Coevolutionary Principles

Elena Popovici, Anthony Bucci, R. Paul Wiegand and Edwin D. de Jong

Abstract Coevolutionary algorithms explore domains in which no single evaluation function is present or known. Instead, algorithms rely on the aggregation of outcomes from interactions among coevolving entities to make selection decisions. Given the lack of an explicit yardstick, understanding the dynamics of coevolutionary algorithms, judging whether a given algorithm is progressing, and designing effective new algorithms present unique challenges unlike those faced by optimization or evolutionary algorithms. The purpose of this chapter is to provide a foundational understanding of coevolutionary algorithms and to highlight critical theoretical and empirical work done over the last two decades. The chapter outlines the ends and means of coevolutionary algorithms: what they are meant to find, and how they should find it.

Elena Popovici
Icosystem Corporation, 10 Fawcett St., Cambridge, MA USA
e-mail: elena@icosystem.com

Anthony Bucci
Icosystem Corporation, 10 Fawcett St., Cambridge, MA USA
e-mail: anthony@icosystem.com

R. Paul Wiegand
Institute for Simulation & Training, University of Central Florida, Orlando, FL USA
e-mail: wiegand@ist.ucf.edu

Edwin D. de Jong
Institute of Information and Computing Sciences, Utrecht University, The Netherlands
e-mail: dejong@cs.uu.nl

1 Introduction

The inspiration for coevolutionary algorithms (CoEAs) is the same as for traditional evolutionary algorithms (EAs): attempt to harness the Darwinian notions of *heredity* and *survival of the fittest* for simulation or problem-solving purposes. To put it simply, a *representation* is chosen to encode some aspects of potential solutions to a problem into individuals, those individuals are altered during search using genetic-like, *variation* operators such as mutation and crossover, and search is directed by selecting better individuals as determined by a fitness *evaluation*. With any luck the iteration of these steps will eventually lead to high-quality solutions to a problem, if problem-solving is the aim; or to interesting or realistic system behavior.

Usually, EAs begin with a fitness function: a function of the form $f\colon G \to \mathbb{R}$ that assigns a real value to each possible genotype in G. Given such a function, the fitness relationship between any two genotypes $g_1,\ g_2 \in G$ is clear: we compare $f(g_1)$ with $f(g_2)$ to see which is more fit. By contrast, CoEAs do not use such a direct metric of the fitness of individuals. Instead, two individuals are compared on the basis of their outcomes from interactions with other individuals.[1] As should become apparent as the chapter unfolds, this seemingly small change to how evaluation is done creates a variety of fundamental differences from traditional evolutionary computation. Most importantly, the fitness ranking of two individuals can change over time, a phenomenon that cannot happen in an ordinary EA. Indeed, virtually all of the techniques and ideas presented in this chapter are affected one way or another by this fundamental question: how could we possibly build an effective algorithm when any time we believe A is better than B it is possible that later we will believe B is better than A?

This chapter is constructed to provide a foundational understanding of coevolutionary algorithms from a problem-solving point of view. Along the way we will survey some of the many CoEAs that have been developed and the wide variety of domains to which they have been applied. Before going into detail, however, we present two coevolutionary algorithm schemes and problem classes to give context for what follows.

1.1 Two Simple Coevolutionary Algorithms

In a *single population* CoEA (Algorithm 1), individuals are evaluated by interacting with other individuals from that population. In a *multi-population* CoEA (Algorithm 2), individuals in one population interact with individuals in one or several other populations.

[1] For this reason the fitness in a CoEA has been called *subjective fitness*, subject as it is to the changing population(s). The objectively-given and unchanging fitness function used in typical EA applications is *objective*.

Algorithm 1 SINGLEPOPULATIONCOEA

Initialize *population*
Select *evaluators* from *population*
Evaluate *individuals* from *population* by interacting with *evaluators*
while not done **do**
  Select *parents* from *population*
  Produce *children* from *parents* via variation
  Select *evaluators* from $\left(children \overset{+}{,} parents\right)$
  Evaluate *individuals* from *children* by interacting with *evaluators*
  Select survivors for next generation
end while
return solution

Algorithm 2 MULTIPOPULATIONCOEA

for each $pop \in populations$ **do**
  Initialize *pop*
  Select *evaluators* from $(populations - pop)$
  Evaluate *individuals* from *pop* by interacting with *evaluators*
end for
while not done **do**
  for each $pop \in populations$ **do**
    Select *parents* from *pop*
    Produce *children* from *parents* via variation
    Select *evaluators* from $(populations - pop)$
    Evaluate *individuals* from *children* by interacting with *evaluators*
    Select survivors for next generation
  end for
end while
return solution

While many CoEAs are variations of these two main frameworks, we have omitted several details, some of which will be discussed later in the chapter. For example, a common mechanism not addressed by those frameworks is for a CoEA (either single or multi-population) to have an *archive* that serves to evaluate individuals or to store potential solutions and is updated from the main population(s). Also, multi-population CoEAs can perform simultaneous or concurrent evolutionary steps, while our example algorithm is sequential. Finally, we have not yet discussed which interactions occur or how the results of interaction outcomes are aggregated into an evaluation that can be used by a selection method. These points will be addressed in Sect. 2.

1.2 Problem Classes

As a consequence of subjective evaluation, the dynamics of coevolutionary algorithms can be frustratingly complex. One can view the interactions themselves as

the basic unit of evaluation; in that view, the fitness landscape for an individual changes as a function of the content of the population(s). Intuitions about fitness landscapes from evolutionary computation do not easily map to this population-dependent, dynamic landscape situation. Nevertheless, coevolutionary algorithms ideally will leverage this mutability of the fitness landscape, adaptively focusing on relevant areas of a search space. This can be particularly helpful when problem spaces are very large or infinite. Additionally, some coevolutionary algorithms appear natural for domains that contain certain, known structure [85, 99] since search on smaller components in the larger structure can be emphasized. Most usefully, though, coevolution is appropriate when domains have no intrinsic objective measure, what we will call *interactive domains*.

Historically, the terms *cooperative* and *competitive* have been used to classify the domains to which coevolution is often applied. Indeed, game theory provides some guidance for making such distinctions. We argue that while such distinctions are relevant and at times useful for classifying the interactive domain over which an algorithm operates, they have not been appropriate for classifying *problems* or algorithms. The interactive domain, like the fitness function, simply gives values to interactions. The problem specifies what to find and the algorithm finds it. Chess is an interactive domain, finding a chess playing strategy capable of Grand Master rating is a problem. Experience shows that details of the problem definition and algorithm design have much more impact on the overall behavior of a CoEA than details of the interactive domain.

