be able to deduce the properties of a suitable genetic representation. Using Juillé’s [41] IDEAL TRAINER/TEACHER model as the basis, in Chapter 3, we deduce the the variational properties required of a secondary substrate to enable two coevolving populations to remain continuously engaged and exhibit pathology-free learning with a Pareto-coEA [24, 53]. Using this model, we theoretically and empirically study the application of coEAs to boolean classification problems.

Chapter 4 provides a synthesis of the findings from these analyses.

Due to the very different terminology and machinery associated with coEAs and devoEAs, each of these chapters is written to be as self-contained as possible.

Chapter 2

Developmental-EAs and the Ideal Delivery problem

"A hen is only an egg's way of making another egg."

- Samuel Butler (1877)

2.1 Introduction

2.1.1 Motivation

A common practice in using generative (or developmental) representations [45, 33, 67, 37] with Evolutionary Algorithms (EAs) has been to run the generative process associated with a genotype till no further change occurs to the "embryo", or till a pre-defined maximum time limit or size is reached. At this point the resultant phenotype is then "delivered" into the evolving population where it is subject to the evolutionary methods of fitness evaluation and selection based on its relative fitness with respect to other members of the population. So far there has been no theoretical justification proposed as to why this "delivery" protocol is a principled way to use developmental representations with an EA. Indeed, when development is considered as being no more than a process of mapping a genetic data-structure to a phenotypic one, this would seem to be the most obvious way to do things. For instance, this would be an odd issue to raise with respect to the arithmetic operation of mapping the binary representation of a number to its decimal form. However, the operation relating genotypes to phenotypes in generative representations is significantly unlike such an arithmetic operation even if both can be abstractly described as realizing a Genotype-Phenotype map.

Two such distinctive properties of generative representations are (a) an indirect *procedural* relationship between the elements of the genome and phenome and the associated need for (b) an *iterative generative process* to obtain the phenotypes. The hypothesis that we present here is that the impact of the secondary substrate on evolutionary performance arises due to this latter property involving the iterative generative process, and this impact exerts itself via the choice of "delivery" time. In this chapter, we present a mathematical formulation of this hypothesis for the basic type of devo-EAs and an empirical assessment of its immediate consequences for evolutionary performance. This formulation takes the form of the IDEAL DELIVERY problem. The problem simply put is - *What is the "ideal" time to halt development and deliver the phenotype into the population?*

2.1.2 Approach

To precisely formulate the IDEAL DELIVERY problem we adopt the analytical stance that there may be value, in terms of evolutionary search performance, in being able to *actively* pre-empt the delivery of the developing phenotype at a point prior to the inevitable termination of the process. Since the embryonic individual undergoes change over the developmental process, a mechanism that can terminate development at any point in time could also vary the phenotype that is delivered into the population. So, the narrow problem to be solved here is to (a) characterize how the choice of delivery point impacts evolutionary performance; (b) determine which of the many possible delivery points over the developmental process has the most desirable impact on performance, i.e. is "ideal"; (c) ascertain whether this "ideal" delivery point corresponds to the default protocol; and (d) identify the consequences of *not* being able to realize this "ideal" delivery.

The very possibility of being able to consider alternate "delivery" protocols is tied to the unique properties associated with having an explicit generative or developmental phase. This is obscured in treating generative representations strictly in terms of the Genotype-Phenotype map formalism. So the first step is reconceptualizing the relation between development and evolution at this basic formal level. This is the subject of section 2.2 where we propose a simple alternative developmental model that is compatible with the notion of the Genotype-Phenotype map but provides a mechanistically richer notion of development.

Using this model, in section 2.3 we show that when developmental control is parameterized by time it results in a space of possible Genotype-Phenotype maps and we pose the IDEAL DELIVERY problem as one of identifying the "ideal" map in this space. The notion of "ideal" is with respect to evolutionary performance under the constraint that no additional knowledge about the properties of the problem and the variation operators is available beyond that already available to the EA. This leads to our main finding.

We analytically show that the impact of the choice of delivery point can impact the *selection probability* of the associated genotypes by introducing a *bias* in selection. The particular Genotype-Phenotype map in the space of maps that guarantees that selection is not negatively biased by the choice of delivery time is equivalent to the Nash Equilibrium for a multi-player game. In this game every genotype is a player and each possible delivery point is a strategy. This Nash map is dependent on the fitness function and unless the variability of the secondary substrate under development is correlated with the fitness function, it is not guaranteed to correspond to the default delivery protocol.

In section 2.4 these analytical predictions are empirically tested on a grammar based generative representation. The experiments unambiguously reveal the occurrence of a selection bias with the default protocol and furthermore shows the positive impact on performance when using a Nash delivery point (explicitly computed in this case) even on a genetic representation that Hornby [37] has shown to have poor evolvability characteristics.

These findings establish that for developmental-EAs, evolvability in its broadest sense encompasses not just genetic variability but also the correlation of *developmental variability* with problem structure, i.e. the properties of developmental change can influence evolutionary performance.

2.2 Model

This section describes the developmental model that forms the basis for our investigation. The main novelty of this model is to provide an alternative to the extreme generality of the concept of the Genotype-Phenotype map with a more specific computational notion explicitly tuned to development. This formally involves a mapping from the genotype to the *actions* of a collection of developmental control mechanisms, where the actual phenotype arises from the controlled dynamical interactions of these mechanisms. The explicit focus is on modeling computational developmental systems and no claims are made with regard to biological correctness.

2.2.1 Evolutionary model

In an early paper describing the possible avenues for research on developmental-EAs, Angeline [4] describes a simple formulation to express the concept of an indirect genetic encoding which can be summarized as follows.

Let $\mathcal{P}$ be the set of objects of interest. The objective is to find an object in $\mathcal{P}$ that has certain desired properties expressed as a fitness function $f : \mathcal{P} \to \mathbb{R}$ that provides a measure of the relative suitability of each object to this desired end. So the general search problem is to find a phenotype $\phi \in \mathcal{P}$ that has a fitness value at or "sufficiently near" a desired maximum in the range of f using an Evolutionary Algorithm.

Suppose P is a vector representing a (multiset) collection of entities from $\mathcal{P}$ i.e. the population, and, for convenience, let f be such that $f(P)$ produces a vector of real-numbered values corresponding to the evaluation of each of the members of P. A general evolutionary computation can be defined as an iteration of the equation:

$$P' = \rho(P, f(P)) \tag{2.2.1}$$

The function ρ is called the *reproduction function* as it generates a new collection of entities from $\mathcal{P}$ from the previous collection P (for a given random seed). This reproduction function performs both selection and variation operations. So evolution is in effect the iterative application of this function till some external stopping criterion is reached. Here, the objects in $\mathcal{P}$ are referred to as the *evaluated* entities and as they are also the *evolved* entities as they are directly manipulated by ρ.

Morphogenic Evolutionary Computation (as Angeline referred to it) is contrasted with this basic form by the introduction of an abstraction that separates the evaluated entities from the evolved entities. The reproduction function for a morphogenic evolutionary computation is defined as

$$G' = \rho'(G, f(\psi(G))) \tag{2.2.2}$$

The function $\psi : \mathcal{G} \rightarrow \mathcal{P}$ is called the *development function*. In keeping with common usage, this will alternatively be referred to the Genotype-Phenotype map. G is a vector representing a (multi-set) collection of entities from $\mathcal{G}$, and $\psi(G)$ is considered to produce a vector P by the application of ψ to each of the elements of G.

To go beyond this typical formalization of development as a Genotype-Phenotype map, we next introduce a more detailed model of computational development. The intent of this model is to serve as a reductionist framework to scrutinize the aspects of development that are of immediate relevance to evolutionary performance while screening off to the extent possible the details of lower-level operations.

2.2.2 Developmental model

The internal workings of the Genotype-Phenotype map are considered here to consist of an interactive protocol involving three key functions that are described below. For simplicity, only the deterministic version is considered here.

Let $\mathcal{E}$ be the set of ontogenetic intermediates possible for a given generative system. Let the set $\mathcal{E}_0 \subset \mathcal{E}$ consist of special start states which we assume has exactly one member

$\mathcal{E}_0 = \{e_0\}$. In the case of "embryonic" development, the embryonic members of $\mathcal{E}$ may not by themselves be valid phenotypes in $\mathcal{P}$. The fitness function is considered to be undefined on the entire set $\mathcal{E}$ and treated as being well defined only on the set of "adult" phenotypes $\mathcal{P}$. Let $\mathbb{E}$ be the environment.

Let $\overline{\gamma}$ be a *generative function* which has a behavior that is parameterized by the genotypes in $\mathcal{G}$ and the environment

$$\overline{\gamma} : \mathcal{G} \times \mathbb{E} \rightarrow (\mathcal{E} \rightarrow \mathcal{E}) \tag{2.2.3}$$

The operation of this function is considered to occur in an iterative fashion where $e_{t+1} = \overline{\gamma}(e_t)$. Each iteration is controlled by two additional functions, a *decision function* $\overline{\delta}$ and a *delivery function* $\overline{\beta}$.

The function $\overline{\delta}$ represents a decision mechanism that provides the basis for when embryogenesis terminates and "delivery" occurs. For simplicity, this is treated here as being a Boolean function

$$\overline{\delta} : \mathcal{G} \times \mathbb{E} \rightarrow (\mathcal{E} \rightarrow \{0, 1\}) \tag{2.2.4}$$

where for an $e \in \mathcal{E}$, $\overline{\delta}(e) = 1$ if the decision criterion is satisfied and 0 if not. We assume that it is guaranteed to return a 1 at some point.

As elements of $\mathcal{E}$ are not considered to be valid phenotypes in themselves, an additional mechanism is required that can return a valid phenotype from this process at some point, i.e. serving as the equivalent of "delivery". This is achieved with the *delivery function* $\overline{\beta}$ that we assume is invoked only if $\overline{\delta}(e) = 1$. It is defined to be

$$\overline{\beta} : \mathcal{E} \rightarrow \mathcal{P} \cup D \tag{2.2.5}$$

where D denotes the set of "premature" or "incomplete" states in the sense that $\overline{\beta}(e) \in D$ if $e \in \mathcal{E}$ is still "incomplete" even when $\overline{\delta}(e) = 1$. The value of the fitness function on D is treated as being $f(d) = c$ where $d \in D$ and c is a constant such that $c < f(\phi)$ for all $\phi \in \mathcal{P}$.

Based on this algorithmic notion of development, the *ontogeny* (or developmental history) of an individual is defined to be an ordered sequence

$$\overline{O} = \langle (e_0, \overline{\delta}(e_0)), (e_1, \overline{\delta}(e_1)), ..., (e_t, \overline{\delta}(e_t)) \rangle \tag{2.2.6}$$

where $e_i \in \mathcal{E}$ and i $(0 \leq i \leq t)$ is an index indicating the time-step at which e_i occurs in $\overline{O}$. This is a multi-set in the sense that e_i can be identical to e_j when $i \neq j$ as the developing embryo may not always be distinguishable from time-step to time-step. Since delivery occurs when $\overline{\delta}(e) = 1$, the value of $\overline{\delta}(e_i)$ is equal to 0 for all $i < t$ and 1 for $i = t$. The *duration* of an ontogeny is given by $u(\overline{O}) = |\overline{O}|$. The set of all ontogenies is denoted by $\overline{\mathcal{O}}$.

For the sake of analysis, the functions described above can be redefined as follows

$$\overline{\gamma_\delta} : \mathcal{G} \times \mathbb{E} \rightarrow \overline{\mathcal{O}} \tag{2.2.7}$$

and

$$\overline{\beta} : \overline{\mathcal{O}} \rightarrow \mathcal{P} \cup D \tag{2.2.8}$$

where $\overline{\beta}$ only operates on the last element of the ontogeny. In this way, the Genotype-Phenotype map can be seen as a composition

$$\psi = \overline{\beta} \circ \overline{\gamma_\delta} : \mathcal{G} \times \mathbb{E} \rightarrow \mathcal{P} \tag{2.2.9}$$

The new pieces of information in this formulation and absent from the Genotype-Phenotype formalism (as described in Section 2.2.1) are:

1. The developmental history of the phenotype over the process of development, and

2. The attribution of the phenotype's characteristics to these systems level computational mechanisms rather than the genotype alone.

The issue we will address next is whether this additional information can be used in some way to make predictions about problem difficulty and evolutionary performance that would be difficult to make otherwise. Specifically, whether differences in how develop-

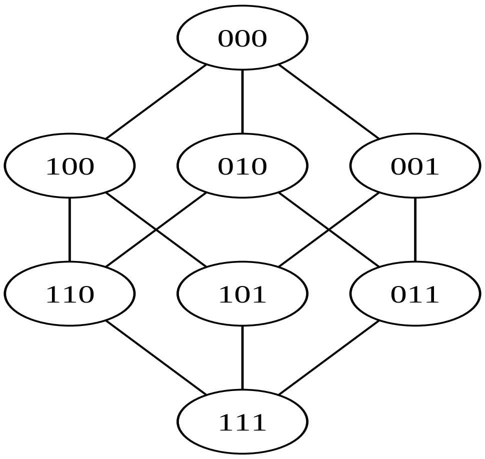


Figure 2.1: A graph Γ with $m = 2^3 = 8$ vertices

mental control mechanisms use this ontogenetic information translates into differences in evolutionary performance. As a prelude to this analysis, we now provide a detailed example to clarify the concepts introduced above and to provide an intuition for the analysis to follow.

2.2.3 Example

Here we provide an example with a strong theoretical computer science flavor, loosely based on Balcazar's succinct representation model [8]. The deliberate simplicity of the example is to provide a sense for the properties of generative representations that are often obscured in the more "biologically inspired" representations used in practice.

Consider a undirected connected graph $\Gamma(V, E)$ where V is the set of vertices and E is the set of edges. Let m be the total number of vertices in V. The graph is such that $m = 2^n$ ($n > 0$) and each vertex has exactly n neighbors. An example is shown in Figure 2.1.

An *explicit* representation of this graph is an adjacency matrix of size m^2. Let this explicit representation be denoted by Γ_x. An instance of such a graph can be represented in an *implicit* form Γ_i with a deterministic procedure μ_n. This procedure is such that it takes as input a bit string b of length n and returns all the binary strings of length n that differ in 1-bit from b. When the size of the implementation of μ_n is proportional to n, it can be considered to be a *succinct* representation of Γ_x as the adjacency matrix of size 2^{2n} is now represented with a procedure of size n.

The explicit representation Γ_x can be obtained from the implicit Γ_i using an iterative generative process.

1. Start with an empty matrix Γ_0 of size $m \times m$.

2. Pick a bit string b_0 in $\{0,1\}^n$ and add it to the list B.

3. Pop the first element from B and feed it as input to μ_n and obtain all the outputs and add them to the list B'.

4. For each string $b_i \in B'$ enter 1 in the adjacency matrix in locations $(\underline{b}, \underline{b_i})$ and $(\underline{b_i}, \underline{b})$, where $\underline{b}$ is the decimal representation of the bit vector b.

5. Add B' to the end of list B excluding any strings already present in B.

6. IF every row in the adjacency matrix has n non-zero entries then exit and return the adjacency matrix Γ_e ELSE goto step 3.

A snapshot of this process of generating the explicit representation of a graph having $m = 2^3$ vertices is shown in Figure 2.2.

This procedure μ_n is an example of an *implicit* representation Γ_i of the graph Γ as the entire description of the graph (as with the adjacency matrix) is not explicitly stored. This "implicitness" of this representation is achieved by the re-labeling the vertices of Γ to make them correspond to the labels of a binary hypercube of size n that is isomorphic to Γ. Due to this relabeling, the procedure can take as input the binary label of a vertex and generate all its neighbors. Since it is a connected graph, every vertex and edge is eventually produced by iteratively using μ_n.

To provide a more "biological" notion of this process, consider the first vertex 000 to be akin to a cell containing μ_n as its genome. At each time step, rather than returning bit-string outputs, the cell duplicates to produce the cells 001, 010 and 100 with each having its own replica of μ_n. In subsequent steps, each of these cells again duplicates in parallel with any

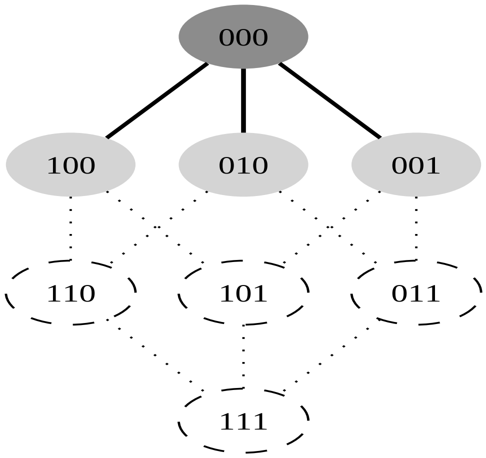


Figure 2.2: Subgraph of a graph Γ of size $m = 2^3$ obtained after initial input $b_0 = 000$ (dark gray) to μ_3 and the successors obtained (light gray).

competitive overlap being resolved randomly. In this fashion the entire graph Γ is obtained within n steps.

The terms of our developmental model can be mapped on to this process as follows:

- The graph described above is one phenotype $\phi \in \mathcal{P}$ and the phenotype space $\mathcal{P}$ is the set of such regular graphs with different numbers of vertices.
- The state of the adjacency matrix is equal to $e \in \mathcal{E}$ and the set of all possible states of the adjacency matrix is $\mathcal{E}$. The starting state of the explicit representation Γ_0 is the equivalent of e_0.
- The generative function $\overline{\gamma}(e)$ is an abstraction of the processes of feeding inputs to μ_n, reading the outputs, adding and deleting items from the list, performing locate and write operations on the adjacency matrix. One entire iteration results in up to 3 new entries in the adjacency matrix and a minimum of 0. It is parameterized by μ_n which is the equivalent of a genotype.
- The decision function $\overline{\delta}(e)$ is the procedure checking whether the condition that all rows of the adjacency matrix have n non-zero entries or alternatively to checking if the list B is empty. If these conditions are not satisfied $\overline{\delta}(e) = 0$ else $\overline{\delta}(e) = 1$

- If $\overline{\delta}(e) = 1$ then the delivery function $\overline{\beta}(e)$ returns Γ_x as a result.

While the genotype performs the key operations, we can see that the ability to generate the explicit representation requires a lot of extra machinery, well-defined initial conditions, a protocol for these operations to occur together, and a mechanism to halt the process. The generative, decision and delivery functions are abstractions of these processes focusing only on the effects perceptible in terms of changes in the explicit representation, which in the above example was the adjacency matrix. The reason for this abstraction is to exclusively focus on the entities on which evolutionarily relevant functionality is defined even though it is determined by lower level mechanisms. As may be evident what these abstractions hide may be intensely complicated.

A key point in the above example is that $\mathcal{E}$ is *not* identical to $\mathcal{P}$ as the graphs corresponding to intermediate states of the adjacency graph may not qualify as being *regular* graphs as defined earlier, i.e. where the number of vertices $m = 2^n$ $(n > 0)$ and each vertex has exactly n neighbors. Figure 2.2 is an example of a graph that appears as an intermediate stage but does not meet these criteria. In this regard, the role played by $\overline{\delta}$ is critical as it is the recognition mechanism that determines when the development process is complete and delivery is to occur, even if in this case it was the equivalent of a simple conditional.