Thus, we primarily divide problems into classes based on what constitutes a solution: *test-based* problems and *compositional* problems. In a test-based problem, we view the interactions as challenges: when A and B interact, we think of B as providing a challenge, or test, of A's capabilities. By contrast, in compositional problems we think of interactions as assemblies: when A and B interact, we think of them as forming a team, and the question is whether they perform well together.

1.3 Successful Applications of Coevolutionary Algorithms

To ground these ideas, let us consider three historically-important applications of coevolutionary algorithms.

1.3.1 Single Population CoEA Applied to a Test-Based Problem

In single population applications to test-based problems, individuals serve two roles: at times they are used as (components of) potential solutions, while at other times they are used as tests to provide evaluation information about other individuals. The problem of finding good game-playing strategies can be viewed as a test-based problem since strategies are tested by playing against other strategies. Chellapilla and Fogel's *Blondie24* is a successful application of a CoEA to such a problem [23]. A

strategy in Blondie24 employs a traditional mimimax algorithm but uses a neural network for board evaluation. Individuals in the CoEA are vectors of weights for a fixed-structure neural network, and are evaluated by playing against other individuals. Points are awarded based on the win / loss / draw record, and fairly standard evolutionary programming methods are used to select players, hence weight vectors, with better records. The CoEA employed for this problem was able to produce a checkers player that is competitive with the best existing human and AI players.

Interactive Domain: Game of checkers; *Test:* Set of playing strategies; *Potential Solution:* Playing strategy; *Problem:* Find a strategy that beats the most opponents.

1.3.2 Two-Population CoEA Applied to a Test-Based Problem

Hillis' coevolution of sorting networks and challenging data sets uses two populations [44]. One population represents sorting networks,[2] while another represents unsorted data sets to test a network's sorting capability. The goal of the algorithm is to produce the smallest network possible that correctly sorts any given data sets, but it does this while simultaneously honing ever more challenging and representative data sets. The technique produces a 61-comparator network, which is just one comparison larger than the at the time smallest-known network for a 16-input problem. A similar, non-coevolutionary technique described in that work was unable to produce networks smaller than 63 comparators.

Interactive Domain: Apply data set to sorting network; *Test:* Collection of data sets; *Potential Solution:* Sorting network; *Problem:* Find smallest, correct network.

1.3.3 Multi-Population CoEA Applied to a Compositional Problem

There are a number of successful applications of CoEAs to compositional problems wherein problems are decomposed in some way and separate EAs are applied in parallel to the components, even though evaluation must involve some kind of aggregation or composition of components from the whole system. Perhaps the oldest, and still among the most successful, of these is Husbands and Mills' work on job-shop scheduling [47]. In this work, individuals encode potential floor plans for managing jobs involving the processing of a particular widget constructed by the shop, and separate populations are used to optimize these process plans for each widget. Fitness, however, includes accounting for shared resources in the shop (time, space, etc.). There is also a population of Arbitrators, agents that resolve conflicts between process plans for different widgets. Resulting schedulers can deal with a great deal of uncertainty on the job shop floor.

While perhaps not as well known as the Blondie24 or sorting network examples, this work provides a simple example of the difference between a cooperative *domain* and compositional *problem*. In this case, the solution to the problem is a collection of

[2] That is, arrangements of compare-and-swap circuits.

floor plans (one for each component) and an Arbitrator, which means the problem is compositional. However, the floor plans compete with one another in the evaluation function because they compete for shared resources, so it is not strictly cooperative.

Interactive Domain: Determination of complete job-shop schedule; *Components:* floor plans for widgets, arbitrators for conflict resolution; *Potential Solution:* set of floor plans for each widget & arbitrators; *Problem:* Find an efficient & robust job-shop schedule.

1.4 Chapter Organization

History has shown that the naïve application of CoEAs to ill-understood domains is as likely to produce confusing and unsatisfactory results as to succeed. Therefore, at least from the perspective of problem-solving, a careful algorithm design process is critical. This chapter is organized to emphasize this process: define the problem, define the relationship between the algorithm and the problem, implement the algorithm in a principled way, and analyze its behavior based on these principles.

The next section establishes clear definitions of domains, problems, and the types of solutions one expects to find. Section 3 describes how problems relate to evaluation and representation choices within a coevolutionary algorithm. We follow by presenting analytical approaches currently available for understanding coevolutionary algorithm design, performance, and behavior. The concluding section provides some broader views of coevolutionary systems and outlines possible future directions of research and application.

2 Problems and Solutions

Coevolutionary algorithms are typically applied to interactive domains. Such domains generally lack an objective function giving a value to each entity. Rather, interactive domains encode the outcomes of interactions between two or more entities; depending on the domain, individual entities or the interaction itself may receive value as a result of an interaction. An algorithm must then decide how to use these outcomes to make decisions about which entities to promote in the next generation and which entities to demote or discard.

In a problem-solving context, an algorithm will need more than just an interactive domain. It will also require some way of deciding which entities in the domain are better than others. Coupled with information like that, an interactive domain becomes a co-search or co-optimization problem.

This section is concerned with detailing interactive domains, co-search, and co-optimization problems in the abstract, surveying some examples from both test-based and compositional problems, and beginning the discussion of what it means to extract a solution from such problems. Here we leave aside all dynamic or algorithmic considerations and focus instead on formalizing static definitions of problem

classes to which CoEAs have been applied. Coevolutionary algorithms and their behavior on these problem classes are discussed in subsequent sections.

2.1 Interactive Domains

The formal notion of interactive domain can be defined as follows:

Definition 1 (Interactive Domain). An *interactive domain* consists of one or more functions, called *metrics*, of the form $p\colon X_1 \times X_2 \times \cdots \times X_n \to R$, where

- each i with $1 \leq i \leq n$ is a *domain role*;
- an element $x \in X_i$ is an *entity* (playing the domain role i);
- each X_i is an *entity set* (for the domain role i);
- a tuple $(x_1, x_2, \ldots, x_n) \in X_1 \times \cdots \times X_n$ is an *interaction*;
- the value $p(x_1, x_2, \ldots, x_n) \in R$ is an *outcome* (of the interaction);
- the ordered set R is the *outcome set*. □

Some remarks about this definition are in order.

The discussion of problem classes in Sect. 1.2 touched on the distinction between cooperative and competitive domains. Here we would like to relate those terms directly to the interactive domain. *Cooperative domains* have n metrics p_i, one for each domain role. The metric p_i is interpreted as giving an outcome to the entities playing role i. In many examples, $p_i = p_j$ for all roles i and j, so that the metrics only differ by which entity receives the outcome (but see [79] for a more nuanced discussion of cooperative domains). *Competitive domains*, by contrast, typically have only two roles. The two corresponding metrics often obey $p_2 = -p_1$, making the domain equivalent to a zero-sum game. Naturally there is an enormous space of interactive domains that do not fall into either the cooperative or competitive categories. Note also that the terms "cooperative coevolution" and "competitive coevolution" refer to coevolutionary algorithms operating on such domains.