A different situation presents itself when the constraints on the properties of the phenotypes are a lot weaker. For example, if the phenotype set were the set of *all* connected graphs rather than only regular graphs then each intermediate is a valid phenotype. This is often the case for the “open-ended” generative systems used in an Evolutionary Computation settings. In such a setting the basis for the decision function becomes particularly interesting as it can now be interpreted in terms of its implications on the overall search problem and the performance of the EA being used. In this context we can now assess the broader adaptive significance of these developmental decisions over and above their immediate consequences for the construction of form. In the next section, we show that some decisions are “better” than others.

2.3 Developmental decision making

2.3.1 Assumptions

To differentiate the case where the embryonic intermediates are themselves valid phenotypes from the general model defined in the previous section, the same notation is retained but the bar on top of the symbols is dropped and some of their behaviors are simplified as described below.

Here the set of ontogenetic intermediates $\mathcal{E}$ is given by $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{P}$. The set D is assumed to be the empty set and the delivery function β is treated as being the identity function. An ontogeny $O \in \mathcal{O}$ takes the form

$$O = \langle (e_0, \delta(e_0)), (\phi_1, \delta(\phi_1)), ..., (\phi_t, \delta(\phi_t)) \rangle \tag{2.3.1}$$

Let $\tau_{max} = \arg \max_{O \in \mathcal{O}} u(O)$. This value τ_{max} ($\tau_{max} > 0$) is the maximum of all the durations of the ontogenies in $\mathcal{O}$. This value will serve as our reference point. So, $\mathcal{O}$ is effectively the set of all ontogenies where $0 < u(O) \leq \tau_{max}$. For notational simplicity, a phenotype ϕ_i will be considered to be an element of an ontogeny O if $(\phi_i, *)$ is an element of O where $*$ could either be 0 or 1.

The action of the decision function δ was defined earlier was treated as being co-determined by the genotype and the environment. However, as the ontogenetic states in O are valid phenotypes, $\delta(\phi_i)$ can in principle take a value of 1 at any point in the ontogeny for $i > 0$. To emphasize this freedom, here we treat δ as being able to take actions based purely on external information. Furthermore, we introduce a behavioral separation between the generative function and the decision function. We consider $\gamma_*(g)$ to be the ontogeny generated by the generative function γ with a genotype g when the decision function is *passive*, i.e. when development ends if the embryo does not exhibit any further change or the maximum time limit is reached rather than being *actively* terminated by the decision function.

In informal terms, the problem to be solved here is finding a principled and general basis by which δ can make an *active* decision about the time of delivery for the ontogenies corresponding to *every* genotype in $\mathcal{G}$, for a given generative function γ and an unknown fitness function.

By "unknown" fitness function we mean the following. Let $\mathcal{F} = \{f_1, f_2, ..., f_q\}$ be a finite *class* of fitness functions where $q \geq 1$ and each fitness function in $\mathcal{F}$ is of the form $f_i : \mathcal{P} \rightarrow \mathbb{R}$. When $q = 1$, this class corresponds to the regular situation where the focus is on a specific fitness function. Our interest is on the case where $q > 1$ and where the algorithm designer knows $\mathcal{F}$ but does not know which fitness function is being used for evaluation. This class $\mathcal{F}$ could, in principle, be fitness functions that are closed under permutation as in the "sharpened" No Free Lunch scenario [78, 66]. The objective is to design the overall algorithm and representation to be able to provide the desired level of performance on any of the fitness functions in this class. Our focus will exclusively be on the role of δ in this task.

All future references to fitness functions will be in this sense unless stated otherwise. This is not an unrealistic scenario as developmental representations have been of particular interest in situations where the desired form of the solution is not known a priori [67, 68]. This is a closely related equivalent to not having a priori knowledge about the properties of the fitness function.

Based on these general assumptions, we now take a step toward a more precise definition of the design problem for δ by defining the space of options available to such a decision function.

2.3.2 Decision space

The combined space of options for the decision function associated with ontogenies of each genotype can be described as a $|\mathcal{G}|$-dimensional space

$$\Theta = \{1, 2, 3..., \tau_{max}\}^{|\mathcal{G}|} \tag{2.3.2}$$

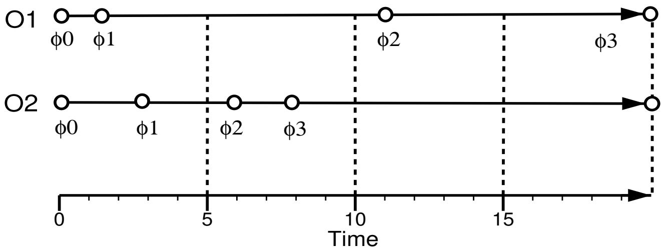


Figure 2.3: Two ontogenies O_1 and O_2 with the distinct phenotypes shown as white circles, and the delivery points corresponding to $\tau = 5, 10, 15$ and 20 units.

Each point $\theta = (\tau_1, \tau_2, ..., \tau_{|\mathcal{G}|})$ in Θ represents one setting of the decision function for all the genotypes taken together where τ_i $(0 < \tau_i \leq \tau_{max})$ is a time at which the decision function can take a value 1 during the generative process with genotype $g_i \in \mathcal{G}$. The reference point when all the values in $\theta \in \Theta$ are equal to τ_{max} is denoted as θ_{max}. This corresponds to the usual practice of running the developmental process till no further changes occur in the developing phenotype or till it reaches the maximum time limit τ_{max}.

Framed in this manner, *every point $\theta \in \Theta$ now defines a possible Genotype-Phenotype map*. To see how and why, consider the following example.

Example Consider the ontogenies shown in Figure 2.3 where $O = \gamma_*(g)$ and $O' = \gamma_*(g')$ for two genotypes $g, g' \in \mathcal{G}$, and $\tau_{max} = 20$. Only the distinct phenotypes are shown in this figure. Figure 2.4 shows the low-dimensional projection of the space Θ corresponding to the two dimensions associated with genotypes g and g' and a sampling of points in this space. Each point in Θ represents the time at which the decision function returns 1 for the ontogeny associated with each genotype. Since each time instant in an ontogeny is associated with a phenotype, this implicitly specifies the phenotype that is to be "delivered" into the population. The corresponding Genotype-Phenotype correspondences are listed in Table 2.1 where ψ_θ denotes the map at θ.

In general, the size of Θ is exponential in the size of the genetic search space as $|\Theta| = (\tau_{max})^{|\mathcal{G}|}$. However, this overcounts the number of maps as each point $\theta \in \Theta$ does not necessarily correspond to a unique Genotype-Phenotype map as the embryo may not always

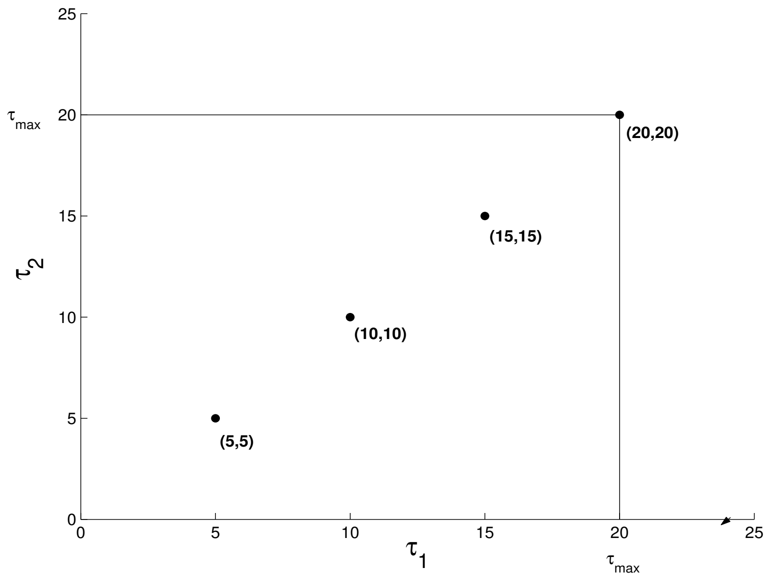


Figure 2.4: A sampling of points in the low dimensional projection of Θ corresponding to the ontogenies O_1 and O_2

change at every time step, as also in the above example. If $l(O)$ is the number of distinct phenotypes in the ontogeny O, then the number of distinct maps is given by $\prod_{g_i \in \mathcal{G}} l(\gamma_*(g_i)) \leq (\tau_{max})^{|\mathcal{G}|}$.

θ	$\psi_\theta(g)$	$\psi_\theta(g')$
(5,5)	ϕ_1	ϕ_1
(10,10)	ϕ_1	ϕ_3
(15,15)	ϕ_2	ϕ_3
(20,20)	ϕ_3	ϕ_3

Table 2.1: Genotype-Phenotype correspondences for different points in Θ

With this abstraction, the IDEAL DELIVERY problem can be narrowed down to one of picking the "ideal" Genotype-Phenotype map from among the points in Θ. This is a critical choice as it would in turn define the evolutionary search problem for an unknown fitness function belonging to $\mathcal{F}$. The next task is to define a suitable solution concept to make this choice based on a principled notion of what kind of effect on performance we expect and would want a suitable decision to have (or not have); and one which could be practically observed.

2.3.3 Solution concept

As with any representation, the specific Genotype-Phenotype mapping has a strong impact on the performance of an Evolutionary Algorithm. For a particular fitness function and set of genetic variation operators, different maps can make a search problem more or less "difficult". For example, the two different maps shown in Figure 2.5 corresponding to different points in Θ can be very different in terms of problem difficulty.

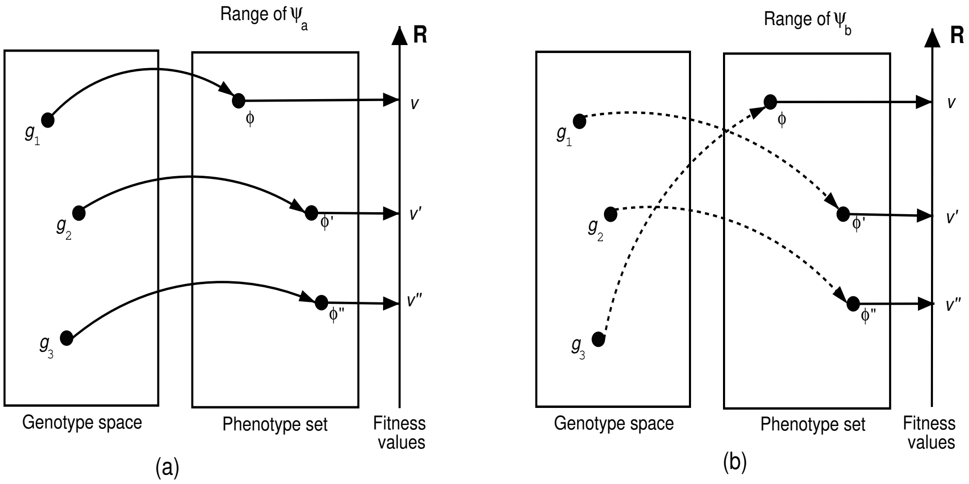


Figure 2.5: Differing outcomes of development for the same genotype and phenotype set

Given the existence of this parameterized space of possible maps, one idealized solution concept for this decision problem would be to choose a point θ_{easy} in Θ that maximally reduces the problem difficulty for the particular EA being used. However, this itself presents several practical difficulties. Firstly, finding such a point θ_{easy} corresponds to a meta-search problem on the space defined by Θ which has a size that is exponential in the size of the genetic search space as noted earlier. Secondly, since the fitness functions are drawn from a class of fitness functions $\mathcal{F}$, this meta-search problem would need to be addressed over this entire class. These are issues over and above the more elemental difficulty of being able to accurately define a quantitative measure of problem difficulty so that two points in Θ (i.e. their corresponding Genotype-Phenotype maps) can be compared.

Apart from this global effect on problem difficulty, the dynamics of development exert another kind of influence on evolutionary performance. The timing of delivery can impact performance by introducing biases in the "greedy" rationality of Darwinian *selection* independent of the properties of the genetic search space.

Since each time step in an ontogeny $O \in \mathcal{O}$ is associated with a phenotype, fitness values can vary along the ontogeny in different ways depending on the fitness function. To provide a concrete example, consider the scenario in Figure 2.6 that shows the variation in fitness for a fitness function f along two ontogenies O and O' corresponding to genotypes g and g' respectively over the time window defined by $\tau_{max} = 8$. For two such genotypes, it can be the case that $f(\psi_\theta(g)) > f(\psi_\theta(g'))$ at θ and $f(\psi_{\theta'}(g)) < f(\psi_{\theta'}(g'))$ at θ' for some θ, θ' in Θ. In words, the genotype g can be associated with a higher fitness than genotype g' with Genotype-Phenotype map ψ_θ. However with the map $\psi_{\theta'}$ the converse in that g' can be associated with have a higher fitness than genotype g. In Figure 2.6, when $\theta = \theta_{max}$ we see that genotype g' is associated with a higher fitness than g. However, we can also see that the ontogeny O has a phenotype at $\tau = 6$ that has a fitness value greater than that of all the phenotypes in ontogeny O'. Given that the goal of the search problem is to find high fitness phenotypes in $\mathcal{P}$, at θ_{max} the Genotype-Phenotype map produces a selection bias that does not correspond with these objectives.

Since search using evolutionary algorithms is directed by selection acting on fitness differences in the members of the population, we see that the choice of θ is a source of biases that can impact evolutionary dynamics negatively. It can conceivably retard the overall rate of improvement in fitness as genotypes that generate high fitness phenotypes at points in the ontogeny that do not correspond to the delivery point can be under-evaluated and possibly even prematurely lost from the evolving population. In the extreme case these high fitness phenotypes could even correspond to the global optima for the search problem. This bias is noticeable only when we consider the additional information that ontogenies bring to the analysis of a problem as it is an implicit effect that occurs *within* the Genotype-Phenotype map. So, the solution concept we focus on here is the "ideal" point in Θ that guarantees that such a bias is absent.

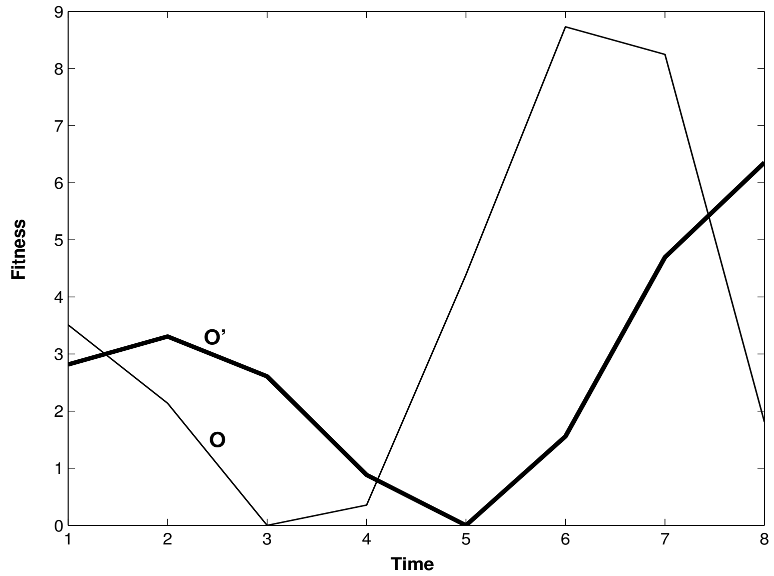


Figure 2.6: Fitness variation along ontogenies O and O'

2.3.4 The developmental Nash Equilibrium

The scenario presented by selection bias corresponds to a "game" in the formal game theoretic sense where there are multiple players each seeking to pick a strategy that would maximize their individual payoffs.

The space Θ in conjunction with a fitness function $f \in \mathcal{F}$ defines a multi-player non-cooperative game where every genotype g_i in $\mathcal{G}$ is a player and the set of strategies available to it are defined by the range of the i^{th} dimension of the space Θ, namely, $\{1, 2, 3, ..., \tau_{max}\}$. So each point $\theta \in \Theta$ is a *strategy profile* describing a configuration of strategy choices made by each of the players.

The "game" corresponding to the scenario shown in Figure 2.6 is represented below in the strategic form where the row-player is genotype g and the column-player is genotype g'. When using fitness proportionate selection, the "payoff" to the row-player is given by $f(\phi_i) - f(\phi'_j)$, where ϕ_i is the phenotype occurring at the i^{th} time step on the ontogeny O and ϕ'_j is the phenotype at the j^{th} time step on the ontogeny O'. To provide a better intuition for this game, this payoff structure is presented in a simplified form where g gets a payoff of $+1$ if $f(\phi_i) \geq f(\phi'_j)$ and -1 otherwise. Only the payoffs of the row player are shown below.

These payoffs serve as a measure of the selection bias associated with g and g' for different values of θ as we can see that they can be associated with a higher or lower relative fitness depending on the choice of θ.

$$\mathbf{Game} = \begin{array}{c|cccccccc} (g,g') & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & +1 & +1 & +1 & +1 & +1 & +1 & -1 & -1 \\ 2 & -1 & -1 & -1 & +1 & +1 & +1 & -1 & -1 \\ 3 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 4 & -1 & -1 & -1 & -1 & +1 & -1 & -1 & -1 \\ 5 & +1 & +1 & +1 & +1 & +1 & +1 & -1 & -1 \\ 6 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ 7 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ 8 & -1 & -1 & -1 & +1 & +1 & +1 & -1 & -1 \\ \hline \end{array}.$$

Let $\mathbf{F}(\theta) = (f(\psi_\theta(g_1)), f(\psi_\theta(g_2)), ..., f(\psi_\theta(g_{|G|})))$ be the *fitness profile* for $\theta \in \Theta$ for the fitness function $f \in \mathcal{F}$. Let $\mathbf{F}_i(\theta)$ be the fitness of the i^{th} player (i.e. genotype g_i). We can see that the concept of a Nash Equilibrium θ_{Nash} where

$$\mathbf{F}_i(\theta_{Nash}) \geq \mathbf{F}_i(\theta), \quad \text{for all } i \ (1 \leq i \leq |\mathcal{G}|) \text{ and for all } \theta \in \Theta \tag{2.3.3}$$

corresponds to the map for which fitness evaluation is guaranteed to be *unbiased* for every genotype. At any other point $\theta \neq \theta_{Nash}$ ($\theta, \theta_{Nash} \in \Theta$), some genotype in $\mathcal{G}$ is associated with a fitness that is lower than the maximum fitness that it could be associated with.

We propose this as a solution concept to the IDEAL DELIVERY problem.

> **Solution concept**: A Genotype-Phenotype map θ in Θ results in unbiased selection for the space of possible maps defined by Θ and the fitness function $f \in \mathcal{F}$ if and only if it is a Nash Equilibrium.

2.3.5 Summary

This solution concept now provides us with a simple and direct answer to the question posed earlier - what is the ideal time to terminate development and deliver the phenotype. As the payoffs are dependent on the totally ordered fitness function $f : \mathcal{P} \rightarrow \mathbb{R}$, a Nash Equilibrium for Θ and a fitness function f in $\mathcal{F}$ is equivalent to δ picking ϕ_τ for which $f(\phi_\tau) \geq f(\phi_i)$ for all $\phi_i \in O$ and for all ontogenies $O \in \mathcal{O}$, i.e. it corresponds to a Genotype-Phenotype map where the phenotypes delivered into the population are the ones having the maximum fitness as compared to the other phenotypes in their corresponding ontogenies.

This leads to the following observations:

1. The Nash map is a property of the entire Genotype-Phenotype map, not a property of any one genotype.

2. The Nash can differ from one fitness function to another, so using the same Genotype-Phenotype map across multiple fitness functions is bound to result in biased selection.

3. **Prediction**: If the Genotype-Phenotype map is not a Nash map, then we can expect to see high fitness phenotypes being generated but lost from the population.

In the next section we explore the characteristics of evolution with the maps $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ using a strategy similar to the comparative studies described in [46, 37, 49]. The strategy in these studies was to keep the fitness function, selection and population size fixed over the evolutionary run for the encodings being compared. So the observed differences in the fitness increase and absolute fitness over a fixed number of fitness evaluations with these differing encodings were attributed to differences in the evolvability of the genetic encoding (and their associated variation operators). In this case as well the fitness function, selection and population size as well as the genetic encodings and variation operators are identical for both $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ with the only difference being the basis used for the ontogenetic decision making.

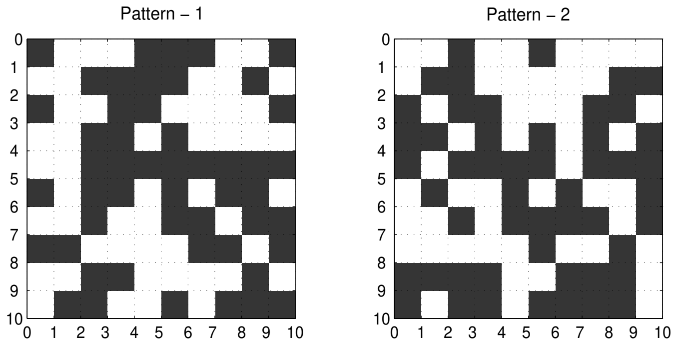


Figure 2.7: Target patterns (randomly generated)

2.4 Experiment

2.4.1 Problem

The toy-problem chosen here is one of pattern construction, similar to that used in [49]. A target pattern consists of a bit-pattern on a 10×10 grid (Figure 2.7). The general objective is to evolve a pattern that corresponds to this target pattern. Here, the phenotype space $\mathcal{P}$ of interest is the set of *connected patterns* on this 10×10 grid, even though the target patterns are not connected. This is irrelevant as the goal is to study the evolutionary dynamics rather than the properties of the solutions obtained.

The fitness function f is defined as follows: Given a phenotype $\phi \in \mathcal{P}$, and the target pattern A, a value x is assigned to each cell of ϕ that matches the target pattern, and a value y is assigned to each cell that does not match this target. The overall fitness is the sum of the x and y values of each cell of ϕ. Here $x = +10$ and $y = -10$. If the phenotype has only one cell, it is assigned a fitness of -500.

2.4.2 Developmental system

The generative function

The developmental system used is based on the the well known mathematical animal - the LOGO turtle. The turtle's movements are controlled by the execution of a procedure based on heading and orientation commands. The trace of the turtle's movements in space forms the basis for the construction of geometric objects (for details see [1]).

To focus on the canonical properties of evolution with a developmental phase, here we consider a simplified version of this turtle. The turtle moves on the 10×10 grid defined by the problem. The basic commands accepted by the turtle are: `forward`, `back`, `left`, `right`. The commands `forward` and `back` change the turtle's *position*, while the commands `left` and `right` change both the turtle's *heading and position*.

The turtle always moves in unit steps. The default input for the `right` command is $90°$ with respect to the current heading, and $-90°$ for the command `left` with respect to the current heading, with a single unit step taken in the new orientation for both commands. So the turtle always moves with the direction of its heading being parallel to one of the (globally defined) principal axes.

All the cells on the grid are assumed to have an initial state 0, and each time the turtle visits such a cell, the state changes to 1. Once the state changes from 0 to 1, it remains unchanged. As a result, the turtle's movements on executing a series of commands result in a connected pattern defined by the cells having the state 1. The start position for the turtle is always at $(5, 5)$ on the grid pointing upward, i.e. in the $Y-$ direction.

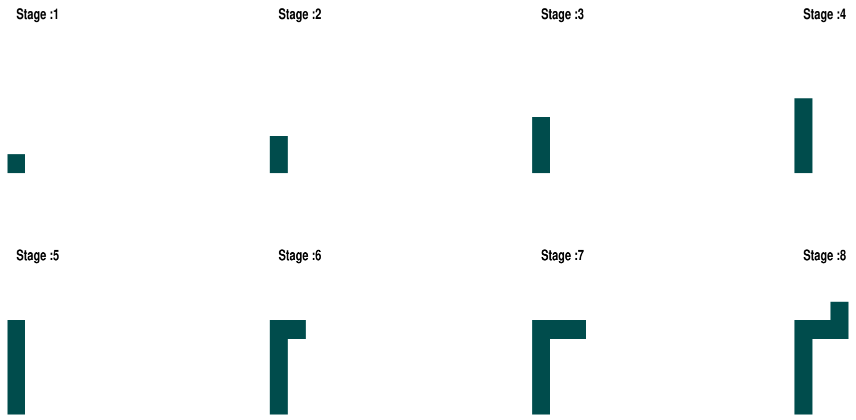
The procedure executed by the turtle takes the form of a list of these commands that are executed in sequential order. The set of all such procedures (with a minimum length $=$ 5 and maximum length $= 200$) is taken to be the set of genotypes $\mathcal{G}$. Every connected trace produced by the turtle is a valid phenotype $\phi \in \mathcal{P}$. The ordered sequence of phenotypes produced during the execution of a particular genotype g starting from the first command (as shown in Table 2.2) is considered to be an ontogeny.

Decision function

The phenotype obtained when all the commands in g have been executed is the phenotype $\psi_{\theta_{max}}(g)$.

To simulate the effect of the Nash map $\psi_{\theta_{Nash}}$, the phenotype having the maximum fitness is determined by exhaustively evaluating the fitness of every phenotype in the onto-

Table 2.2: Example of phenotypic change over an ontogeny



genies and this fitness value is used as the basis for selection.

Variation and selection operators

The variational operators include both mutation and crossover. Three mutational operators were used. Mutational operator M_1 randomly replaces a randomly selected command (with uniform probability) on the given procedure by one of the other three commands. The operators M_2 and M_3 were specifically designed noting that the procedures are executed sequentially. Mutational operator M_2 reduces the length of the given procedure by removing a segment of randomly chosen length (with a maximum of 5 commands) from the end of the procedure up to the minimum permissible procedure length. M_3 adds a list of randomly generated commands (with a maximum of 5 commands) to the given procedure up to the maximum permissible procedure length. M_1 was applied with a probability 0.5, and M_2 and M_3 with probability 0.25 each. Crossover is at a single common locus that is randomly chosen on the shorter procedure with uniform probability over its entire length.

Selection is fitness proportionate. The initial population consists of randomly generated procedures of lengths ranging from 5 to 60. This population is evolved with a fixed-population size, generational EA with an elitism of 3. 20% of the remaining slots (rounded to the nearest even number) of every successive generation are reserved for genotypes ob-

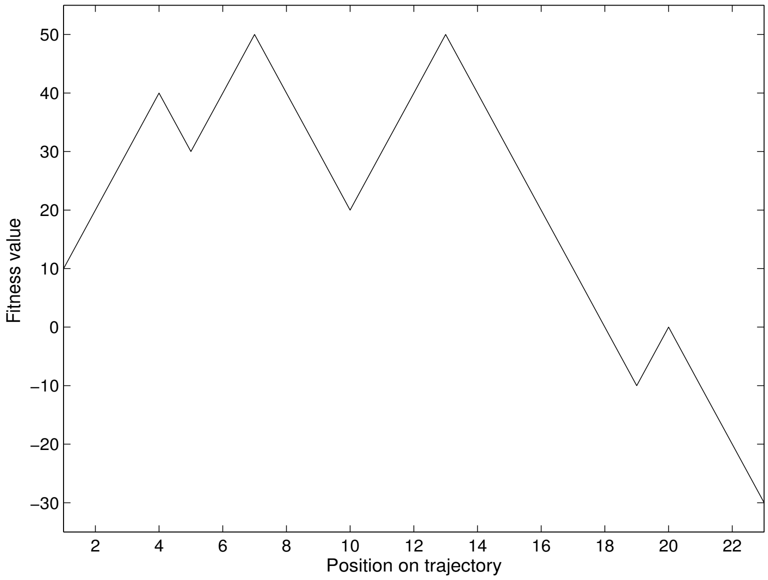


Figure 2.8: Fitness variation along ontogeny for a random genetic procedure on Pattern-1.

tained by crossover, while the remaining slots are filled by mutational variants.

2.4.3 Results

The developmental variability of this Turtle-based generative function is uncorrelated with these pattern-based fitness functions. Figure 2.8 shows the fitness variation for the distinct phenotype on the ontogeny for a randomly generated genetic program `LRBFLLRLFRLFFLFFR RFFLBFLBFBFBLLLLBFBFFLRFRLFL` (where `L = left, R = right, B = back, F = forward`) for the fitness function corresponding to Pattern-1.

The results described below were obtained from running an EA with a population of 50 individuals over 150 generations (over 10 runs) for the randomly generated patterns in Figure 2.7 with the maps $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$. Due to the similarity of the results obtained with both patterns as well as several other randomly generated patterns, our discussion will mainly focus on the representative results obtained with Pattern-1.

As predicted by our analytical model, Figure 2.9 shows an example of the loss of high fitness phenotypes due to selection bias on a single evolutionary run with $\psi_{\theta_{max}}$ with the fitness function being Pattern-1. The plot shows the gap between the maximum fitness of

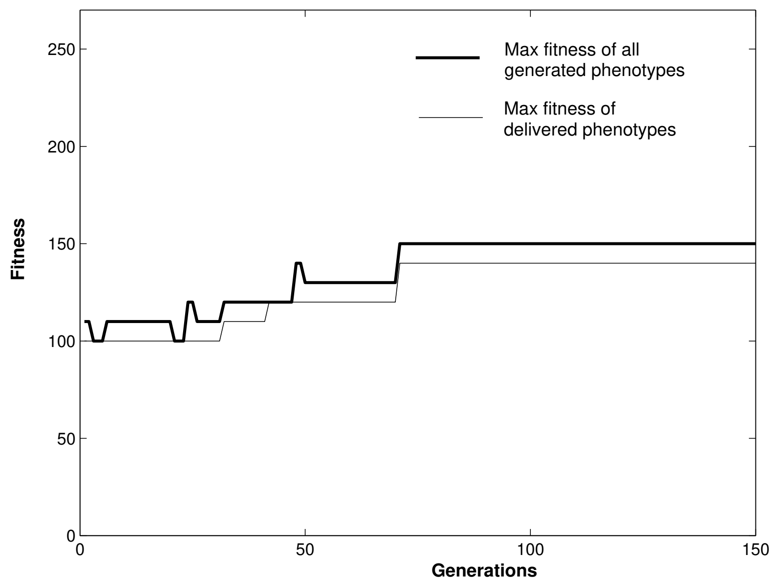


Figure 2.9: Loss of high fitness phenotypes due to selection bias over the entire population (Pattern-1)

all the generated phenotypes in a particular generation and the maximum fitness of the phenotypes that were actually delivered into the population in each generation. There is a distinct time lag between when a high fitness phenotype is first generated at some point on the ontogeny and when a phenotype having a comparable fitness arises in the terminal position of the ontogeny to be delivered into the population.

This is a fundamentally different phenomenon from the Baldwin expediting effect[35] where "development" is used synonymously with life-time learning. In the Baldwinian learning scenario, high fitness phenotypes are discovered by explicit local search in the phenotype space but there is a time lag before this phenotype can be produced directly by a genotype independent of this learning process, i.e. be canalized [74]. In the case of morphological development which has been the subject of our discussion, there is no learning process and the occurrence of these high fitness ontogenetic intermediates is hidden from selection due to the nature of the delivery mechanisms. Rather than expediting adaptation, here the ontogenetic structure seems to inadvertently retard the rate of adaptation in preventing these high fitness phenotype from being discovered and exploited by evolution.

Figure 2.10 shows the frequency of selection "mismatches" when genetic players are

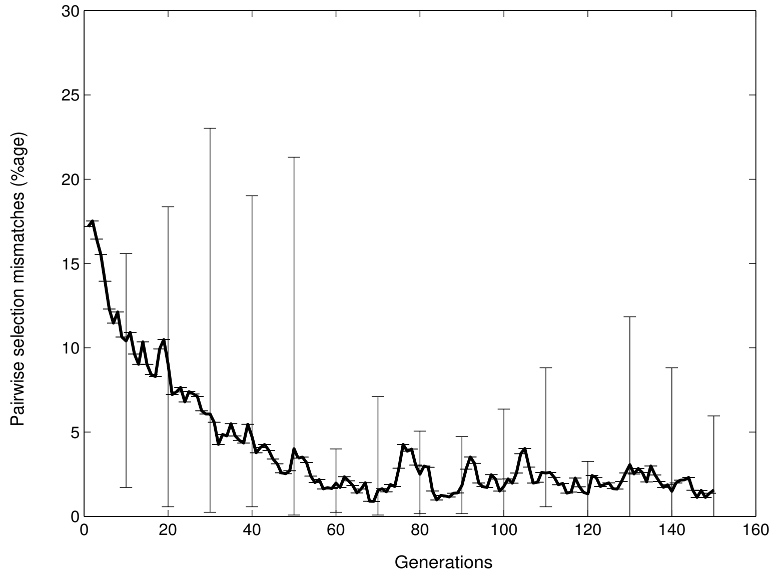


Figure 2.10: Selection mismatches with $\psi_{\theta_{max}}$ on Pattern-1. The error bars indicate the maximum and minimum values obtained over 10 runs.

considered pairwise as described in Section 2.3.4. A "mismatch" is deemed to occur if $f(\psi_{\theta_{max}}(g)) > f(\psi_{\theta_{max}}(g'))$ and $f(\psi_{\theta_{Nash}}(g)) < f(\psi_{\theta_{Nash}}(g'))$ for two genotypes g and g' in the population. This is expressed as a percentage of the total number of pair-wise comparisons of genotypes in the population. With a population of 50, the total number pair-wise comparisons is equal to 1225. The plot in Figure 2.10 shows the frequency (averaged over 10 runs) with which these selection mismatches occur in the population with $\psi_{\theta_{max}}$. Here the average number of such mismatches is approximately 17% (approximately 220 mismatched pairs) in the initial random population which then shows a decreasing trend, remaining at the order of 3% (approx. 60 mismatches) without entirely stabilizing to zero over the 150 generations.

In general, we would expect that the number of such selection mismatches decreases as the fitness of the delivered phenotypes in the population approaches the global optimum. However, this downward trend in the number of mismatches shown in Figure 2.10 is not reflective of such a scenario as the fitness of the delivered phenotypes is well below the high fitness regimes of these fitness functions.

These result establish the occurrence of the predicted selection bias. So, it now presents

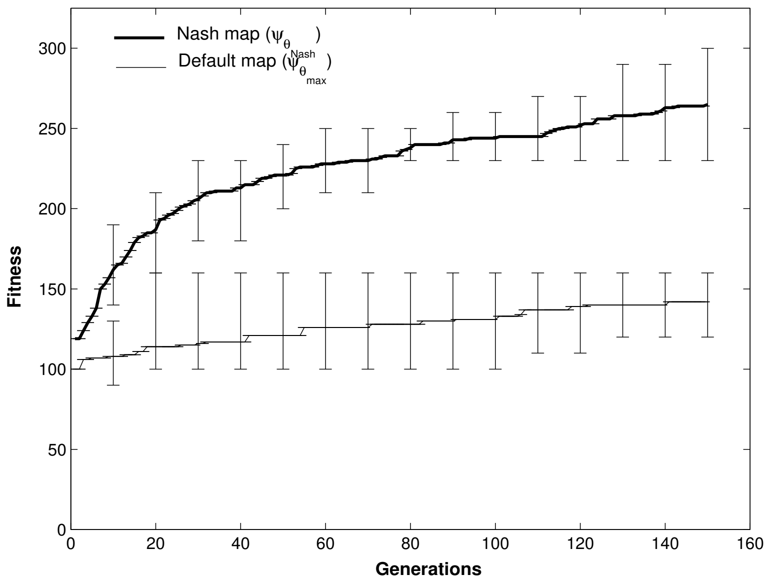


Figure 2.11: Fitness of best delivered individuals for $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ on Pattern-1. The error bars indicate the maximum and minimum values obtained over 10 runs.

the question of the effect on evolutionary performance of having *unbiased* selection. Figure 2.11 and Figure 2.12 reveal that this difference is significant. Figure 2.11 is a comparison of the change in fitness values of the best evaluated phenotypes with $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ for Pattern-1, and Figure 2.12 shows the same comparison for Pattern-2. In both cases, it can be unequivocally seen that the rate of fitness increase as well as the absolute fitness of the best individual after 150 generations with $\psi_{\theta_{max}}$ is considerably lower than that obtained using $\psi_{\theta_{Nash}}$.

To provide a further contrast, consider the equivalent of the selection mismatches that occur in the population with $\psi_{\theta_{Nash}}$ as shown in Figure 2.13. With $\psi_{\theta_{Nash}}$, these mismatches do *not* have any consequences for selection as the maximum fitness phenotype in each ontogeny is delivered into the population. Even so, this plot is instructive as it shows that the number of such mismatches consistently remains at a high level (at about 10% or 122 mismatches).

This suggests that the search strategy in each case is very different specific to the characteristics of the genetic representation and variation operators used here. This is also noticeable in the significant difference in the rate of increase in the number of distinct phe-

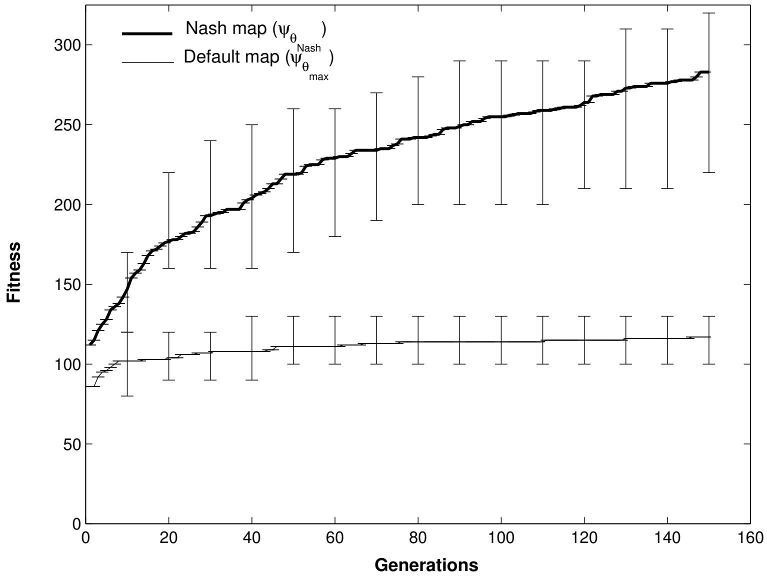


Figure 2.12: Fitness of best evaluated individuals for $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ on Pattern-2. The error bars indicate the maximum and minimum values obtained over 10 runs.