While an interactive domain has n entity sets, two or more of these sets may be the same. We will use the word *type* to refer to the sets from which entities are drawn, independently of what domain roles the entities are playing. There are special cases in which all domain roles are played by entities of the same set and, furthermore, the outcomes a particular entity receives do not depend on which role it plays. We will refer to these as *role-symmetric* domains.

Example 1 (Rock-paper-scissors). The game rock-paper-scissors furnishes a particularly simple example of an interactive domain. This game is generally expressed with a payoff matrix:

	rock	*paper*	*scissors*
rock	0	−1	0
paper	1	0	−1
scissors	−1	1	0

To view rock-paper-scissors as an interactive domain, observe

- there are two roles;
- $X_1 = X_2 = \{rock, paper, scissors\}$ are the entity sets, so there is one type;
- the matrix encodes a single metric p, whose value is given to the entity on the line. This domain is role-symmetric; for instance, *rock* receives the same payoff versus *paper* regardless of whether it is playing the first role or the second. □

Example 2. Here is another example coming from an abstract game:

	t_1	t_2	t_3
s_1	1	1	0
s_2	0	1	2
s_3	2	1	1

As an interactive domain, note

- there are two roles;
- $X_1 = \{s_1, s_2, s_3\}$; $X_2 = \{t_1, t_2, t_3\}$ are the entity sets (there are two types);
- the matrix encodes a single metric p. This domain is not role-symmetric. □

For a large majority of interactive domains that have been studied in practice, the outcome set R is a subset of $\mathbb{R}$; however, that requirement is not strictly necessary (see [9] for a discussion of ordered outcome sets that are not subsets of the reals).

Finally, note that this definition of interactive domain closely resembles the notion of game defined in [34]. What we are calling an entity is there called a behavior, reflecting that work's closer focus on agents selecting behaviors. What we are here calling an interaction is called an event, and our notion of outcome corresponds to what Ficici calls a measurement. That work should be consulted for a more detailed treatment of interactive domains.

2.2 Co-Search and Co-Optimization Problems

A function $f\colon X \to \mathbb{R}$ gives values to elements of X. However, more information is needed to define a *problem* that can be solved. For instance, we may wish to find all the elements of X that maximize the value of f. Or, we may wish to find only one such element. Or, we may wish to find the elements that minimize f. Notice that for a fixed function f, there are many, distinct ways to decide what makes a "good" element of X. To give a well-defined search or optimization problem, one must specify not just the function f but also one way for deciding which elements of X are to be found.

By analogy, once we have an interactive domain we have a way of giving one or more values to interactions of entities, but do not yet have a well-defined problem to solve. The purpose of this section is to detail the kinds of problem we can define over interactive domains. We will call these *co-search problems* and *co-optimization problems*. First,

Definition 2 (Co-Search Problem). Given an interactive domain, a *co-search problem* over it consists of:

- a non-empty subset $I \subset (1,\ldots,n)$ of the n domain roles;
- a set $\mathscr{C}$ of *potential solutions* aggregated from entities of the domain roles in I;
- a *solution concept* that specifies a partitioning of the potential solutions into (actual) solutions and non-solutions. We represent this as a subset $\mathscr{S} \subset \mathscr{C}$ of the potential solutions. □

Let us examine the last two components of the definition in more detail.

Potential Solutions: There are two critical points to emphasize:

- In many problems of practical interest, no single entity makes for a sufficient solution. Rock-paper-scissors stands as an intuitive example: none of the entities *rock*, *paper*, or *scissors* is, by itself, a reasonable strategy to play in this game. In such cases, potential solutions are *aggregated* from, meaning built or constructed out of, multiple entities.
- In some problems we would like potential solutions to either lie in or be aggregated from entities in a single domain role, while in other problems, potential solutions should be aggregates of entities from several domain roles. The subset I is used to distinguish cases like these. If $I = (1)$, solutions are aggregated only from entities playing domain role 1, while if $I = (1,2,\ldots,n)$, solutions are aggregated from entities playing all n roles. We could, of course, have an I that is not all of $(1,\ldots,n)$ but contains more than one domain role.

Section 2.3 will give examples of what we mean by saying potential solutions are "aggregated from" entities. While a fully general formalization of this notion is beyond the scope of the present chapter, in typical examples potential solutions are sets of entities, mixtures of (distributions over) entities, tuples of entities, or perhaps combinations of these.[3] The notion of potential solution here is closely related to the idea of a *configuration* in [34], which should be consulted for more detail; [79] discusses the idea of aggregating solutions in more detail as well.

Solution Concepts: While we presented the idea of a solution concept in the abstract, as a subset of $\mathscr{C}$, in reality solutions are identified using the metrics of the interactive domain.[4] That is, it is typical to seek entities or combinations of entities that in some sense maximize the metrics of the interactive domain. Section 2.4 illustrates the point by describing several examples of solution concepts that have arisen in practice.

A *co-optimization problem* is closely related to a co-search problem. However, instead of a solution concept, a co-optimization problem specifies an order on the potential solutions and implicitly requires that solutions be maximal elements of the order. Specifically,

[3] For instance, Nash equilibria are usually pairs of mixtures.

[4] What Ficici calls an *intensional* solution concept [34].

Definition 3 (Co-Optimization Problem). Given an interactive domain, a *co-optimization problem* over it consists of

- a non-empty subset $I \subset (1,\ldots,n)$ of the n domain roles;
- a set $\mathscr{C}$ of *potential solutions* built from entities of the domain roles in I;
- an ordering[5] $\leq$ of $\mathscr{C}$ such that if $c_1, c_2 \in \mathscr{C}$ and $c_1 \leq c_2$, c_2 is interpreted as being no worse a solution than c_1. □

A co-optimization problem can be converted into a co-search problem by defining the solution concept $\mathscr{S} \subset \mathscr{C}$ to be the set of maximal elements of $\leq$. In that sense, co-optimization is a more refined notion than co-search.

As a final bit of terminology, recall that both co-search and co-optimization problems have a subset I of the domain roles that specifies which entities are aggregated into potential solutions. Let us write $\widehat{I}$ for the complement of I in $(1,\ldots,n)$, so, for instance, if $n = 3$ and $I = (1,2)$ then $\widehat{I} = (3)$. Then we can distinguish two *problem roles*:

- If $i \in I$, X_i is a *component role* and $x \in X_i$ are *component entities* or *components*;
- if $i \in \widehat{I}$, X_i is a *test role* and $x \in X_i$ are *test entities* or *tests*.