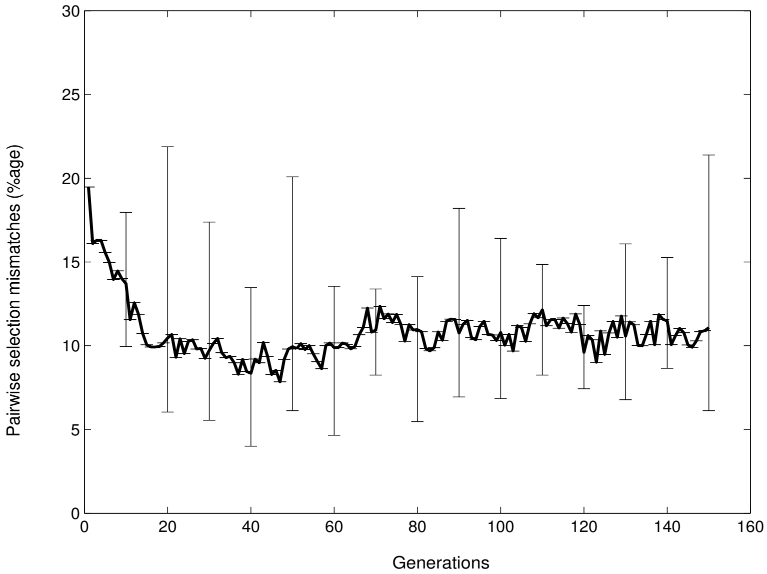


Figure 2.13: Non-selective mismatches with $\psi_{\theta_{Nash}}$ on Pattern-1. The error bars indicate the maximum and minimum values obtained over 10 runs.

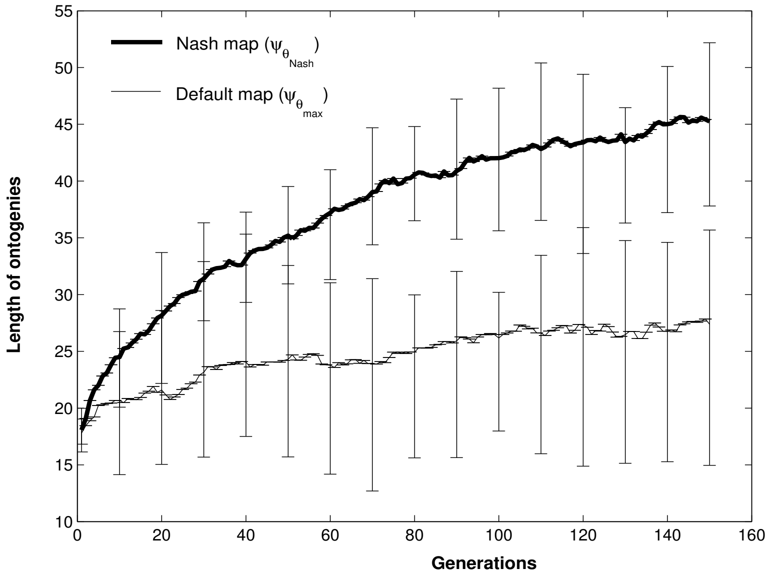


Figure 2.14: Average length of ontogenies ($l(O)$) for $\psi_{\theta_{max}}$ and $\psi_{\theta_{Nash}}$ on Pattern-1. The error bars indicate the maximum and minimum values obtained over 10 runs.

notypes in the ontogenies (i.e. $l(O)$ where $O \in \mathcal{O}$) over evolutionary time as shown in Figure 2.14. The difference suggests that a reason for the low number of selection mismatches with $\psi_{\theta_{max}}$ may be reflective of a population that has low diversity, which reduces the possibility of a mismatch.

2.5 Discussion

One of the main motivations for using indirect genetic encodings in evolutionary problem solving has been the possibility of improving the *evolvability* of the genetic representations by design [4, 75]. From this evolvability perspective, the genetic representation is a rate-limiting factor [77] in that different representations of a given problem can be seen as effecting differences in the rate at which high fitness variants are produced under variation and selection.

To this end, development has largely been treated as a vehicle for an indirect genetic encoding of the search problem. However, in this chapter, we have demonstrated that evolution with an explicit developmental phase can bring some novel issues to bear on evolutionary search different from the genetic issues. By explicitly considering the structure latent in developmental processes, we have shown that this additional structure can have the net effect of *retarding* rather than promoting evolution. Where phenotypic plasticity aided by life-time learning by interaction with the environment leads to an expediting effect on evolution, here the plastic phenotypic transformations occurring under the control of the genotype has the contrary effect of retarding evolution. Analogous to the *Baldwin expediting effect* [35], we can refer to this contrary phenomenon as the *Haeckel retarding effect* based on the classic recapitulationist conflation of the awe-inspiring process of morphogenesis with evolutionary progress [32].

Looking forward, the key conceptual issue that this phenomenon raises is that the genotype needs to be viewed as more than just a recipe for how a phenotype is to be constructed but also as a *strategy for the evaluation of the products of development*. While the strategy of evaluating every phenotype generated was presented as a basic resolution of the problem posed by evaluating only the final phenotype, it is far from being a satisfactory natural resolution. As described earlier, by completely ignoring the inherent structure in the ontogeny, it takes a hammer to the problem by converting it into one of local search. However, this involves the caveats of having to cache the entire trajectory rather than providing a solution that is integrated into the generative character of the development process. Finding a more sound solution is an open question that needs to be resolved.

There are also several empirical issues associated with addressing this problem. The analysis and the demonstration presented here is clearly simplistic and, among other things, does not (a) adequately address the properties of several existing developmental approaches, and (b) says little about the prevalence or the importance of this effect in real-world problems. Furthermore, the critical issue of whether the cost of resolving this issue is commensurate with the gains obtained has remained largely unaddressed here.

Evolution with indirect encodings have already produced numerous successes even without the recognition of this underlying issue. The recognition of this retarding effect therefore suggests an opportunity, rather than a shortcoming, to tap the structure provided by the development processes to further enhance the evolutionary capabilities with such encodings.

Chapter 3

Coevolution and the Ideal Teacher

3.1 Introduction

3.1.1 Background

A Test-Based Problem is a search problem where the suitability of a candidate solution is defined with respect to its behavior on a set of *test cases* [62, 19]. To illustrate the difficulty that they pose, let us compare the canonical combinatorial search problem posed by the TRAVELING SALESMAN PROBLEM (TSP), i.e. a non test-based problem, and the search problem posed by game learning.

An instance of a TSP problem can involve a large and complicated search space but the evaluation of the cost of a tour is efficiently computable. In contrast, even on the simple game of NIM where the problem is to find a first-player strategy that defeats any possible opponent by a domain-knowledge independent process of search, the difficulty posed by evaluating the fitness of a strategy is much of an issue as the size and complexity of the search space. For example, In Rosin's [62] implementation of the game of NIM, for the instance having 4 piles of $3, 4, 5, 4$ stones a strategy was represented using a binary string of length $= 599$. The number of opponents for a given strategy excluding itself is $2^{599} - 1 \approx 2.07 \times 10^{180}$. Here, the computational complexity of using naive exhaustive evaluation grows proportional to the size of the *search space* rather than the size of the *candidate solution* as in the TSP (i.e. the number of cities). This makes the accurate and efficient evaluation of a candidate solution a major practical challenge.

Though such an extreme scaling property may be specific to games [62, 53], this challenge posed by fitness evaluation is characteristic of an important class of problems that includes classification [56, 39, 42, 24], the design of robotic controllers [25, 67] and, in general, the automated generation of programs for complex tasks [34, 48, 67, 22, 69]. The property they share in common is that the quality of a candidate solution is defined in terms of its behavior over a typically large number of discrete test cases (alternatively examples, instances or inputs).

Typically, addressing this situation can require significant domain knowledge to identify a suitable subset of tests to be used as a training set. *Two-population competitive coevolution* [34, 67, 16, 62] is a biologically inspired strategy to address this problem of picking appropriate tests to drive the evolution of candidate solutions. A competitive coevolutionary algorithm (coEA) is an evolutionary problem-solving approach distinguished by the feature that fitness evaluation itself involves another Evolutionary Algorithm. Rather than being an algorithm that is used offline to find suitable tests independent of the main evolutionary algorithm, this second EA operates over the space of test-cases with the intent being to adaptively find subsets of tests that can be used to evaluate the main evolving population of candidate solutions *at run time*. A schematic of the canonical setup of a coEA is shown in Figure 3.1.

This particular commitment of coEAs requires that the members of the test set be encoded with an evolvable representation as well. In requiring tests to have a searchable representation on such problems, coevolution differs from other machine learning approaches that use adaptive testing such as Dynamic Subset Selection used in Genetic Programming [29, 50], boosting [65, 27, 28] and active learning methods [5, 31, 17].

In general, the success of an Evolutionary Algorithm (EA) on a problem is critically linked to how the candidate solutions are represented as genetic data-structures [2, 75]. This raises the question of how the evolvability of the genetic representation plays into the

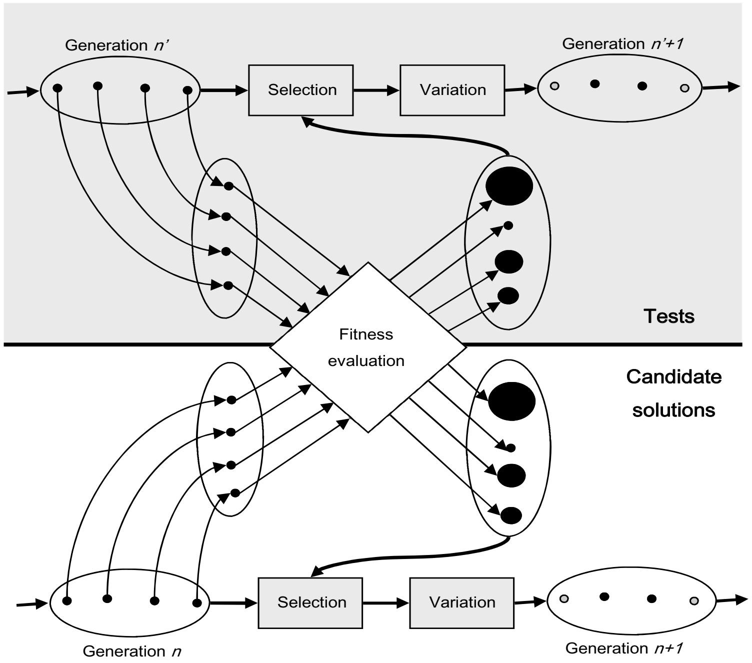


Figure 3.1: Schematic of the classic algorithmic configuration of of a competitive coevolutionary algorithm.

performance of coEAs given that they involve *two* distinct genetic substrates, one encoding the set of candidate solutions for the problem to be solved (i.e. the primary substrate) and the other encoding the set of tests (i.e. the secondary substrate), and hence two independent sources of variation.

As with standard EAs, the performance of a coEA is ultimately determined by whether the quality of the candidate solutions improves over the coevolutionary process. If the associated rate of improvement is to be any better than random search then the correlation between the variability of the primary substrate and the structure of the problem would need to hold here as well [78, 60]. For example, let S be the set of first player strategies for the game of NIM and T be the set of second player strategies, with the goal being to find a first-player strategy s^* in S that beats the most number of second player strategies in T. Independent of other algorithmic concerns, achieving suitable performance on this problem requires a genetic representation of S that has variability properties that are related to the rules of the game as well as the composition of T.

The problem structure posed by such a scenario has been described as having a "multi-objective" structure to it with each member of T being an "objective" to be improved on [41]. This multi-objective perspective has come to serve as an important explanatory concept in the design and analysis of coEAs [76]. In fact, the existence of explicit and formalizable "underlying objectives" to a test-based problem is a central theoretical premise of *Pareto coevolutionary algorithms* [23, 53, 13, 19].

In contrast, the corresponding problem structure governing the evolvability of the secondary representation presents a conundrum. Unlike the well-defined objective function specified on S, the EA operating on the test space T has the dedicated task of "driving" the improvement of the evolving candidate solutions. Ficici [23] notes that this secondary search problem and the associated ambiguity with how this "driving" task is to be technically interpreted is an underappreciated element of coEA design and possibly the source of many of the performance inadequacies of coEAs observed in practice.

As a result of these ambiguities about the relevant problem structure for the EA operating on the test space, little exists in terms of formal models of how the secondary substrate influences the performance of a coEA and the extent of this influence. The goal of this chapter is to take a step towards explicitly addressing this issue.

3.1.2 Approach

An analogy often used to describe the intuition for the operation of a coEA is that of an arms-race between adversaries. This analogy of an arms-race also provides a useful intuition for the relationship between the two substrates. For an arms-race to be possible and sustainable, it requires adversaries that are matched in their capabilities. Only when they are matched can each be repeatedly capable of making a suitable response to the other's challenge and hence remain reciprocally *engaged*. Going beyond this intuition, it has been widely noted that in actual coEAs as well a pre-condition for successful learning is that the two evolving populations remain continually engaged in a coevolutionary arms-race [14, 23, 69]. In this population evolution case, the lowest level determinants of each population's capability to remain engaged by make "matching moves" are their respective genetic representations and corresponding variation operators.

The strategy adopted here is to convert this intuition about "matching moves" into a concrete formal model by posing the question: *given a description of the capabilities of one adversary, can we deduce the capabilities required of the other adversary for a sustainable arms-race to be possible between the two adversaries*? When this capability is defined by the variational properties of each substrate as in a coEA, it translates into the following question: given (a) the objective function defined on the candidate solutions, (b) a description of the substrate encoding the candidate solutions and its associated variation operators, (c) the algorithm used to compute the fitness values of the individuals in the candidate solution population using the members of the test population and (d) the associated selection algorithm, can we deduce the variational properties of substrate encoding the tests as well as the corresponding evaluation and selection algorithms required to evolve the test population so that the two populations would be perfectly "matched" and remain engaged over the duration of the coevolutionary run to result in learning?

We propose a formal solution to the above question when the algorithm operating on the candidate solutions is a mutation-only hillclimber on test-based problems where the outcome of the interaction of a candidate solution and test is binary, i.e. "win" ("correct") or "lose" ("wrong"). The objective function is simply to find the candidate solution that has the most number of "wins" (or "corrects"). The specific focus is on the case where all the candidate solutions are evaluated on all the tests in the test population (i.e. all vs all), and the selection algorithm on the candidate population is based on the Pareto dominance relation.

The basis for the proposed solution is Juillé's *Ideal Trainer* model [41]. Using this model, Juillé originally provided a detailed yet informal description of the ideal *behavior* of the EA operating on the test space (this is described in Section 3.2). Here, we convert this behavioral model into a formal geometric model loosely based on the fitness landscape concept that is explicitly tuned to the peculiarities of coevolution. This is termed the Δ ("Delta") landscape and integrates the ideal behavior of the testing EA with the variational structure of the candidate solution space (Section 3.3). When a hillclimber is assumed to operate on this Δ landscape, the model describes the exact set of tests required in every generation for a perfect matching to exist (Section 3.4). So, this provides a concrete description of the required variational properties of the substrate used to encode the test set and the EA operating on this space such that the two populations would always be matched and exhibit learning that is free from the menagerie of pathological dynamics that are know to plague coEAs [23, 76].

Using this model, we pose the question - is pathology-free engagement a predictor of the ability of a Pareto-coEA to find high quality solutions to the problem? Using the problem posed by designing a coEA to effectively solve a *class* of fitness functions, in Section 3.6, we formally show that on concept learning problems, when the two populations are perfectly matched the test population does not actively "drive" the candidate solution population to regions of the state space having high quality solutions. We empirically demonstrate a consequence of this result, namely, that the ability of two coevolving populations to remain engaged without any pathologies in evaluation and selection can still result in poor overall performance. Furthermore, this suggests that the amount of "knowledge" required to design both substrates to enable perfect matching is insufficient to solve the problem itself, and hence an aspect worth taking seriously for coEAs to be practically useful.

3.2 Rationale

In his dissertation, Juillé [41] proposed an informal yet comprehensive articulation of the ideal behavior of the testing EA. This was framed in terms of the coevolving populations playing the role of a *Learner* and a *Teacher/Trainer* respectively [58, 41, 23, 12, 19], where the teacher poses different tests for the learner (i.e. creates gradient) and the learner attempts to acquire the capability to solve these tests by repeated interactions with them (i.e. follows gradient). Drawing on Epstein's work [21], Juillé described this idealization using the notion of an *Ideal Trainer*.

This is a rare idealization in the coevolutionary literature as it explicitly provides a role for evolvability (referred to as "adaptability") making for a valuable starting point for our discussion. In this section, we provide a brief description of this model and its subsequent modifications that form the main underpinnings of our model.

3.2.1 Assumptions

Before proceeding to Juillé's model, the assumptions that we will adhere to for the rest of this chapter are described below.

Let S be the finite set of candidate solutions and T be the finite set of tests. The interaction between the solutions and the tests is defined by the function $p : S \times T \rightarrow R$, where R is the set of ordered outcomes. We restrict our focus to the binary outcome case where $R = \{0 < 1\}$. The outcomes 0 and 1 correspond to "not solved"(lose) and "solved"(win) respectively. This function p can be represented as shown in Table 3.1. The value in position (i, j) is the outcome of the interaction $p(s_i, t_j)$ between $s_i \in S$ and $t_j \in T$, where $|S| = N$ and $|T| = M$.

The interaction function p is assumed to be noiseless. Furthermore we assume that no two individuals in S have the identical behavior over all the tests, i.e. no two rows are identical. Similarity we assume that no two tests are identical in their behavior over all solutions, i.e. no two columns are identical. The entire row describing the behavior of an individual s_i against all the tests will also be referred to as the interaction profile of s_i.

	t_1	t_2	t_3	t_4	$\cdots$	t_M
s_1	1	1	0	1	$\cdots$	1
s_2	0	0	0	1	$\cdots$	1
s_3	0	1	0	1	$\cdots$	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_N	0	0	1	1	$\cdots$	1

Table 3.1: Matrix representation of p

The objective function of interest here is defined as

$$f_T(s) = \sum_{t_i \in T} p(s, t_i) \tag{3.2.1}$$

for each s in S. So, the objective fitness $f_T(s)$ of an element s in S is equal to the total number of "corrects" or 1s in its interaction profile.

3.2.2 The Ideal Trainer [Teacher] model

Juillé[41] proposed a model of coevolution based on a theory of state-space search. The motivating observation for this model is that continuous progress in search is frequently a result of the search algorithm identifying domains of the state space which correlate better with the operators embedded in the search algorithm, which he broadly termed as *adaptability*. Based on these premises, Juillé proposed that coevolution could be viewed as a strategy that introduces a selection pressure such that individuals that are located in portions of the search space having higher adaptability have a greater evolutionary advantage, i.e. where fewer transformations of the individuals by the search operators are needed to improve their performance. When this occurs, it would imply that the variability characteristics of such individuals have captured some intrinsic properties of the training environment that enables them to react effectively to it.

He then argued that since this adaptive transformation is dependent on the properties of the training environment, adaptability by itself is not enough and what was needed for problem-solving was the *right* kind of adaptability, i.e. the kind that corresponds to increasing competence against tests of "greater difficulty". To this end, he identified two key desirable requirements for an ideal coevolutionary setup:

1. *The need to maintain useful feedback from the training environment*. This is to enable the differences in the adaptability of individuals to become evident during fitness evaluation and selection.

2. *The need for a meta-level strategy to ensure progress in the long-term*. This is to prevent

adaptation that manifests itself as cycling and fixed patterns of exchange, and instead directs coevolution toward increasing progress on the problem.

To address these two requirements, he proposed the following heuristic principle which we will refer to as the "pedagogical principle" (based on the articulation of a similar idea by Rosin and Belew [63], and Pollack and Blair [58]):

> **Pedagogical principle**: *The best way for adaptive agents to learn is to be exposed to [tests] that are just a little more difficult than those they already know how to solve*.

The *Ideal Trainer* is introduced as an abstract entity that can provide such an ideal training environment that realizes the pedagogical principle. To maintain consistency with the broader Machine Learning literature, we will refer to this idealization as the IDEAL TEACHER instead.

An IDEAL TEACHER that can provide such a training environment would need to address the two previously stated requirements by (a) providing tests that are a "little more difficult" than the current capabilities of the learners, and (b) posing tests of "increasing difficulty". He proposed two key methods to realize this IDEAL TEACHER:

1. A distance measure over the tests to formalize the notion of "little more difficult"
2. A mechanism to maintain a partial order over the tests to control the evolution of the tests toward incremental increases in difficulty.

Of the two design issues proposed by Juillé, the first to do with the notion of a "difficulty" measure has so far been a problematic one and has seen subsequent revision.

Difficulty measure

One of the reasons for the interest in coEAs has been the belief that they could reduce the inductive bias and domain knowledge required to engineer the fitness function in standard

EAs [34, 67]. However, the need to define a specialized notion of difficulty involves the use domain-specific knowledge that conflicts with this belief.

In his paper introducing Pareto coevolution, Ficici [24] argued that an explicit notion of difficulty, as proposed by Juillé, was unnecessary. Since the effect of tests being a "little more difficult" was to provide a learnable gradient, all that mattered was to provide the net effect. To this end, Ficici proposed replacing the *variation-based gradient* of Juillé's model associated with adaptability with a *selection gradient*.

For example, if there are two learners A and B in the evolving population, and the training environment provides a test t such that A can correctly solve the test and B cannot then this test reveals a behavior difference (or distinction) between A and B. As selection operates on fitness relevant differences in behavior, such a distinction making test t enables selection to act preferentially on A over B.

Ficici claimed that when this distinction-based selection gradient is combined with a mechanism to maintain a partial order over the *solutions* (rather than the tests), the benefits of the IDEAL TEACHER model could be obtained without the need for any additional domain knowledge about the problem. Consequently, the need to define an explicit notion of difficulty is replaced by the problem of finding distinction making tests for the evolving population of candidate solutions.

Partial order on solutions

This notion of using a partial ordering mechanism on the candidate solutions without an explicit difficulty measure over the tests has since been formalized via the notion of Pareto dominance from Multi-Objective Optimization (MOO) [24, 53]. Using this MOO framework, Bucci and Dejong [12, 19] strengthened the requirement of distinction-making tests to one of tests that can accurately reveal whether a candidate solution s_i is in fact "better" than s_j with respect to f_T.

As the set of outcomes R of the interaction function are ordered, a preference relation $\preceq$ between each $s, s' \in S$ is defined by a pairwise comparison $\leq_{pw}$ of the outcomes on each test [12]. The relation $\preceq$ is defined as being the *Pareto dominance* relation. So $s \prec s'$ implies that s' Pareto-dominates s.

For example, in the tableau shown below, s_1 *dominates* s_2 as it correctly solves all the tests that s_2 solves and at least one more test

$$s_2 \prec s_1 \;=\; \begin{array}{c|cccccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\ \hline s_1 & 1 & 1 & 0 & 1 & 0 & 1 \\ s_2 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

In the following tableau, however, s_3 and s_4 are *non-dominated* or incomparable as s_3 solves at least one test that s_4 doesn't and vice versa

$$s_3 \circ s_4 \;=\; \begin{array}{c|cccccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\ \hline s_3 & 1 & 1 & 1 & 0 & 0 & 1 \\ s_4 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array}$$

The resultant ordering obtained from the complete set of interactions defines a partial order $(S, \preceq)$ on S. So, the maximal elements of $(S, \preceq)$ are the desired solutions to the problem. Accordingly, the coevolutionary search problem is to find the maximal elements of $(S, \preceq)$ defined by the Pareto dominance relation over the set of all interaction outcomes as defined by p. Since there may be several maximal elements, it is important to note here that a solution to the problem may be a subset of S rather than necessarily being a single individual learner.

Summary

Despite being couched in informal and analogy-laden terms, Juillé's model is conceptually valuable in proposing that the ideal properties of the testing EA are intimately related to the variability properties of the primary substrate that encodes the candidate solutions (i.e. the learners).

However, while Juillé's model explicitly treats "adaptability" of the learners as a central element, it is silent on the issue of the "adaptability" of the tests. The learner is treated as an entity bounded by variational constraints but the EA realizing the IDEAL TEACHER is treated as a decision-making entity, namely, one that "picks" or decides on the appropriateness of tests rather than searches for them by random variation and selection. As a result, this model provides no concept of the search performance of a Teaching EA and its ability to (a) find the suitable tests as required by the theoretical model (b) using selection and variation on the secondary representation with (c) a performance better than random search on (d) a particular problem as defined by p. As the problem of interest here is this very aspect, we treat the Teaching algorithm as an explicitly resource bounded and representation dependent entity.

The approach we adopt here is to retain the basic representation-based rationale of Juillé's IDEAL TEACHER concept but instead replace the difficulty measure with the notion of Pareto dominance. All subsequent references to the IDEAL TEACHER are with respect to this modified version. While we retain the objective function f_T that sums the number of "corrects", Pareto dominance is used for fitness evaluation *during* coevolution. In the next section, we propose a geometric formalism to integrate the behavior of the IDEAL TEACHER with the structure of the state-spaces of the candidate solutions and the tests. This formalism is a variant of the fitness landscape that is explicitly tuned to coevolution.

3.3 The Delta landscape

3.3.1 Learnability and test difficulty

As noted earlier, an IDEAL TEACHER can consistently provide tests that are neither too "difficult" nor too "easy" but that are at a level of difficulty that provides a learning gradient that is "just appropriate" to promote the adaptation of the learner based on its current capabilities. In order to address how this notion can be operationalized, we need to define what it means for a test to be "difficult" or "easy" for a particular learner. We interpret this as a difference in terms of the *learnability* of a test.

Ficici and Pollack [24] define the *learnability* of a test with respect to a particular learner as "the probability that the learner can be transformed, *over some number of variation steps*, to become competent (or more competent) at the task posed by the teacher" [emphasis added].

From this definition, we can see that given a learner $s \in S$ and a test $t \in T$ such that $p(s,t) = 0$, the ability of s to learn to solve t is dependent on the variational structure of the *learner space* $\mathcal{S}$. This space is essentially the set S augmented by the topological structure induced by the variational operators particular to the encoding of the members of S.

For simplicity, we restrict our attention to variation with mutation operators. With mutational operators, the topology induced on S can be assumed to take the form of an undirected graph $\mathcal{S} = (S, E)$, where S is the vertex set and E is the set of edges. An edge $e \in E$ exists between s_i and s_j ($s_i, s_j \in S$) if and only if s_i can be obtained by a single application of the mutational operator μ to s_j. We assume here that the effect of the mutation operator is reversible, i.e. if s_i can be obtained from s_2 by a single application of the operator, then the reverse is also possible.

Given this space $\mathcal{S}$, if a learner $s' = \mu^n(s)$ can be obtained by n applications of the mutation operator to s such that $p(s', t) = 1$, then it would follow that t is learnable by s [1]. Critical to this interpretation is the value of n. Here we focus on the case where $n = 1$. Therefore, the learnability of a test t by the learner s is the likelihood that there exists a learner $s' = \mu(s)$ such that $p(s', t) = 1$.

Based on this notion of learnability, the "difficulty" of a test t for a learner s, can be interpreted as follows. A test t is said to be "too difficult" for a learner s, if $p(s,t) = 0$ and $n > 1$ mutations of s are required to produce s' such that $p(s', t) = 1$. However, if $p(\mu(s), t) = 1$ then the test is "appropriate", in being just beyond the present capability of s. On the other hand, if $p(s,t) = 1$ then no variation on s is required to solve the test. Such a test can be considered to be "too easy". The "fitness landscape" corresponding to these three cases is shown in Figure 3.2.

[1] $\mu^n(s)$ is used to indicate n applications of μ to s

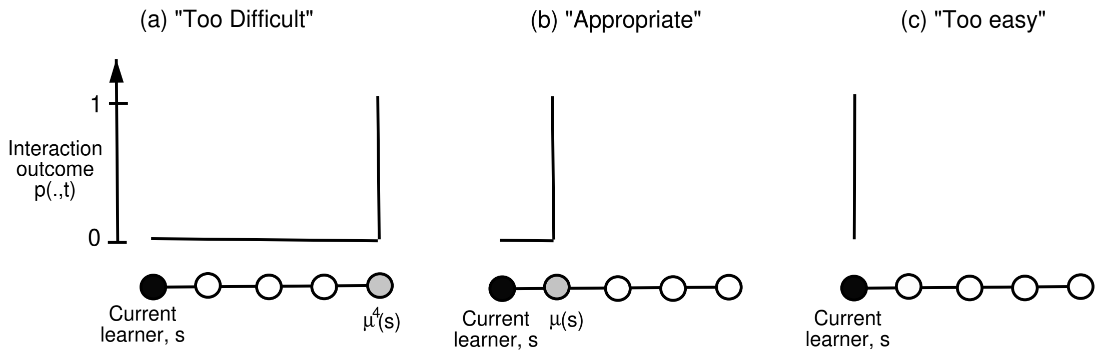


Figure 3.2: Effect of variation on the "difficulty" of learning to solve test t by a learner s

3.3.2 Learnability and Improvement

The above definition of learnability with respect to a single test however requires an amendment in the context of the global search problem. Suppose $s' = \mu(s)$ was such that it indeed solves the test t. This by itself is insufficient to determine whether s' Pareto dominates s, i.e. s' is better or no worse than s on *all* the tests in T.

In order for a teacher to ascertain whether s' is indeed a true improvement over s, the relative performance of the two learners would, in principle, need to be evaluated across *all* the tests in T. Indeed if the teacher could present all the tests to the learner at each instance then there would no demands on the teacher to provide graded challenges to the learner and there would no need for coevolution. Therefore, rather than posing a gradient defined by a single learnable test for s, we would ideally like the teacher to pose a *small* and *sufficient* collection of tests $\Delta \subset T$ such that if learnable by s would indicate an improvement with respect to the global solution concept.

In this regard, Δ may need to contain tests that s *can solve*, in addition to tests that s cannot solve. This is to avoid a situation where a variant s' that solves tests that s cannot solve also "forgets" how to solve the tests that s can solve. For example, consider the scenario in Table 3.2. Let $\Delta = \{t_1, t_2, t_3\}$ be a subset of tests that s cannot solve. The perceived learnability of Δ due to the existence of a variant s' that solves all the tests in Δ is deceptive. Even though s' solves all the tests in Δ, it has "forgotten" how to solve t_5. So, an evaluation of $s \prec s'$ based on Δ alone would be inaccurate in this case.

	t_1	t_2	t_3	t_4	t_5	t_6
s	0	0	0	1	1	1
s'	1	1	1	1	0	1

Table 3.2: Deceptive evaluation due to "forgetting"

It is learnability in this stronger sense that is of particular relevance to the overall goals of coevolutionary search. This brings us to the question – given a learner s in $\mathcal{S}$ what is the set of sufficient tests Δ_s that are learnable by s? The answer to this question follows from the above definitions, as discussed next.

3.4 The Complete Learnable Test set

As $\mathcal{S}$ can be treated as a graph, each learner s is associated with a set $\mathcal{N}_s \subset S$ of all 1-neighbors obtained by a single application of the mutation operator to s. Corresponding to the edge between s and each member of $\mathcal{N}_s$ is a unique test set as described below.

Consider the interaction profile of a learner s given by p_s. Applying a mutation to s produces another learner, say $s' \in \mathcal{N}_s$. If the interaction profiles are such that $p_{s'} \neq p_s$, it implies that s and s' have different behaviors. Let $\Delta_{s,s'}$ be the set of all tests in T such that $\Delta_{s,s'} = \{t|p(s,t) \neq p(s',t), s' \in \mathcal{N}_s, t \in T\}$.

The properties of the tests in $\Delta_{s,s'}$ can be interpreted in a dynamic way. The tests $\Delta_{s,s'} \subseteq T$ are *sensitive* to the variation of s by responding to this change by a change in their outcomes. Since each test $t \in \Delta_{s,s'}$ produces different values corresponding to s and s', each t distinguishes between s and s'. Similarly, the tests in $T - \Delta_{s,s'}$ are *insensitive* as the change of s to s' does not result in a change in their values.

From this perspective, if the change in s to s' is such that only the tests in $\Delta_{s,s'}$ having interaction outcomes of 0 with s change their values to being 1 with s' then it implies that $s \prec s'$. Similarly if this change in the test outcomes is from 1 to 0 then it implies that $s' \prec s$. And finally, if there exist at least two tests in $\Delta_{s,s'}$ such that one changes its outcome from being 0 to 1 and the other from 1 to 0, then s and s' are mutually non-dominated or

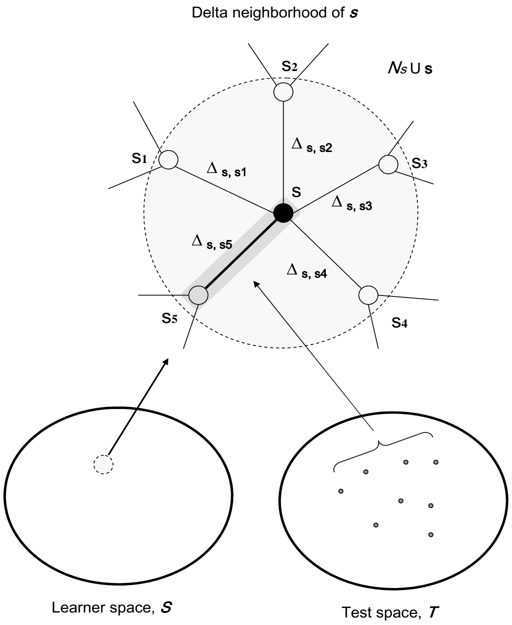


Figure 3.3: Subgraph of $\mathcal{S}$ corresponding to $\mathcal{N}_s \cup \{s\}$

incomparable by $\preceq$.

Such a set Δ_{s,s_i} of "sensitive" tests, with respect to s and s_i, exists for each $s_i \in \mathcal{N}_s$. This set Δ_{s,s_i} can be considered to be an attribute associated with the edge joining s and each s_i as shown in Figure 3.3. The *complete* set of tests which are learnable (and possibly improvable) by s can therefore be obtained as $\Delta_s = \bigcup \Delta_{s,s_i}$. Δ_s is *complete* in that s *cannot learn to or forget how to solve any further test from* T, for the given variational structure $\mathcal{S}$. From this point on we will refer to Δ_s as being the *Complete Learnable Test* (CLT) set for s.

At the outset, we can see that the Complete Learnable Test set has the following characteristics:

- If $s \prec s'$ with respect to Δ_s ($s' \in \mathcal{N}_s$) then $s \prec s'$ with respect to T.
- A test t, where $p(s,t) = 0$, is learnable by s if and only if t is a member of Δ_s.
- Similarly, for every test $t' \in \Delta_s$ where $p(s,t') = 1$, there exists some variant in $\mathcal{N}_s$ that can forget how to solve t'.

It is important to note that Δ_s is not necessarily the minimal set of tests required to *accurately* evaluate the relation between s and its neighbors, if they were simultaneously present. The set $\Delta_{s,s'}$ may contain a number of tests that are redundant in the information that they provide [13]. Furthermore, there may be different non-minimal proper subsets of $\Delta_{s,s'}$ that can perform the same role, i.e. where the relation between s and every $s' \in \mathcal{N}_s$ as evaluated using these test sets is identical to that obtained with Δ_s. It is in this sense that Δ_s is the set of *sufficient* tests for evaluating learning though all the tests are not *necessary* for this purpose.

So, to summarize what we have achieved: Starting from the general intuition about the dynamic behavior of an IDEAL TEACHER, we have arrived at a definition of a specific concept describing the exact properties of the tests generated by the IDEAL TEACHER to achieve this dynamic process of continuous learning. So, when we speak of an IDEAL TEACHER that constructs a learnable gradient for an individual learner, the gradient it provides to the learner takes the form of a Complete Learnable Test set.

In the next section, we describe how such an IDEAL TEACHER can produce the dynamic of continuous improvement.

3.5 Idealized coevolution

To reduce the various complications of coevolutionary dynamics and focus on the low-level properties of the representations, we will focus on local search.

Let $\gamma = \{\Delta_s | s \in \mathcal{S}\}$. This is the set of all the CLT sets corresponding to each of the elements s in $\mathcal{S}$. We can define a topological structure as $\Gamma = (\gamma, E)$, where an edge e exists between Δ_s and $\Delta_{s'}$ if and only if there exists an edge between s and s' in $\mathcal{S}$, i.e. Γ and $\mathcal{S}$ are isomorphic. The key idea that we propose here is the conception of the IDEAL TEACHER as operating on the structured state space defined by Γ rather than on the test space $\mathcal{T}$. Whenever presented with a learner from $\mathcal{S}$, the meta-problem that the teacher poses to this learner is not a single test but a collection of tests corresponding to a particular member of

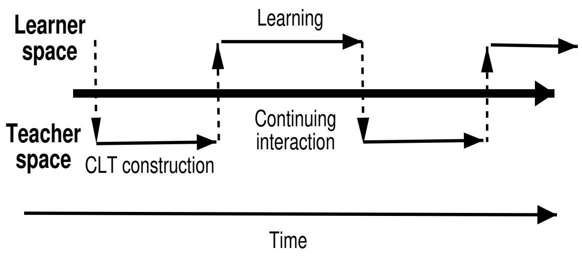


Figure 3.4: Idealized (asynchronous) coevolution with CLT sets

the set γ.

This process can be conceived as taking the form shown in Figure 3.4. Given a learner s, the meta-problem posed by the teacher is the corresponding collection of tests Δ_s. Given the gradient posed by Δ_s, the learner performs a local hillclimbing operation. All the variants of s are generated, and if a variant $s' \in \mathcal{N}_s$ dominates s with respect to the tests in Δ_s then it is selected.

Rather than a synchronous adaptation, at the next iteration when presented with s', the teacher correspondingly performs a local search in Γ using s' as the basis to find the corresponding CLT set for s', i.e. ideally "moving" along the edge from Δ_s to $\Delta_{s'}$. The test set $\Delta_{s'}$ is in turn presented as the learning gradient for s', and so on. In this idealization, the tests posed by the teacher are *always* learnable and the learning that occurs corresponds to progress with respect to the global learning problem.

If this idealized coevolutionary process were realizable in this form, the pathologies of "disengagement" (i.e. loss of gradient), "forgetting" and cycling [76] would be impossible. Even so, one pathology typical of hill-climbers would however be present, namely, of the learners getting stuck on local optima when the learner dominates or is incomparable to all its neighbors. As all the relevant tests are intrinsically contained in each CLT set, there would be no possibility of making a locally inaccurate evaluation to escape the local optimum and the notion of variation opening up a new dimension along which learning could continue [24, 69] would not be meaningful.