Note that $\widehat{I}$ may be empty, meaning there is no test role explicitly defined by the problem. However, I cannot be empty by definition, meaning there is always at least one set of entities playing the role of component.

To summarize some of the key concepts relating to entities that have been introduced in this section:

- An interactive domain defines two or more *domain roles* and entities that play each role.
- The notion of type is coarser than domain role: several domain roles might correspond to the same type of entity, but each type corresponds to at least one domain role.
- A co-search or co-optimization problem lies at a conceptually higher level, defined over an interactive domain.
- Co-search and co-optimization problems define two *problem roles*, components and tests.
- At the problem level, the component role corresponds to one or more lower-level domain roles. The test role may correspond to zero or more domain roles.
- Since types sit at the domain role level, the relationship between types and problem roles can be complicated. There may be components of different types, tests of different types, components and tests that both have the same type, etc.

Section 1.2, besides distinguishing cooperative and competitive domains, also mentioned the distinction between compositional and test-based problems. Before elaborating on potential solutions in Sect. 2.3 and solution concepts in Sect. 2.4, let us define and illustrate these two important classes of co-search and co-optimization

[5] Which may be a partial order or even a preorder.

problems. Briefly, compositional and test-based problems correspond to the extremes where $|I| = n$ and $|I| = 1$, respectively. The middle ground between these extremes is virtually unexplored.

2.2.1 Compositional Problems

A *compositional problem* is a co-search or co-optimization problem in which all domain roles are used as components to build solutions. That is, $|I| = n$ and, conversely, $|\widehat{I}| = 0$. The idea is that each entity in each domain role is a component. To build potential solutions one must use entities from each of the domain roles; one does not have a complete potential solution until one has used at least one component from each domain role. An intuitive example is a baseball team: one does not have a baseball team until one has a pitcher, a catcher, outfielders, etc.

Compositional problems have largely been explored under the rubric of cooperative coevolutionary algorithms (CCEAs). Besides theoretical works that study features of algorithm behavior on arbitrary or abstract test problems [50, 49, 51], or empirical work of algorithm dynamics on simple test problems [83, 81], work applying CCEAs to multi-variate function optimization or multi-agent learning have also appeared. In multi-variate function optimization, one treats each input variable as a distinct domain role, so that each entity in a particular domain role is a potential setting for one of the input variables [86, 87, 76]. Multi-agent learning applications have treated each domain role as the space of possible behaviors or actions for agents that form a team performing a task together [78, 74].

2.2.2 Test-Based Problems

A *test-based problem* is a co-search or co-optimization problem in which $|I| = 1$; that is, in which a single domain role contains components, and all other domain roles are tests. Intuitively speaking, the test entities are used to probe or give information about the potential solutions that can be aggregated from the components.

Test-based problems are discussed as such in [32] and analyzed in connection with multi-objective optimization in [31]; however, the idea is implicit in early work on Pareto coevolution [61, 38] and is more explicit in later works such as [27], [25], and [60]. [9] and [11] formally analyze test-based problems from the perspective of order-theory, while [12] and [71] treat compositional problems using ideas developed to study test-based problems. [28] and [52, 7, 4, 103] approach two types of test-based problems not based on Pareto dominance.

In many of these works, the terms *candidate*, *candidate solution*, or *learner* are used to describe what we would describe as a component Likewise, the term *test* is used in those works to denote what we might call entities playing the test role.

2.3 Potential Solutions in Co-Search and Co-Optimization Problems

Section 2.2 defined a co-search problem as one that, given an interactive domain, specifies a solution concept as a subset of a set of *potential solutions* that are built from the interactive domain. This section is intended to detail, by example, several common extant ways of creating potential solutions from entities.

2.3.1 Single Entity

The simplest and most obvious set of potential solutions is a set of entities. Imagine we have an interactive domain with n roles and a co-search problem with $I = (1)$, so that the component role is being played by the entities in domain role 1. Then one set of potential solutions for this problem is the set X_1 of components itself. In this case, the set $\mathscr{C}$ is X_1, so that a solution concept $\mathscr{S}$ gives a subset of X_1.

Seminal work by Hillis [44] and Sims [97] both use the single entities as potential solutions. Hillis' algorithm sought a single sorting network; there was no sense in which two or more sorting networks could be combined. Likewise, Sims' algorithm sought a morphology and behavior for a simulated creature; again, there was no mechanism for combining several creatures into a composite.

Stipulating that single entities serve as solutions is appropriate in certain domains, particularly those involving the design of a complex object, i.e. when no clear way of combining entities together is present; or those in which we expect that a single entity that solves the problem may exist. However, as noted above about rock-paper-scissors, many domains have no single entity that is adequate to solve the problem.

2.3.2 Sets

Consider, for concreteness, that we have an interactive domain with two roles and our co-search problem specifies that $I = (1)$. The entities in X_1 are components, while those in X_2 are tests. In this example, the set $\mathscr{C}$ is the powerset of X_1, $\mathscr{P}(X_1)$. A solution concept gives a subset of $\mathscr{P}(X_1)$, equivalently one or more subsets of X_1, where each subset is a solution.

Set-based notions of potential solution have been studied under the umbrella of Pareto coevolution [38, 61]. Treating each test as an objective to be optimized, the problem-solving goal is to find (an approximation of) the non-dominated front among the components. Therefore a potential solution is a subset of X_1, in other words an element of $\mathscr{C} = \mathscr{P}(X_1)$. The non-dominated front itself is the solution.

When we elaborate more on solution concepts in Sect. 2.4, we will see the situation is more subtle; in fact, a set related to, but not quite, the non-dominated front is desirable. However, we will defer that discussion until then.

2.3.3 Mixtures

Let's maintain the co-search problem of the previous example. However, let's say that a solution is not a subset of X_1, but a probability distribution over X_1. Following Vose [105], denote the set of these by Λ^{X_1}. Then another choice for the set of potential solutions $\mathscr{C}$ is Λ^{X_1}.

Mixture-based solutions are most often discussed in the context of Nash equilibria; the Nash Memory [39] and Parallel Nash Memory [62] are examples of algorithms that treat mixtures as potential solutions.

2.3.4 Compositions in General

Sets and mixtures are both examples of aggregates; that is, potential solutions created by putting together components drawn from one or more of the component roles defined in the problem. Sets and mixtures are particularly simple kinds of aggregates. More complicated compositions are conceivable.

Compositional coevolution in general aims to discover good assemblies. The originally-stated aim of Cooperative Coevolutionary Algorithms, from which the more general notion of compositional coevolution sprang, was to attack the problem of evolving complicated objects by explicitly breaking them into parts, evolving the parts separately, and then assembling the parts into a working whole [87]. The nature of the "coadapted subcomponents" discussed in that work was not strictly defined; hence, the idea is that any subcomponent of any assembly (aggregate) was fair game for the method. Since then, compositional coevolution has been applied to coevolving teams of agents performing a task, for instance, where the composition here is of several different agents collaborating as a team.