So, the overall behavior of such an algorithm can be represented as the ordered sequence

$$O_S = \langle (s_0, \Delta_{s_0}), (s_1, \Delta_{s_1}), \cdots, (s_k, \Delta_{s_k}) \rangle \tag{3.5.1}$$

for the learner algorithm where $s_{i+1} \in \mathcal{N}_{s_i}$ $(s_i \in \mathcal{S})$, and $\Delta_i \subset T, \Delta_i \in \Gamma$. The solution s_k is either the global optimum, or a local optimum where it has no neighbors that dominate it. Similarly for the teaching algorithm,

$$O_T = \langle (\Delta_{s_0}, s_1), (\Delta_{s_1}, s_2), \cdots, (\Delta_{s_{k-1}}, s_k) \rangle \tag{3.5.2}$$

where $\Delta_{s_{i+1}}$ is the neighbor of Δ_{s_i} in the abstract evaluation space Γ.

A candidate and test space pair on which such an algorithm could work will be considered to have matching variability properties.

These definitions are of a black-box coEA that is uncommitted to any problem domain. This by itself is inadequate much more as it provides no notion of why such variability matching should or could be possible. Furthermore, this definition is dependent on the interaction function p, which is problem dependent. So, it presents the question of whether and how sensitive this variability matching property is to uncertainty in the fitness function as with standard EAs.

To explicitly ascertain the nature of this sensitivity, we use a framing similar to Valiant's PAC-learning model [72] for concept learning, where the target concept to be learned is drawn from a class of concepts but where the specific concept that is being used to evaluate the candidate hypotheses is unknown. Since concept learning is a test-based problem to which coEAs have been widely applied [40, 38, 9, 57, 55], here we pose the question of whether variability matching and hence engagement is impacted by this uncertainty in the concept learning domain.

In order to distinguish the black-box model of Pareto coevolution from the domain specific application, we use the typical notation used for concept learning while explicitly retaining their correspondence.

3.6 Concept learning

In this section, we define the basic terms used in the Computational Learning Theory framework and use an example to provide an intuition for the main question to be answered.

3.6.1 Definition

The following definitions are derived from Kearns and Vazirani [44].

Let X be a set called the *instance* set. This corresponds to the set of encodings of the objects in the learner's world or possible inputs.

A *concept* over X is a subset $c \subseteq X$ of the instance space. It can be considered as the set of all instances that positively exemplify some rule over X. This can be represented as a boolean function $c : X \to \{+, -\}$, with $c(x) = +$ indicating that x is a positive example of c and $c(x) = -$ indicating that x is a negative example. So, an instance x in X is an element of c if and only if $c(x) = +$. The elements of the set $\{+, -\}$ are referred to as *labels*.

A *concept class* over X is a collection of concepts $\mathcal{C} \subseteq 2^X$. A *hypothesis* class over X is a collection of concepts $\mathcal{H} \subseteq 2^X$. As such $\mathcal{H}$ need not necessarily be contained in $\mathcal{C}$ but here we assume that $\mathcal{H} = \mathcal{C}$.

In the PAC framework, the general problem takes the following form. The learning algorithm has access to positive and negative examples of an unknown *target concept* c^*, chosen from a known concept class $\mathcal{C}$. The learning algorithm is evaluated by its ability to identify a *hypothesis* concept that can accurately classify instances as positive or negative examples of c^*. The focus is on the scenario where the designer of the learning algorithm is guaranteed that the target concept will be chosen from $\mathcal{C}$ but where the challenge is to design an algorithm and representation that meets the constraints defined by the learning protocol and satisfies the desired performance criteria for *any* target concept in $\mathcal{C}$.

Our interest in adopting this methodological framework to study coEAs is not motivated by an interest in forcing coevolutionary learning into the PAC learning framework but the value of the precise methodology that Computational Learning Theory provides as a way to study coEAs.

The mapping of this concept learning problem to coevolution is as follows: the instance space X corresponds to the set of tests T, the hypothesis space $\mathcal{H}$ corresponds to the learner space $\mathcal{S}$ and the unknown target concept c^* from $\mathcal{C}$ is part of the interaction function p and is denoted as p_{c^*}. The goal is to find a hypothesis in $\mathcal{H}$ that minimizes the error with respect to the unknown target concept c^*, or alternatively maximizes $\sum\limits_{x \in X} p_{c^*}(h, x)$, where the error of a hypothesis h on a target concept c^* is defined as

$$error(h) = \frac{\sum\limits_{x \in X}(p_{c^*}(c^*, x) - p_{c^*}(h, x))}{|X|} = \frac{\sum\limits_{x \in X}(1 - p_{c^*}(h, x))}{|X|} \tag{3.6.1}$$

Let the EA operating on the hypothesis space be denoted as A_H (i.e. the learning algorithm) and that operating on the instance space be A_X (i.e. the teaching algorithm). We assume that neither EA has access to the correct labels associated with each instance. The only information obtained during search is whether the prediction was correct or wrong, or the labels returned by the hypothesis on an instance but not whether it is correct or wrong. In the following example, the learning algorithm only has access to the evaluation of the hypothesis h, where $p_{c^*}(h, x) = 1$ if $c^*(x) = h(x)$ and $p_{c^*}(h, x) = 0$ if $c^*(x) \neq h(x)$.

	x_1	x_2	x_3	x_4	x_5	x_6
c^*	$+$	$+$	$-$	$-$	$+$	$+$
h	$+$	$-$	$+$	$-$	$-$	$-$
$p_{c^*}(h, x)$	1	0	0	1	0	0

The concept class $\mathcal{C}$ is said to be Pareto-*coevolvable* if there exists a Pareto-coevolutionary algorithm $L = \langle A_H, A_X \rangle$ and a pair of genetic representations $\langle \mathcal{H}, \mathcal{X} \rangle$ with the following property: For every concept $c^* \in \mathcal{C}$, if L is given access to $p_{c^*}(h, x)$ then it has a performance defined by the sequence of hypotheses $O_L = \langle h_0, h_1, h_2, ..., h_k \rangle$ such that the terminal solution $error(h_k) < \epsilon$ and $error(h_i) \geq error(h_{i+n})$ $(n > 0)$, where k is the maximum number of generations, h_i is the best hypothesis in generation i, and $0 < \epsilon < 0.5$ is an externally defined tolerance on the acceptable error. Our focus is on the case where L is the algorithm described earlier in Section 3.4. We consider the desired tolerance $\epsilon = 0$, i.e. the target concept has to be exactly learnt.

3.6.2 Example

To provide an intuition for this problem, consider the geometric concept class of d-dimensional axis-parallel rectangles described by Maass and Turan [51] as the BOX_n^d concept class. The primary reason for choosing to focus on this concept class is its intuitiveness both for analysis and for convenient visualization for small dimensions. Despite seemingly being simplistic, we will use this simplicity to draw some general conclusions about coEAs.

For any fixed finite dimension d, the instance set is defined as $X_n^d = \{0, ..., n-1\}^d$. This can be considered to be the equivalent of a discrete grid within the Euclidean space $\mathbb{R}^d$. The concept class is defined as follows:

$$\begin{aligned} BOX_n^d &= \{c \subseteq X_n^d \mid \text{ there is a } d-\text{dimensional axis-parallel rectangle } R \subseteq \mathbb{Z}^d \\ &\quad \text{with } R \cap X_n^d = c\} \\ &= \{\mathop{\times}_{k=1}^{d} \{i_k, ..., j_k\} \mid 0 \leq i_k \leq j_k \leq n-1 \text{ for } k = 1, ..., d\} \end{aligned}$$

An example of a concept belonging to this class containing all the points in $\{3, ..., 6\} \times \{3, ..., 6\}$ for X_n^d where $n = 11$ and $d = 2$ is shown in Figure 3.5. The positive examples associated with c are indicated by the label $+$. In more concrete computational terms, a concept belonging to this class implements a rule that returns a label $c(x) = +$ for a point $x = (a, b) \in X$ if $i_1 \leq a \leq j_1$ and $i_2 \leq b \leq j_2$, and $c(x) = -$ otherwise. In this case $i_1 = 3, j_1 = 6, i_2 = 3, j_2 = 6$.

This concept class consists of all such rectangle concepts in a particular X_n^d. A difference here from the original definition is that the null concept $\{\emptyset\}$ is not a member of the concept class.

To provide an intuition for the nature of the problem, there are $n^d(n+1)^d/2$ different target concepts for a particular class of BOX_n^d. The Figures 3.6, 3.7 and 3.8 show some of the fitness functions for the different target concepts (where the fitness of a hypothesis

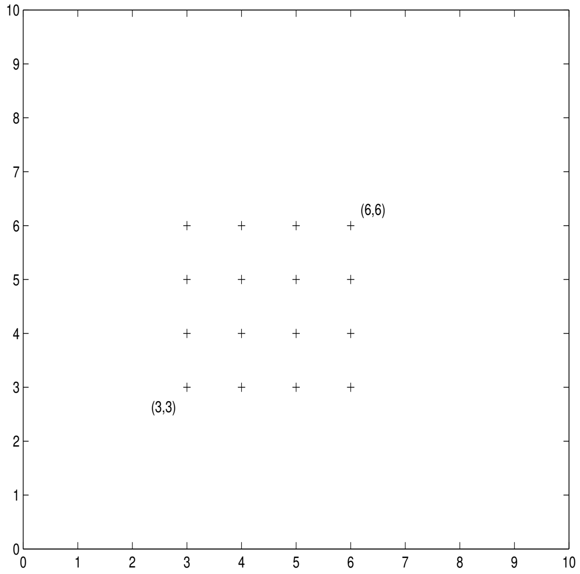


Figure 3.5: The concept defined by $[3, 6] \times [3, 6]$.

is treated as being $1 - error(h)$ for a particular target concept) for the $d = 1$ case and $n = 70$. Each hypothesis is treated as being defined by its endpoints denoted as X_{lower} and X_{upper}. So, each hypothesis in $\mathcal{H}$ can be considered as a single point (X_{lower}, X_{upper}). Since $X_{lower} \leq X_{upper}$ for all the hypothesis, only the portion of the plane above the diagonal is sufficient for this purpose. The target concept in each case corresponding to the point having fitness = 1 is shown with an arrow.

Given this significant difference from one fitness function to another, it suggests that if the target concept c^* is variable then the CLT set Δ_h also variable. If this is so, then variability matching and hence coevolutionary engagement would be effected. In the next section, we prove that this surprisingly is not the case, i.e. Δ_h remains unchanged for any target concept in $\mathcal{C}$.

3.6.3 Invariance of Δ_h

The question we posed earlier can be restated as follows. For a hypothesis h, is Δ_h different when the interaction function is p_{c_i} from the case where p_{c_j}. If this is so then we expected that the ability of A_X operating on X would suffer.

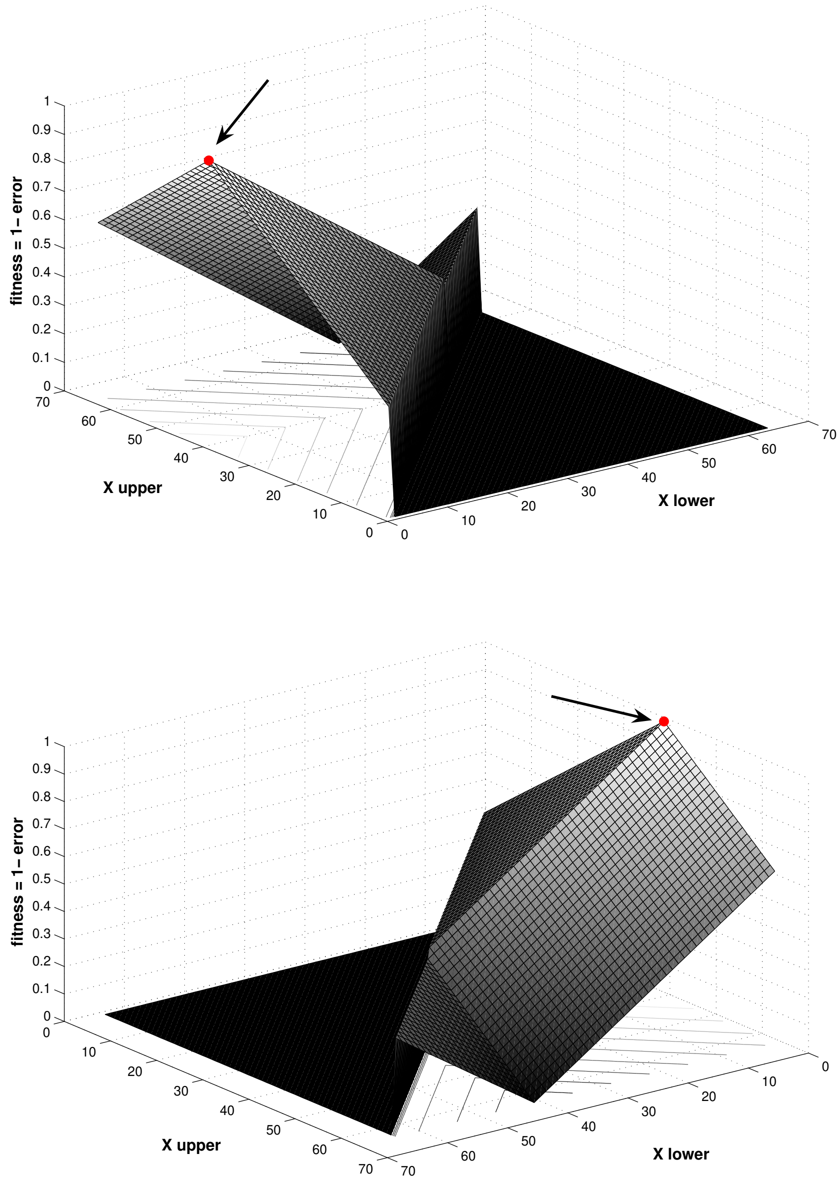


Figure 3.6: The fitness function for target concept at (1,40) viewed from different perspectives.

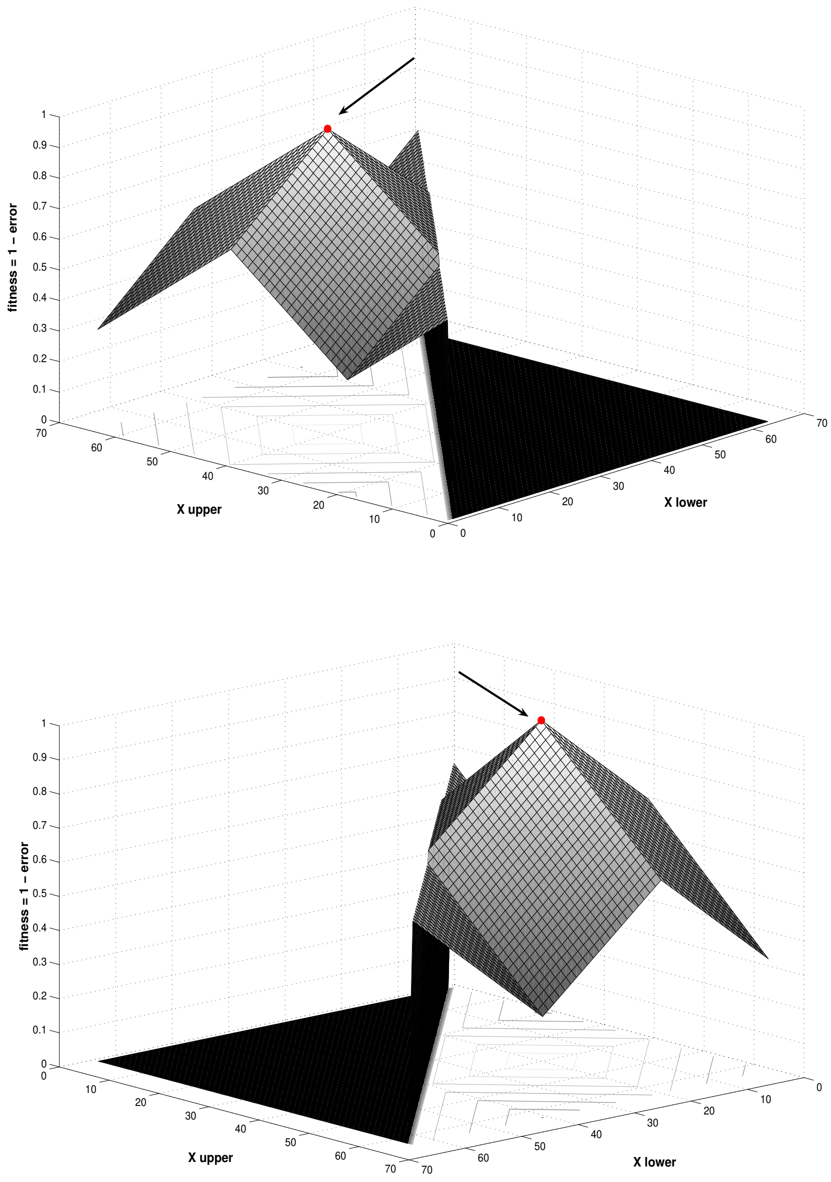


Figure 3.7: The fitness function for target concept at (20,40) viewed from different perspectives.

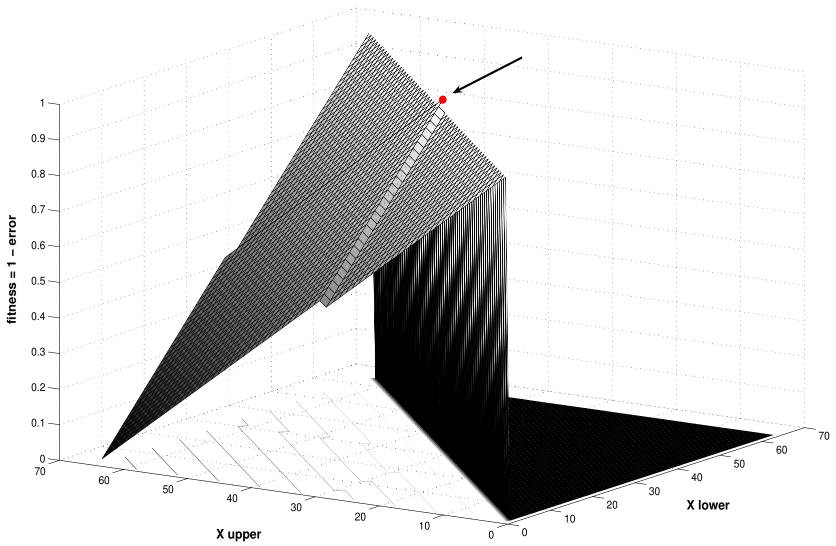


Figure 3.8: The fitness function for target concept at (30,30).

Each hypothesis h is defined by the tests on which it returns a positive label as follows.

Definition 3.6.1. A hypothesis $h \subseteq X$ defined to be the set $h = \{x|\ h(x) = +, x \in X\}$.

Since we use the same representation even when the target concept is unknown, a hypothesis remains the same independent of the target concept to be learnt. Even though it remains the same, the instances for which it returns correct predictions would change with the target concept. To capture this, we introduce the notion of the *evaluation* form of a hypothesis which is the subset of all instances in X that are correctly classified by the hypothesis, i.e. exactly like the target concept.

Definition 3.6.2. The evaluation of a hypothesis h on a target concept $c^* \in \mathcal{C}$ is the set $\overline{h} = \{x|\ p_{c^*}(h, x) = 1, x \in X\}$.

In general, these two are not always identical except in the special case below (proof in Appendix).