In the language developed in this section, agents or coadapted subcomponents are domain entities playing the component role of a compositional problem. A collaboration among entities is an interaction, and the means for evaluating a team or assembly would constitute the metric of the problem. The potential solutions in these cases, then, are the possible tuples over the domain roles; that is, $\mathscr{C} = X_1 \times \ldots \times X_n$.

Though the translation into the notion of a co-search problem is less obvious, another important, non-trivial example of composition found in coevolutionary algorithms are in applications of neuro-evolutionary algorithms. The Neuro-Evolution through Augmenting Topologies (NEAT) algorithm [100] has been applied to coevolve robot controllers in a simulated robot duel [101, 100, 102] as well as players of a variant of the video game Pong [60]. A hallmark of the NEAT algorithm is the separation between components of neural networks, which consist of small assemblies of neurons and associated weighted synapses, and the topology of a complete network, which functions as a blueprint for how assemblies are put together. A second, important feature of NEAT emphasized in these works is the possibility for complexification or elaboration: since neural network topologies are not limited by NEAT, they can grow to arbitrary complexity, elaborating on previously-evolved

behaviors. In contradistinction to typical CCEAs, which pre-specify the size and complexity of compositions, NEAT permits open-ended complexity increase.[6]

Finally, it is worth pointing out that in general game-theoretic analysis, Nash equilibria are pairs of mixtures of pure strategies. In many cases only the first mixture in the pair is used to solve a problem; the other is there for testing purposes. However, should we actually need both mixtures in the pair, the set of potential solutions would be $\Lambda^{X_1} \times \Lambda^{X_2}$ which involves two levels of aggregating: first mixing (over X_1 and X_2), then pairing.

2.4 Solution Concepts

In this section we will detail several solution concepts for both compositional and test-based problems that have been used in coevolutionary algorithms. Recall that a solution concept specifies a subset of the potential solutions, $\mathscr{S} \subset \mathscr{C}$. For each example we will detail the interactive domain, the co-search problem including the space of potential solutions, and how solutions are determined from the metric(s) of the domain. We begin with solution concepts for compositional problems.

2.4.1 Compositional Solution Concepts

Ideal Team

Say we have an interactive domain with n domain roles, entity sets $X_1, X_2, \ldots, X_n$ and one metric $p\colon X_1 \times \ldots \times X_n \to \mathbb{R}$. Say also that we have a co-search problem over this domain with potential solutions drawn from the set of tuples $\mathscr{C} = X_1 \times \ldots \times X_n$, so in fact this is a compositional problem. Observe there is a one-to-one correspondence between a potential solution and an interaction. The ideal team solution concept defines as solutions those potential solutions that maximize the value received from the metric:

$$\mathscr{S} = \left\{ \bar{x} \in \mathscr{C} \;\middle|\; \forall \bar{x'} \in \mathscr{C}. p(\bar{x}) \leq p(\bar{x'}) \Rightarrow p(\bar{x}) = p(\bar{x'}) \right\} \tag{1}$$

where we have used the shorthand $\bar{x}$ for an arbitrary tuple in $\mathscr{C}$. In multi-agent parlance, the teams for which no other teams perform better are ideal teams.

The ideal team is not the only relevant compositional solution concept, and the question has been posed to what extent cooperative coevolution algorithms converge to it [108]. Another solution concept of interest arising from that investigation is *maximizing robustness*. The idea here is that rather than maximizing the outcome of a single potential solution, a solution should be robust in the sense that a "small" change in one of the components results in "small" changes in the composite's value.

[6] And, one would hope, an increase in capability as well.

2.4.2 Test-Based Solution Concepts

Unless otherwise specified, the following examples of solution concepts for test-based problems are appropriate for problems of the following form:

- the interactive domain has two roles with entity sets X_1 and X_2;
- there is a single metric $p\colon X_1 \times X_2 \to \mathbb{R}$;
- the co-search problem specifies X_1 as the only component role and X_2 as the only test role.

What varies in these examples is the set of potential solutions and how the subset of solutions is defined. We will detail these now.

Best Worst Case

Best Worst Case operates over single-entity potential solutions, meaning $\mathscr{C} = X_1$. This solution concept specifies as solutions those components that maximize the minimum possible outcome over interactions with all tests.[7] That is,

$$\mathscr{S} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C}. \min_{t \in X_2} p(x,t) \leq \min_{t \in X_2} p(x',t) \Rightarrow \min_{t \in X_2} p(x,t) = \min_{t \in X_2} p(x',t) \right\} \quad (2)$$

This criterion is appropriate in real world domains where one needs to protect against the worst scenario possible.

Simultaneous Maximization of All Outcomes

Simultaneous Maximization of All Outcomes requires a solution to be a component that maximizes its outcome over all possible tests simultaneously. That is, $\mathscr{C} = X_1$ is a single entity set of potential solutions, and the solution concept is:

$$\mathscr{S} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C}\ \forall t \in X_2. [p(x,t) \leq p(x',t) \Rightarrow p(x,t) = p(x',t)] \right\} \quad (3)$$

This solution concept has a limited application scope, as for many problems there does not exist a single potential solution that simultaneously maximizes the outcome against all possible tests.

Maximization of Expected Utility

Maximization of Expected Utility (MEU) is also relevant when $\mathscr{C} = X_1$. It specifies as solutions those components that maximize the expected score against a randomly selected opponent:

$$\mathscr{S} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C}. \mathbf{E}(p(x,X_2)) \leq \mathbf{E}(p(x',X_2)) \Rightarrow \mathbf{E}(p(x,X_2)) = \mathbf{E}(p(x',X_2)) \right\} \quad (4)$$

[7] This version has also been called *maximin*. The corresponding *minimax* can similarly be defined.

where we are abusing notation and treating X_2 as a uniformly-distributed random variable ranging over the set of tests, so that $\mathbf{E}(p(x,X_2))$ is the expected value of $p(x,t)$ when t is selected uniformly randomly from X_2.

Equation (4) is equivalent to:

$$\mathscr{S} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C}. \sum_{t \in X_2} p(x,t) \leq \sum_{t \in X_2} p(x',t) \Rightarrow \sum_{t \in X_2} p(x,t) = \sum_{t \in X_2} p(x',t) \right\} \quad (5)$$

when all the sums are defined. That is, maximizing expected utility is equivalent to maximizing the sum of outcome values over all tests when that sum is defined for all the components. Thus, MEU essentially assumes that all tests are of equal importance, which limits its generality.

Nash Equilibrium

The Nash Equilibrium solution concept is inspired by game theory [65] and operates over mixtures of components. When the problem is as before, $\mathscr{C}_1 = \Lambda^{X_1}$. However, we will deviate slightly from our previous examples and also consider problems with $I = (1,2)$, so that $\mathscr{C}_2 = \Lambda^{X_1} \times \Lambda^{X_2}$. That is, there are no tests, so in fact $\mathscr{C}_2$ defines a compositional problem. Furthermore, in both cases we will assume there are two metrics, $p_1 \colon X_1 \times X_2 \to \mathbb{R}$ and $p_2 \colon X_1 \times X_2 \to \mathbb{R}$ interpreted as giving outcomes to entities in X_1 and X_2, respectively. In many domains such as those arising from zero sum games, $p_1(x,y) = -p_2(x,y)$ for all $x \in X_1$ and $y \in X_2$.

Let us first treat the case $\mathscr{C}_2$. Let $\alpha \in \Lambda^{X_1}$ and $\beta \in \Lambda^{X_2}$ be two mixtures over X_1 and X_2, respectively. We can write these as formal sums:

$$\alpha = \sum_{x \in X_1} \alpha_x \cdot x \quad (6)$$

and

$$\beta = \sum_{y \in X_2} \beta_y \cdot y \quad (7)$$

where α_x is the probability assigned to the component $x \in X_1$ by the mixture α, and β_y is the probability assigned to the test $y \in X_2$ by β and, since α and β are both distributions, $\sum_{x \in X_1} \alpha_x = 1$ and $\sum_{y \in X_2} \beta_y = 1$. Using this notation, define a function $\mathbf{E}_{p_1} \colon \Lambda^{X_1} \times \Lambda^{X_2} \to \mathbb{R}$ such that for all $(\alpha,\beta) \in \Lambda^{X_1} \times \Lambda^{X_2}$,

$$\mathbf{E}_{p_1}(\alpha,\beta) = \sum_{\substack{x \in X_1 \\ y \in X_2}} \alpha_x \cdot \beta_y \cdot p_1(x,y) \quad (8)$$

$\mathbf{E}_{p_1}(\alpha,\beta)$ is interpreted as giving the expected outcome that the mixture α receives when interacting with β. $\mathbf{E}_{p_2}(\alpha,\beta)$ is defined similarly and gives the expected outcome to β.

A Nash equilibrium is a pair $(\alpha,\beta) \in \Lambda^{X_1} \times \Lambda^{X_2}$ such that neither α nor β can unilaterally change to some other mixture and receive a higher payoff. That is, (α,β) is a Nash equilibrium if the following two conditions hold:

- for all $\alpha' \in \Lambda^{X_1}$, $\mathbf{E}_{p_1}(\alpha,\beta) \leq \mathbf{E}_{p_1}(\alpha',\beta) \Rightarrow \mathbf{E}_{p_1}(\alpha,\beta) = \mathbf{E}_{p_1}(\alpha',\beta)$; and
- for all $\beta' \in \Lambda^{X_2}$, $\mathbf{E}_{p_2}(\alpha,\beta) \leq \mathbf{E}_{p_2}(\alpha,\beta') \Rightarrow \mathbf{E}_{p_2}(\alpha,\beta) = \mathbf{E}_{p_2}(\alpha,\beta')$

The Nash Equilibrium solution concept for problems with $\mathscr{C}_2 = \Lambda^{X_1} \times \Lambda^{X_2}$ is then

$$\mathscr{S}_2 = \left\{ (\alpha,\beta) \in \mathscr{C}_2 \;\middle|\; (\alpha,\beta) \text{ is a Nash equilibrium} \right\} \tag{9}$$

For $\mathscr{C}_1 = \Lambda^{X_1}$, in other words problems in which we only care about the mixture over components in X_1, the Nash Equilibrium solution concept is

$$\mathscr{S}_1 = \pi_1(\mathscr{S}_2) \tag{10}$$

where π_1 projects a pair (α,β) onto the first coordinate, α.

An attractive feature of the Nash equilibrium as a solution concept is that Nash equilibria have certain "security" guarantees: the expected payoff to α in a Nash equilibrium (α,β) can be no lower than $\mathbf{E}_{p_1}(\alpha,\beta)$, regardless of the strategy with which it interacts. In a game like $\{rock, paper, scissors\}$, for example, any individual strategy like *rock* will receive a -1 payoff against some opponent (if *paper* plays against *rock*, for example). The mixture $\frac{1}{3} \cdot rock + \frac{1}{3} \cdot paper + \frac{1}{3} \cdot scissors$, by contrast, will never receive an expected payoff lower than 0 against any opponent because $(\frac{1}{3} \cdot rock + \frac{1}{3} \cdot paper + \frac{1}{3} \cdot scissors,\ \frac{1}{3} \cdot rock + \frac{1}{3} \cdot paper + \frac{1}{3} \cdot scissors)$ is a Nash equilibrium for that game and $\mathbf{E}_{p_1}(\frac{1}{3} \cdot rock + \frac{1}{3} \cdot paper + \frac{1}{3} \cdot scissors,\ \frac{1}{3} \cdot rock + \frac{1}{3} \cdot paper + \frac{1}{3} \cdot scissors) = 0$. Note that this is a kind of worst-case guarantee for the mixture α that is similar in spirit to that sought in the Best Worst Case solution concept; the primary difference is that Best Worst Case seeks single components with such a guarantee, while Nash Equilibrium seeks a mixture of many components.

Pareto Optimal Set

Multi-Objective Optimization extends traditional optimization through the introduction of multiple *objectives*. This may be viewed as the use of a function that is vector-valued rather than scalar. In Pareto-based solution concepts, every possible test is viewed as an objective to be optimized.

Potential solutions for the Pareto Optimal Set solution concept are subsets of X_1; that is, $\mathscr{C} = \mathscr{P}(X_1)$. Here the set of solutions will consist of a single member, the non-dominated front, which is defined

$$\mathscr{F} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C}. \left[\forall t \in X_2. [p(x,t) \leq p(x',t)] \Rightarrow \forall t \in X_2. [p(x,t) = p(x',t)] \right] \right\} \tag{11}$$

We can define an order on X_1, the *Pareto covering* order, as follows:

$$x \preceq x' \qquad \text{if } \forall t \in X_2 . p(x,t) \leq p(x',t) \tag{12}$$

for all x and x' in X_1. This definition essentially says that the outcome of x' against any test is at least as high as that of x. However, $\preceq$ is not a total order on X_1, because it is possible that $p(x,t) < p(x',t)$ while $p(x,t') > p(x',t')$ for two tests $t,\ t' \in X_2$. Furthermore, while $\preceq$ is reflexive and transitive, it need not be a partial order: two distinct components x and x' might receive precisely the same outcomes on all tests, so $x \preceq x'$ and $x' \preceq x$ both hold and anti-symmetry fails to hold.