Theorem 3.6.3. *A hypothesis $h \in \mathcal{H}$ is equal to its evaluation $\overline{h}$ on a target concept $c^* \in \mathcal{C}$ if and only if $h \subseteq c^*$ and $c^* = X$.*

Similarly, the intersection of $\overline{h}$ and h is empty only in the following special case (proof in Appendix).

Corollary 3.6.4. *The intersection of a hypothesis h and its evaluation $\overline{h}$ on a target concept $c^* \in \mathcal{C}$ is the empty set if and only if $h \cap c^* = \emptyset$.*

In coevolution, we use the instance population to distinguish between the hypotheses in the population as well as evaluate whether one is "better" than the other. For this evaluation to be accurate, any arbitrary set of tests will not do. The *jury* is defined to be a set of tests that provides an accurate evaluation. The term "jury" is deliberately an extension of the notion of a witness set [43] which is the set of instances that can distinguish a hypothesis from every other hypothesis, however the witness makes no value judgment but the "jury" does.

Definition 3.6.5. A subset $j \subseteq X$ is a *jury* for two hypotheses h_i and h_j $(h_i, h_j \in \mathcal{H})$ on a target concept $c^* \in \mathcal{C}$ if $error(h_i) - error(h_j) = k \sum_{x \in j} (p_{c^*}(h_j, x) - p_{c^*}(h_i, x))$, where $k = 1/|X|$.

Definition 3.6.6. The *minimal* jury for two hypothesis $h_i, h_j \in \mathcal{H}$ is the smallest set $J \subseteq X$ that can serve as a jury for the two hypothesis on a target concept $c^* \in \mathcal{C}$.

A concrete example of a minimal jury is the CLT set $\Delta_{h,h'}$, which is exactly those instances in X on which h and h' differ.

The following two theorems will show that no matter what the target concept, the minimal jury for two hypothesis remains unchanged.

Proposition 3.6.7. *The minimal jury $J \subseteq X$ for two hypotheses $h_i, h_j \in \mathcal{H}$ on a target concept $c^* \in \mathcal{C}$ is equal to the symmetric difference [2] of h_i and h_j.*

Proof. An instance $x \in X$ is in the minimal jury J_{h_i,h_j} only if $p_{c^*}(h_i, x) \neq p_{c^*}(h_j, x)$. So, an instance $x \in X - (h_i \cup h_j)$ is not in J_{h_i,h_j} as $h_i(x) = h_j(x) = -$. Similarly, an instance $y \in$

[2] The symmetric difference of two sets A and B is given by $(A \cup B) - (A \cap B)$, or alternatively by $(A - B) \cup (B - A)$.

$h_i \cap h_j$ is not in J_{h_i,h_j} as $h_i(y) = h_j(y) = +$. Therefore $J_{h_i,h_j} = X - ((X - (h_i \cup h_j)) \cup (h_i \cap h_j))$. This can be reduced further as follows

$$\begin{aligned} J_{h_i,h_j} &= X - ((X - (h_i \cup h_j)) \cup (h_i \cap h_j)) \\ &= (X - (X - (h_i \cup h_j)) \cap (X - (h_i \cap h_j)) \\ &= ((X \cap (h_i \cup h_j)) \cup (X - X)) \cap (X - (h_i \cap h_j)) \\ &= (h_i \cup h_j) \cap (X - (h_i \cap h_j) \\ &= X \cap ((h_i \cup h_j) - (h_i \cap h_j)) \\ &= (h_i \cup h_j) - (h_i \cap h_j) \end{aligned}$$

Therefore, the minimal jury for two hypotheses $h_i, h_j \in \mathcal{H}$ is equal to their symmetric difference. □

Now, black-box coEAs do not access the labels and only use the feedback of "correct" and "wrong". So, the following corollary to Theorem 3.6.7 says that the minimal jury is identical even when using only the evaluated form.

Corollary 3.6.8. *The minimal jury $J \subseteq X$ for two hypotheses $h_i, h_j \in \mathcal{H}$ on a target concept c^* is equal to the symmetric difference of their evaluations $\overline{h}_i$ and $\overline{h}_j$ on c^*.*

Proof. Let $A \blacktriangle B$ represent the symmetric difference relation between two sets A and B. So, the assertion to be proved here is that $J_{h_i,h_j} = \overline{h}_i \blacktriangle \overline{h}_j$.

The error of a hypothesis h on a target concept c^* is equivalent to $error(h) = error(h) - error(c^*)$. So J_{h,c^*} is the minimal jury for h and c^*. As $\overline{h}$ is the subset of instances in X where $h(x) = c^*(x)$ $(x \in X)$, it can be written as $\overline{h} = X - J_{h,c^*}$. So $\overline{h}_i \blacktriangle \overline{h}_j = (X - h_i \blacktriangle c^*) \blacktriangle (X - h_j \blacktriangle c^*)$.

To reduce this expression, we use the following two general identities described here with respects to some sets A, B and C,

1. $(A - B) \blacktriangle (A - C) = B \blacktriangle C$.

2. $(A \blacktriangle C) \blacktriangle (B \blacktriangle C) = A \blacktriangle B$.

Using these identities

$$\begin{aligned}
\overline{h}_i \blacktriangle \overline{h}_j &= (X - h_i \blacktriangle c^*) \blacktriangle (X - h_j \blacktriangle c^*) & \\
&= (h_i \blacktriangle c^*) \blacktriangle (h_j \blacktriangle c^*) & \text{(by identity 1)} \\
&= h_i \blacktriangle h_j & \text{(by identity 2)}
\end{aligned}$$

Now, $J_{h_i,h_j} = h_i \blacktriangle h_j$ from Proposition 3.6.7.

Therefore, $J_{h_i,h_j} = \overline{h}_i \blacktriangle \overline{h}_j$. □

Based on this, we can now state that

Theorem 3.6.9. *Let $\Delta_h^{c_i}$ be the Complete Learnable Test set for a hypothesis h and target concept c_i. The CLT sets $\Delta_h^{c_i} = \Delta_h^{c_j}$ for every target concept $c_i, c_j \in \mathcal{C}$.*

This result suggests that even though the overall performance of the coEA may be effected by different target concepts, the difficulty of engagement is unchanged for any target concept. So for two hypotheses h_i and h_j, the same tests are needed to distinguish them for every target concept, even if h_i may have a lower error than h_j in some cases and vice versa on other cases. Furthermore, it clearly indicates that the absence of problems such as cycling, forgetting and disengagement is only a pre-condition for learning. In the next section, we implement a simple algorithm for the BOX_n^d problem and show that coEAs are not immune to the fundamental constraints of blind search as with standard EAs. The "ridges" on the fitness functions in Figures 3.6 and 3.7 have a definite meaning.

3.7 Learnability

For the sake of easy visualization, we consider the two dimensional case $d = 2$ and $n = 16$. Each hypothesis is encoded using a scalar vector $[x, x', y, y']$ similar to the one dimensional case described earlier, where $1 \leq x, x', y, y' \leq n$ and x and x' are the lower and upper limits of the interval in one dimension and y and y' are the lower and upper limits of the interval on the other dimension. The mutation operators either add $+1$ or -1 to each gene. The instances are also encoded as scalars $[x, y]$, with identical mutation operators.

For every hypothesis $h = [x, x'] \times [y, y']$, the corresponding CLT set $\Delta_h = ([x-1, x'+1] \times [y-1, y'+1]) - ([x+1, x'-1] \times [y+1, y'-1])$. These are all the points contained in the mutant h' that fully contains h but not in the mutant h'' that is fully contained by h. For this pair of representations, the following proposition holds.

Proposition 3.7.1. *For the given mutation operator, if $x \in \Delta_h$ then x has a mutant x' such that $h(x) \neq h(x')$.*

This states very basically that a point x on the edge of a rectangle h has a mutant that is not in h.

This fact enables the encodings of $\mathcal{H}$ and $\mathcal{X}$ as described above to be variationally matched. Based on this we construct a simple hillclimbing algorithm A_H and A_X as described below. These two algorithms operate interactively in alternating learning and teaching sessions.

A_H

Initialize with hypothesis $h = h_0$, and $pop_X = \Delta_{h_0}$.

while(1)

- All the variants of h are generated (pop_H) and evaluated on the test population pop_X using p_{c^*}.
- If no member of pop_H dominates h then exit.
- Else, pick the successor h' in pop_H that (a) Pareto dominates h on pop_X, and (b) if there are multiple individuals in pop_H that dominate h, pick the one that has the least error.
- Set $h \leftarrow h'$
- Call A_X

The teaching algorithm performs the following operations to always find the tests in Δ_h.

A_X

while(1)

(a) Initialize a temporary population pop'_X

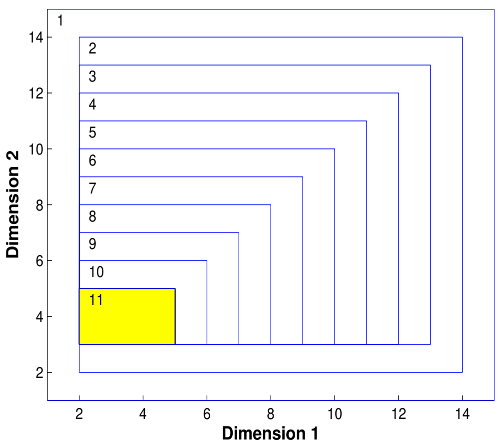
(b) For each test x in pop_X

- Generate all the variants of x and add to tmp_x.
- If there is a test in x' in tmp_x such that $h(x) \neq h(x')$ then add x and x' to pop'_X (without duplication).

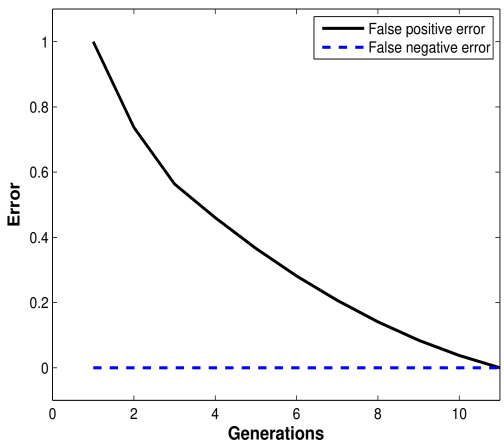
(c) Set $pop_X \leftarrow pop'_X$.

(d) Call A_H.

When the algorithm is initialized with the most general hypothesis, i.e. $[1, n, 1, n]$, the entire concept class BOX_n^2 is learnable. This follows as a direct consequence of Proposition 3.7.1. An example of this is shown in Figure 3.9 with the corresponding change in error, here distinguished into the false-positive and false-negatives. For this initial condition, the change in error is guaranteed to decrease in a monotonic fashion.



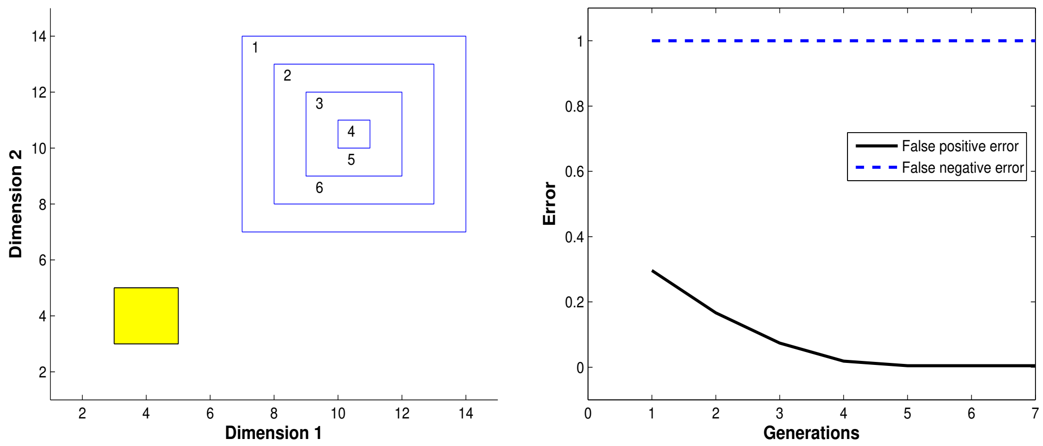
(a)



(b)

Figure 3.9: Trace of algorithm behavior with $h_0 = [1, n, 1, n]$ where the colored rectangle is the unknown target concept, with the corresponding false-positive and false-negative error.

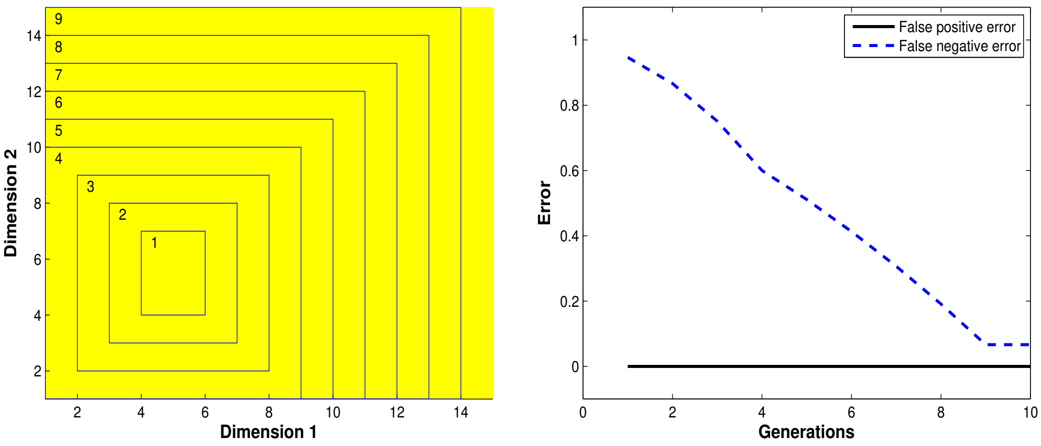
In contrast, compare the behavior of the same algorithm with the initial condition $h_0 = [7, 14, 7, 14]$ as shown in Figure 3.10. Since, this hypothesis does not overlap with c^* (shown as a colored rectangle) the error is entirely of the false positive variety. So, it is dominated by the inner rectangle which has a lower false positive error. This occurs till it collapses to a single point. This is because every hypothesis h such that $h \cap c^* = \emptyset$, $|h| = 1$ and where no hypothesis in the neighborhood of h has a point in c^*, is a local optimum. Since this single point hypothesis dominates all its neighbors that are larger, and is incomparable to all its single point neighbors, it has no neighbor that dominates it. At this point, the only remaining mode of search is by random drift.



(a) **(b)**

Figure 3.10: Trace of algorithm behavior with $h_0 = [7, 14, 7, 14]$ exhibiting a collapse to a local optimum.

This provides a special case of the Red Queen effect [16]. In this case, the error of the hypothesis is technically reducing however it leads to a degenerate or mediocre solution. In the absence of inductive bias about the target concept, we see that a fully engaged coEA can flounder even in the absence of any of the well-known failures. Rather than appealing to the mystery of coevolutionary dynamics, here we see that this occurs due to the use of an imbalanced set of examples to train the learner over the coevolutionary process even though technically the error is decreasing. This provides a principled explanation for this observation that was made empirically by Bongard and Lipson [10], however the solution they used to address this was the introduction of an external memory mechanism (or "test bank").



(a) (b)

Figure 3.11: Trace of algorithm for target concept $c^* = [1, n, 1, n]$

In contrast, when the target concept is $c^* = [1, n, 1, n]$, it can be learnt from any initial condition. An example is shown in Figure 3.11. The reason that this concept class is learnable only from $h_0 = [1, n, 1, n]$ is because, this most-general hypothesis contains every possible target concept and hence cannot collapse to a local optimum.

3.8 Conclusions

Here we proposed a novel representational model of coevolution that specifies the properties required of the secondary substrate to enable perfect matching in the capabilities of the two populations. This model provides a principled way to analyze the determinants of coevolution performance in terms of the variability properties of *both* representations.

In the regime of concept learning, we showed that there exist regularities that on one hand can enable ideal engagement with coevolution even in the presence of uncertainty but in the absence of variational matching would remain problematic for any target concept. The finding of the invariance of Δ_h with the target concept shows that even though the testing algorithm has all the trappings of an EA involving typical selection and variation operators, it is not involved in adaptation in a real sense but serves more as a recognition mechanism for the solutions in the learning population. Whether an EA is adequately suited to perform this function for practical applications, and conversely whether Pareto coevolution is indeed a form of "evolution" presents a semantic question.

This model provides a starting point towards a principled basis for the practical design of representations for coevolution as well as a treatment of representational issues as an integral part of the coevolutionary learning process. Finally, this provides a direct intuition that the variability properties of the secondary substrate and its properties have a key role to play in coevolution.

3.9 Appendix

Theorem 3.9.1. *A hypothesis $h \in \mathcal{H}$ is equal to its evaluation $\overline{h}$ on a target concept $c^* \in \mathcal{C}$ if and only if $h \subseteq c^*$ and $c^* = X$.*

Proof. We prove the assertion in the backward direction prior to the forward direction.

Backward:*If $h \subseteq c^*$ and $c^* = X$ then it implies that $h = \overline{h}$*

By definition 3.6.1, an instance $x \in X$ is in h if and only if $h(x) = +$. Similarly, an instance $y \in X$ is in c^* if and only if $c^*(y) = +$. So, if $h \subseteq c^*$ then it follows that for every instance x in h it must be the case that $h(x) = +$ and $c^*(x) = +$. By definition 3.6.2, an instance $x \in X$ is in $\overline{h}$ if and only if $p_{c^*}(h, x) = 1$. This can be the case only if either $c^*(x) = +$ and $h(x) = +$, or $c^*(x) = -$ and $h(x) = -$. So it follows that if $h \subseteq c^*$ then every instance x in h is also in $\overline{h}$, i.e. h is a subset of $\overline{h}$.

When c^* is a proper subset of X, there necessarily exists an instance $y \in X$ for which $c^*(y) = -$. If $h \subseteq c^*$ then it must also be the case that $h(y) = -$. As $c^*(y) = -$ and $h(y) = -$, from definition 3.6.2 it follows that the instance y is in $\overline{h}$ and from definition 3.6.1 it also follows that y is not in h. Therefore, if c^* is a proper subset of X and $h \subseteq c^*$ then h is a proper subset of $\overline{h}$. By the same reasoning, if c^* is equal to X then there exists no instance y in X for which $c^*(y) = -$. So, if $h \subseteq c^*$ then there is also no instance y in X for which $h(y) = -$. In such a case, an instance $z \in X$ is in $\overline{h}$ if and only if $c^*(z) = +$ and $h(z) = +$, i.e. if z is in h.

Therefore if $h \subseteq c^*$ and $c^* = X$ then it implies that $h = \overline{h}$.

Forward: *If $h = \overline{h}$ then it implies that $h \subseteq c^*$ and $c^* = X$*

By definition 3.6.1, an instance $x \in X$ is in h if and only if $h(x) = +$. So, if $h = \overline{h}$ then it implies that $h(y) = +$ for every instance $y \in \overline{h}$. Now, by definition 3.6.2, an instance $z \in X$ is in $\overline{h}$ if and only if $p_{c^*}(h, z) = 1$. This can be the case only if either $c^*(z) = +$ and $h(z) = +$, or $c^*(z) = -$ and $h(z) = -$. From this definition it follows that if $h(y) = +$ for every instance y in $\overline{h}$ then $c^*(y) = +$ for every such instance as well. As $h = \overline{h}$, it follows that every instance in h is also in c^*. Therefore $h \subseteq c^*$.