We can simplify the definition of $\mathscr{F}$ using $\preceq$:

$$\mathscr{F} = \left\{ x \in \mathscr{C} \;\middle|\; \forall x' \in \mathscr{C} . x \prec x' \Rightarrow x' \preceq x \right\} \tag{13}$$

The Pareto Optimal Set solution concept is defined as:

$$\mathscr{S} = \{\mathscr{F}\} \tag{14}$$

Some remarks:

- There is a formal similarity between (3), the Simultaneous Maximization of All Objectives, and the non-dominated front $\mathscr{F}$ defined in (11). They differ only in the placement of the quantifier $\forall t \in X_2$. However, while formally similar these two solution concepts have significantly different interpretations: while (13) shows that $\mathscr{F}$ is the set of *maximal* elements of the order $\preceq$, in fact (3) is the set of *maxima* of that order, the difference being that maxima must be larger than all other components across all tests simultaneously. Thus, the non-dominated front puts weaker constraints on its members than the Simultaneous Maximization of All Objectives, and we would expect the former to be a larger set of components.
- The Pareto Optimal Set solution concept has only one member, $\mathscr{F}$, which is itself a set of components. This definition is chosen for consistency with the other solution concepts; each solution concept is a set whose members are solutions. The non-dominated front forms a single, unique solution to a multi-objective problem. A limitation of the Pareto Optimal Set solution concept is that this set may be very large; while not being covered is a sensible minimum requirement for solutions, for certain problems it may be insufficiently weak, in that it can insufficiently narrow down the set of possible solutions.
- As noted, two components may receive precisely the same outcomes against all possible tests. Thus, while they may be distinct components in the set X_1, as far as performance against the tests in X_2 goes, they are indistinguishable.

Pareto Optimal [Minimal] Equivalence Set

The indistinguishability of some elements of the non-dominated front entails this set may be too large. We can partition this set in equivalence classes by this property, like so. Say that $x \sim x'$ if $x \preceq x' \wedge x' \preceq x$; that is, x and x' are equivalent if they receive

the same outcomes against all tests in X_2. $\sim$ is an equivalence relation on X_1.[8] It is reasonable to suppose that we only need one representative of each equivalence class under $\sim$, and this intuition leads to two more solution concepts.

A Pareto Optimal Equivalence Set is a set containing *at least one* element from each equivalence class of the full Pareto Optimal Set. Note that while there is a single Pareto Optimal Set, there may be a combinatorial number of Pareto Optimal Equivalence sets. A Pareto Optimal Minimal Equivalence Set contains *exactly one* component from each equivalence class of the full Pareto Optimal Set. There may be a combinatorial number of Pareto Optimal Minimal Equivalence sets as well.

These two solution concepts typically define a smaller set of potential solutions than the Pareto Optimal Set. Yet, depending on the characteristics of the problem, such a set can still be very large.

3 Design of Coevolutionary Algorithms

As we have seen, there is a wide spectrum of problems in interactive domains. While throughout the chapter we discuss generic issues that any algorithm targeted at such co-search problems will have to address, in this section we concern ourselves with issues specific to approaching these problems via coevolutionary algorithms. Like with any method, application of coevolutionary algorithms to a problem tends to go more smoothly and have more opportunity for success when algorithm engineers *first* go through the process of formalizing the domain and the problem, and *then* design the algorithm.

For domains that are inherently interactive, the formalizing process involves more rationalization than choosing. However, sometimes we may want to reformulate a non-interactive domain into an interactive one. The domains of problems featuring the ideal team solution concept are often the result of such a reformulation. In such cases, many different choices may exist as to how to decompose elements in the original domain, and they may have different effects on problem-difficulty.

The design process entails finding useful and productive ways of mapping problem particulars into the algorithmic framework, and heuristic search tends to require some means of making qualitative comparisons and decisions in order to guide the path of algorithm. There are principles behind, and effects from these decisions. In this section we discuss the choices available to the algorithm designer, and the biases introduced by these mapping choices are discussed throughout Sect. 4.

Once the engineer identifies the types, domain roles, problem roles and potential solution set, they must determine how a CoEA can be structured to represent them, and encode data structures within the algorithm to instantiate them in a way that is consistent with the problem's solution concept. We consider such decisions to relate to *representation*. Additionally, one must decide how to explore the set of interactions, how outcomes from multiple interaction will be aggregated for selec-

[8] This is a simple, well-known consequence of $\preceq$ being a preorder.

tion purposes, and how interaction information is communicated throughout various parts of the algorithm. We consider such decisions to relate to *evaluation*.

3.1 Representation

Successful design and application of any heuristic depends on a number of key representation decisions for how search knowledge is encoded and manipulated. In any evolutionary system, these choices involve questions about how aspects of potential solutions are represented genotypically, modified by genetic operators and expressed phenotypically.

Coevolution brings additional subtleties to representation issues that are worth special consideration. For one, the relationship between basic terms such as *individuals*, *populations* and *solutions* can be quite complicated in coevolution. Related to these terms are the foundational, domain-centric terms discussed above such as *test* and *component*, and a well-designed CoEA should consider how these problem-based concepts map to representations used by the search system. Additionally, many modern CoEAs make use of archival and memory mechanisms for a variety of purposes, and it is important to understand how they relate to the above notions. We will discuss these notions in turn and provide examples of common/possible mappings from problem to algorithm.

3.1.1 Individuals

In most traditional evolutionary systems, individuals represent potential solutions to some search problem, and the EA manipulates encoded potential solutions by modifying them using variational operators (mutation, crossover) and choosing the better from among them using selection operators (proportionate selection, rank selection, etc.). When optimizing some real-valued function, an individual might be encoded as a vector of real values to be used as arguments for the function, mutations could be accomplished by applying vectors of Gaussian noise, and selection might involve simply choosing those with the highest objective function value. Since the goal of the search is to find an optimal argument vector, one can typically equate potential solution and individual. Even when the EA is searching program spaces, such as in genetic programming, traditional algorithms still make use of individuals who are essentially procedural solutions to the problem.

In coevolution, individuals serve the same mechanistic purpose as in any evolutionary algorithm: they are the fundamental units being manipulated by the search operators themselves. That is, mutation and crossover modify individuals, and selection chooses among them.