If $h = \overline{h}$ then, from the above reasoning, it follows that there exists no instance $x \in X$ in $\overline{h}$ for which $h(x) = -$ and $c^*(x) = -$. So, if there existed an instance $z \in X$ such that $c^*(z) = -$ then it would imply that $h(z) = +$. However, as $h \subseteq c^*$, if $z \in X - c^*$ then it must be the case that $z \in X - h$ that in turn implies that $h(z) = -$. This leads to a contradiction as the hypothesis h cannot return two different labels for the same instance. Therefore such an instance z such that $c^*(z) = -$ cannot exist. Hence if $h = \overline{h}$ then it implies that $X - c^* = \emptyset$, i.e. $c^* = X$.

Therefore if $h = \overline{h}$ then it implies that $h \subseteq c^*$ and $c^* = X$.

So, we can conclude that a hypothesis $h \in \mathcal{H}$ is equal to its evaluation $\overline{h}$ on a target concept $c^* \in \mathcal{C}$ if, and only if, $h \subseteq c^*$ and $c^* = X$.

□

Corollary 3.9.2. *The intersection of a hypothesis h and its evaluation $\overline{h}$ on a target concept $c^* \in \mathcal{C}$ is the empty set if and only if $h \cap c^* = \emptyset$.*

Proof. If $h \cap \overline{h} = \emptyset$ then $h \cap c^* = \emptyset$. An instance x is in h if and only if $h(x) = +$. If such an instance is not in $\overline{h}$ then it implies that $c^*(x) = -$ (by Definition 3.6.2). Since $h \cap \overline{h} = \emptyset$, it implies that $c^*(x) = -$ for every x in h. Since an instance y is in c^* only if $c^*(x) = +$, it follows that no instance in h is in $c^*(x)$. Therefore $h \cap c^* = \emptyset$.

If $h \cap c^* = \emptyset$ *then* $h \cap \overline{h} = \emptyset$. An instance x is in h if and only if $h(x) = +$. If $h \cap c^* = \emptyset$ then it implies that $c^*(x) = -$ for every x in h. Since $h(x) \neq c^*(x)$ for every x in h, these instances are also not in $\overline{h}$ (by Definition 3.6.2). Therefore $h \cap \overline{h} = \emptyset$. $\square$

Chapter 4

Conclusions

4.1 Contributions

In Chapter 1 we singled out the *secondary substrate* as being a distinguishing feature of coEAs and devo-EAs that has evaded a rigorous characterization comparable to the study of the evolvability of genetic representations of candidate solutions. Observing that this additional substrate was associated with mechanisms of change that were independent of the variation operators of the primary genetic substrate, we posed the basic question - does this secondary substrate present its own unique version of the representation problem?

The main contribution of this dissertation is to show that the *secondary substrate* in coEAs and devo-EAs exerts a systematic and structured effect on the performance of both algorithms and suggests the need to complement the standard single substrate genetic models with this consideration when using devo-EAs and coEAs.

One way to empirically answer this question would be to compare the performance using, say, two different secondary substrates A and B on a particular problem while keeping all others variables constant. In this way, any observed difference in performance can be attributed to the choice of secondary substrate, and furthermore, one could even conclude that the substrate associated with better performance is therefore a "better" substrate for the problem being considered. However, such an approach comes with the danger of merely reiterating that some substrates are better than others on a particular problem and that the choice of secondary substrate "matters" to the performance of a coEA or a devo-EA.

Our motivation for adopting a top-down, theory-driven approach to study the secondary substrate problem posed by coEAs and devo-EAs was to seek a substantive answer the questions of *why* and *how* this substrate matters and to do so manner that could advance the quantitative understanding of these otherwise complicated algorithms. To do so, the key emphasis in both cases was on modeling the influence of a particular secondary substrate on the performance across different fitness functions drawn from a problem class. Adopting this approach resulted in the development of a novel analytical framework for devo-EAs and one for co-EAs, both of which are capable of making quantitative predictions about performance effects of the secondary substrate.

To model the effect of the secondary substrate in development, in Chapter 2, we proposed a novel computational model for deterministic development consisting of the generative, decision and delivery functions. This also provided a descriptive model for developmental dynamics expressed as *ontogenies*. This model enabled a principled modeling of computational decisions taken during development by mapping this to a decision-theoretic framing as a multi-player non-cooperative game. This model was used to identify a novel phenomenon associated with devo-EAs, which we termed as the Haeckel effect. Depending on the fitness function, the relationship between the secondary substrate and the decision function can introduce biases in selection that can result in a covert retarding effect on evolutionary performance for different fitness functions. This retarding effect is covert as it isn't detectable using typically used performance measures. The key conceptual issue that this effect raises is that the genotype needs to be viewed as more than just a recipe for how a phenotype is to be constructed but also as a *strategy for the evaluation of the products of development*.

In Chapter 3, we defined a geometric variational model for coEAs called the Δ landscape, using the IDEAL TEACHER model as the basis, to explicitly incorporate the role of both representations into the modeling of performance. The computational model introduced to integrate their behavior was an algorithm based on the concept of *variability matching*. Using this characterization, we demonstrate a very different effect from that arising in devo-EAs. It has typically been assumed that the main reason for the failure of coEAs is attributable to the use of a dynamic rather than a fixed fitness function. Using a secondary substrate that ensures that no such distortions in fitness evaluation and selection occur with dynamic fitness evaluation on a standard class of classification problems, here we show that it is still possible for a coEA to perform poorly. Rather than being a pathology attributable to the dynamic peculiarities of a coEA, this poor performance is due to typical limitations associated with any greedy, blind-search algorithms with inadequate inductive bias. This provides a first demonstration that the absence of pathologies only implies that the coEA satisfies the baseline requirement to exhibit learning on a particular problem and does not imply high quality performance on the problem.

4.2 Synthesis

Despite seemingly being disparate techniques intended to address very different concerns, a key observation arising from the work presented here is that when we take a large step back and consider the as specialized interpretations of the basic EA, both devo-EAs and coEAs share some fundamental similarities both at the micro-scale of their implementations and at the abstract macro-scale of the general formalisms of EAs. The value of this comparison is not as a metaphor or to claim that one is really the other. Instead, the deliberately weak analogy that we draw between coEAs and devo-EAs is to provide a novel way of looking at these algorithms.

In mechanistic terms both coEAs and devo-EAs involve:

1. **An additional representation (and associated computational mechanisms of change) distinct from the genetic representation**:

   (a) In a coEA, this is the data-structure used to represent the members of the test set and its associated variation operators that induce an adaptively searchable test space.

   (b) In a devo-EA, it is the data-structure used to represent the phenotypes and the associated operators used by the interpreter to construct the phenotype based on the rules specified by the genotype.

2. **An independent algorithm defined on the secondary representation distinct from the EA acting on the primary genetic representation**:

   (a) In a coEA, this is the EA operating on the test space.

   (b) In a devo-EA, this is the algorithm that defines how each genotype in the genotype set $\mathcal{G}$ is to be interpreted as a procedure to construct a phenotype.

3. **A protocol for the interaction between the secondary computational process and the primary evolutionary process**:

   (a) In a coEA, this is the protocol that determines how the individuals in the test population are to be used to evaluate the current population of candidate solutions and how the corresponding fitness values are computed and assigned to the individuals in both populations based on the outcome of these interactions. This protocol has traditionally been considered to the heart of a coEA and has been the dominant focus of research on coEAs.

   (b) In a devo-EA, this is the protocol that determines which of the many states occurring during the construction process is to be treated as the "phenotype" associated with a specific genotype.

In juxtaposing these two otherwise unrelated algorithms, firstly, it suggests the value of applying the information processing perspective, that has achieved significant sophistication in coevolution research, to the study of development as well.

A first key issue to see that the use of development converts the problem of fitness maximization to a test-based problem, where intermediate states in the ontogeny serve as "tests" providing information about the problem. By this we do not imply the need for incorporating learning in development but to bring greater attention to the problems of developmental control, i.e. how the developmental substrate and the computational capabilities of the genome could result in a performance-relevant control of the developmental process.

Secondly, the problem of progress due to the absence of a fixed fitness function that has dominated coevolution research is of relevance to the interpretation of development. Even if the genotype is a “recipe” for the construction of a phenotype, it does not follow that the actual process of development is associated with any notion of ”progress”.

Conversely, our analysis suggests the value of the mechanistic perspective that has dominated developmental research to coevolution, namely, treating coEAs explicitly as mechanical procedures for state-space search without invoking misleading analogies that attribute greater capabilities to the two interacting EAs than they necessarily possess. Furthermore, it suggests the value of viewing coevolution from a more domain centered perspective, i.e. where different problem domains require coEAs to have different capabilities, rather than treating games, concept learning, and function optimization as being equivalent problems for a black-box coEA.

4.3 Conclusions

Thomas English [20] in his analysis of the implications of the No Free Lunch theorems [78] notes that algorithms are akin to tools and contain information about the nature of the problems they are good for. In the empirical study of optimizers, the objective is to determine the properties of the task using the information *in* the tool. However, in EC, the tool is often made and buried, before attempting to excavate it and trying to explain them.

The same can be said about coEAs and devo-EAs. While the biological analogies inspiring these algorithms are compelling, the unfortunate trend has often been to treat the algorithms as being as opaque as their inspirations. Where biologists borrow analogies and the language of computation to speak of organisms, EC researchers seem to borrow the opaqueness of biological terminology and then re-discover the computational metaphors, even when the tool is designed by them. Indeed, with this approach, the phenomena and dynamics that arise seem puzzling and difficult to understand. However, it begs the question whether this is simply a distraction that brings a misplaced concreteness to the dynamics that do not necessarily require it.

An example is the pathologies in coevolution. In naming them and treating them as units of study, it gives them independent existence which may not be merited resulting in an implicit assumption that the absence of pathologies implies "success". However, we have shown that the absence of pathologies merely indicates that coEAs are legitimately capable of learning. This does not imply "success" as there remain hard problems and coEAs are still blind search algorithms.

Development provides another example of a tool that is made and buried to be re-excavated and puzzled about. In a scathing deconstruction of the "information"-gene metaphor, Oyama [54] criticizes the lack of intellectual hygiene in using computational and informational metaphors in developmental biology. A key dictum she proposes to avoid the slip into the vacuous use of the term "interaction" while talking about such processes is the need for a *parity of reasoning*, i.e. a logical consistency and completeness in talking about such processes. While directed toward biologists, in designing developmental-representations in a Computer Science setting, we ought to be able to exercise this parity of reasoning in a very detailed fashion.

Far from being a criticism of the use of biological analogies and the value of biologically motivated thinking, the power of coEAs and devo-EAs may in fact lie in taking these analogies with utmost seriousness.

Bibliography

[1] H. Abelson and A. diSessa. *A. Turtle Geometry: The Computer as a Medium.for Exploring Mathematics*. MIT Press, Cambridge, MA, 1981.

[2] L. Altenberg. The evolution of evolvability in genetic programming. pages 47–74, 1994.

[3] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 264–270, San Mateo, Calif., 1993. Morgan Kaufmann.

[4] P.J. Angeline. Morphogenic evolutionary computations: Introduction, issues and example. In John R. McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 387–401. MIT Press, March 1995.

[5] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[6] Robert Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*. Pitman: London, 1987.

[7] Thomas Bäck. Evolution strategies: An alternative evolutionary algorithm. In J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, number 1063 in LNCS, pages 3–20. Springer Verlag, 1995.

[8] José L. Balcázar. The complexity of searching implicit graphs. *Artificial Intelligence*, 86(1):171–188, 1996.

[9] Josh Bongard and Hod Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6:1651 – 1678, 2005.

[10] Josh C. Bongard and Hod Lipson. 'managed challenge' alleviates disengagement in coevolutionary system identification. In Hans-Georg Beyer et al., editor, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 1, pages 531–538, Washington DC, USA, 25-29 June 2005. ACM Press.

[11] Josh C. Bongard and Rolf Pfeifer. Evolving complete agents using artificial ontogeny. In *Morpho-functional Machines: The New Species (Designing Embodied Intelligence)*. Springer-Verlag, 2003.

[12] Anthony Bucci and Jordan B. Pollack. A mathematical framework for the study of coevolution. In Kenneth A. De Jong, Riccardo Poli, and Jonathan E. Rowe, editors, *Foundations of Genetic Algorithms 7*, pages 221–236. Morgan Kaufmann, San Francisco, 2003.

[13] Anthony Bucci, Jordan B. Pollack, and E. D. De Jong. Automated Extraction of Problem Structure. In Kalyanmoy Deb et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference 2004 (GECCO 2004)*, pages 501–512, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[14] John Cartlidge and Seth Bullock. Combating coevolutionary disengagement by reducing parasite virulence. *Evolutionary Computation*, 12(2):193–222, 2004.

[15] Kumar Chellapilla and David B. Fogel. Evolving neural networks to play checkers without expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[16] D. Cliff and G. Miller. Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Advances in Artificial Life: Third European Conference on Artificial Life*, number 929 in Lecture Notes in Computer Science, pages 200–218, Berlin, New York, 1995. Springer.

[17] David A. Cohn, Les E. Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[18] R. Dawkins and J. R. Krebs. Arms races between and within species. *Procs of the Royal Society of London*, Series B(205):489 – 511, 1979.

[19] E. De Jong and Jordan B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159 – 192, 2004.

[20] Thomas M. English. Evaluation of evolutionary and genetic optimizers: No free lunch. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proc. of the Fifth Annual Conf. on Evolutionary Programming*, pages 163–169, Cambridge, MA, 1996. MIT Press.

[21] Susan L. Epstein. Toward an ideal trainer. *Machine Learning*, 15(3):251–277, 1994.

[22] S. Ficici and J. B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In C Adami, editor, *Artificial Life VI*, 1998.

[23] Sevan Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Brandeis University, 2004.

[24] Sevan G. Ficici and Jordan B. Pollack. Pareto optimality in coevolutionary learning. In J. Kelemen and P. Sosik, editors, *Advances in Artificial Life: 6th European Conference (ECAL 2001)*. Springer, 2001.

[25] D. Floreano, S. Nolfi, and F. Mondada. Competitive co-evolutionary robotics: From theory to practice. In *From Animals to Animats 4*. MIT Press, 1998.

[26] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[27] Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *COLT*, pages 325–332, 1996.

[28] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[29] C. Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 312–321, Berlin, 1994. Springer. Lecture Notes in Computer Science 866.

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

[31] S. A. Goldman and R. H. Sloan. The power of self-directed learning. *Machine Learning*, 14(3):271–294, 1994.

[32] S. J. Gould. *Ontogeny and phylogeny*. Belknap press, 1985.

[33] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.

[34] D. Hillis. Co-evolving parasites improves simulated evolution as an optimization procedure. In J. Farmer C. Langton, C. Taylor and S. Rasmussen, editors, *Artificial Life II*. Addison-Wesley, Reading, MA, 1991.

[35] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987. reprint in: Adaptive Individuals in Evolving Populations: Models and Algorithms, R. K. Belew and M. Mitchell (eds.), 1996, pp. 447–454, Reading, MA: Addison Wesley.

[36] J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[37] G.S. Hornby and J.B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246, 2002.

[38] H. Juillé and J. B. Pollack. Co-evolving intertwined spirals. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 351–358, 1996.

[39] H. Juillé and J. B. Pollack. Dynamics of co-evolutionary learning. In *From Animals to Animats 4*, pages 526–534. MIT Press, 1996.

[40] H. Juillé and J. B. Pollack. Coevolving the ”ideal trainer”: Discovery of cellular automata rules. In Koza, editor, *Proceedings Third Annual Genetic Programming Conference*, July 1998.

[41] Hugues Juillé. *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm*. PhD thesis, Brandeis University, 1999.

[42] Hugues Juille and Jordan B. Pollack. Coevolving the ideal trainer: Application to the discovery of cellular automata rules. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[43] Stasys Jukna. *Extremal Combinatorics — With Applications in Computer Science*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, Berlin-Heidelberg-New York-Hong Kong-London-Milan-Paris-Tokyo, 2001.

[44] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, 1994.

[45] H. Kitano. Designing neural network using genetic algorithm with graph generation system. *Complex Systems*, 4:461–476, 1990.

[46] Maciej Komosinski and Adam Rotaru-Varga. Comparison of different genotype encodings for simulated three-dimensional agents. *Artificial Life*, 7(4):395–418, 2001.

[47] J. Koza. *Genetic Programming*. MIT Press, Cambridge, 1992.

[48] John R. Koza. Genetic evolution and co-evolution of computer programs. In Christopher Taylor Charles Langton, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume X, pages 603–629. Addison-Wesley, Santa Fe Institute, New Mexico, USA, 1990 1991.

[49] Sanjeev Kumar and Peter J. Bentley. Computational embryology: past, present and future. *Advances in evolutionary computing: theory and applications*, pages 461–477, 2003.

[50] Christian W. G. Lasarczyk, Peter Dittrich, and Wolfgang Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223–242, Summer 2004.

[51] Wolfgang Maass and Gyrgy Turn. Algorithms and lower bounds for on-line learning of geometrical concepts. *Machine Learning*, 14(3):251 –269, 1994.

[52] John Maynard-Smith. *Evolution and the Theory of Games*. Cambridge: Cambridge University Press, 1982.

[53] Jason Noble and Richard A. Watson. Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection. In Lee Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 493–500, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[54] S. Oyama. *The ontogeny of information*. Duke University Press, 2000.

[55] Ludo Pagie and Paulien Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary Computation*, 5(4):401–418, 1998.

[56] Ludo Pagie and Melanie Mitchell. A comparison of evolutionary and coevolutionary search. *International Journal of Computational Intelligence and Applications*, 2(1):53–69, 2002.

[57] Ludo Pagie and Melanie Mitchell. A comparison of evolutionary and coevolutionary search. *International Journal of Computational Intelligence and Applications*, 2(1):53–69, 2002.

[58] J. B. Pollack and A. D. Blair. Coevolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240, 1998.

[59] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[60] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. Van Leeurwen, editor, *Computer Science Today*, pages 275–291. Springer-Verlag, 1995.

[61] C Reynolds. Competition, coevolution, and the game of tag. In *Artificial Life IV*, pages 59–69. MIT Press, 1994.

[62] C. D. Rosin. *Coevolutionary Search Among Adversaries*. PhD thesis, University of California, San Diego, 1997.

[63] C. D. Rosin and R. K. Belew. Methods for competitive co-evolution: finding opponents worth beating. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 373–380. Morgan Kaufman, 1995.

[64] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[65] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[66] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and description length. In L. Spector, E. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 565–570. Morgan Kaufmann, 2001.

[67] K Sims. Evolving 3D morphology and behavior by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 28–39. MIT Press, 1994.

[68] Kenneth O. Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9:93–130, 2003.

[69] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[70] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[71] L. Van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1 – 30, 1973.

[72] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[73] Wouter van Haaften, Michiel Korthals, and Thomas Wren, editors. *Philosophy of development*. Kluwer Academic Publishers, 1997.

[74] C. H. Waddington. *The strategy of the genes*. George Allen & Unwin Ltd., 1957.

[75] G. Wagner and L. Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50:967 – 976, 1996.

[76] Richard. A. Watson and Jordan B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 702–709, 2001.

[77] A. S. Wilkins. *The Evolution of Developmental Pathways*. Sinauer Associates, 2001.

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

[79] David H. Wolpert and William G. Macready. Coevolutionary free lunches. *IEEE Trans. Evolutionary Computation*, 9(6):721–735, 2005.