Definition 4 (Individual). An *individual* is a representational unit that is subject to selection and variational operators. □

But individuals in a CoEA will not always represent potential solutions to the problem being solved. In coevolution, one must think of individuals in terms of how they relate to types, domain roles and problem roles, *in addition to* traditional issues of how to encode problem aspects. For example, suppose we are developing an AI for controlling an airplane that generalizes over different flight conditions and we have formalized this as an interactive domain as follows: airplane controllers interact with different flight conditions / scenarios, while the problem is test-based where the potential solutions are controllers and the flight scenarios are tests, and the solution concept is maximum expected utility. We might design a CoEA that maintains two kinds of individuals, corresponding to the two types of the domain, namely individuals representing controllers and individuals representing flight scenarios. But we still have to decide how controllers and scenarios will *actually be encoded*, which will involve additional choices. The consideration of how types, domain-roles and problem-roles map to individuals is subtly different from traditional genotype / phenotype decisions.

Most often individuals represent entities from the domain's types. Thus, for a domain with two symmetric roles we may have a single kind of individual, and such individuals might participate in interactions in either of the domain's roles. In general, we will have at least as many kinds of individuals as types in the domain. From a problem-role perspective, individuals might represent components of a potential solution or tests helping judge the quality of potential solutions. [44] provides a less common example, where some of the individuals do not correspond to entities, but to sets of entities, namely those playing the test role. We will revisit the relationship between individual and solution in the subsection dedicated to solutions.

3.1.2 Populations

Just as with traditional EAs, it is useful to think of populations as simply collections of individuals. Most traditional EAs (excepting island models) have only a single, explicit population, while many CoEAs have multiple populations.

But even in traditional methods the notion of a population can be a bit murky when dealing with mechanisms like spatial models, niching, or restrictive mating [46, 98] since stable subsets of individuals in the overall population can arise between which very little genetic material is exchanged. These subsets are sometimes referred to as "populations", and it is interesting to note that many of these are essentially coevolutionary in nature since fitness can be quite subjective (e.g., niching methods using fitness sharing).

Definition 5 (Population). A *population* is:

1. a set (or multiset) of individuals;
2. a subset of individuals isolated from other individuals by some kind of barrier (e.g., inability to interbreed successfully, geographic, etc.). □

Designing effective CoEAs involves decisions about how aspects of a problem map to populations. The main purpose of the populations is to provide an explo-

ration mechanism for the entity sets of the different types in the domain. Thus a CoEA will generally maintain at least one population per type and most often a single population per type.

For example, for an ideal team problem in a domain with a fixed number of asymmetric roles (and therefore types), the typical CoEA will maintain one population for each. For a test-based problem in a domain with two asymmetric roles (and therefore types), the typical CoEA will also maintain one population for each type.

For domains with two symmetric roles (thus a single type), the problem definition itself may not distinguish between the component role and the test role and the algorithm designer can choose to do the same, by maintaining a single population, or to make the distinction explicit by maintaining two populations. A unique case found in [85] features an ideal team problem in a domain with a variable (unbounded) number of symmetric roles (neural networks with variable number of neurons), approached via a CoEA with a dynamic number of populations, all containing individuals of the same type.

Populations may also be used to represent potential solutions, for example when these are aggregates of entities (and therefore individuals) from one type, as is the case with Pareto Optimal solution concepts.[9]

Last but not least, individuals in a population, by taking part in interactions, serve the purpose of evaluating individuals in the same or other populations. This is because while a problem definition may specify only some of the domain roles as contributing to the potential solution set, an algorithm needs some criteria based on which to select and promote individuals for the test roles as well.

3.1.3 Archives

Coevolutionary methods often employ another kind of collection, typically referred to as an *archive*. Archives span generations, and thus can be considered a kind of search memory. The purpose of the archives is to help the populations with, or release them from, some of their multiple duties. Archives can allow algorithm designers to separate exploration from evaluation and/or solution representation.

Definition 6 (Archive). An *archive* is a collection of individuals that spans multiple generations of a coevolutionary algorithm. □

Thus, archives often contain the end solution [90] or actually are the solution, mainly when potential solutions are aggregations of entities [25, 27, 62, 28, 39, 34]. So an analogy in traditional EC for this purpose of an archive is the *best-so-far* individual. But they need not only represent solutions, they can and typically do influence the search — a bit like elitism would in a traditional generational EA.

Archives can also serve for evaluation purposes. A simple example of this are *hall-of-fame* methods [90] where successful individuals discovered during search

[9] This is analogous to traditional multi-objective optimization, where the population can be used as an approximation of the Pareto-front.

are added to a hall-of-fame and part of the fitness of new individuals comes from the outcome of interactions with some subset of individuals from that archive.

In domains with multiple types there might be an archive for each type (in addition to a population for each type). Most archive-based CoEAs involve an UPDATE step where individuals in a current population are considered for inclusion in the corresponding archive, and individuals in that archive may be re-organized or removed. There is a certain appeal to this approach because there are often straightforward ways to explicitly implement a particular solution concept by crafting the correct update procedure for an archive.

3.1.4 Solutions

A potential solution can be many different things even as part of the problem specification, the latter being a higher-level notion defined over the low-level elements of an interactive domain. It should therefore not be surprising that, as we have already seen, a potential solution can map to different things in the algorithm. A potential solution may be a single individual, in which case it may be extracted out of:

- Any population (e.g., if approaching a maximum expected utility problem in a domain with two symmetric roles via two populations);
- A specific population (e.g., if approaching a maximum expected utility problem in a domain with two asymmetric roles via two populations);
- An archive (e.g., if using a hall-of-fame approach).

A potential solution may also be a set of individuals, in which case it may be:

- The entire contents or a subset of an archive or population (e.g., for CoEAs approaching Pareto Optimal solution concepts);
- A collection of individuals, one from each archive/population (e.g., for CoEAs approaching the ideal team solution concept for asymmetric domains).

Further, there may be populations / archives in a CoEA that never contribute any individuals to a potential solution (e.g., populations / archives corresponding to test roles in a test-based problem that are nevertheless used during evaluation).

3.2 Evaluation

Some issues concerning evaluation are pertinent to both single- and multi-population CoEAs. Regardless of whether interactions occur between individuals in the same population or in different populations, decisions need to be made as to what *interactions* should be assessed and how the outcomes of those interactions should be *aggregated* to give individuals fitness. When using multiple populations, the additional issue of *communication* between these populations arises. Each of these three matters will be discussed in turn.

3.2.1 Interactions

The definition of interactions was discussed in the previous section. Here we concentrate on the *selection* of interactions (also referred to as the *interaction method* or, in cooperative domains, *collaboration method*). The simplest choice is to assess all interactions possible given the individuals present in the system at evaluation time. This has been referred to as full mixing or complete mixing. While no additional decisions would have to be made, this choice has the disadvantage of being very expensive, as the time cost of the algorithm is counted in interactions assessed. To reduce cost, one must assess only some of all possible interactions. This immediately raises the question "which ones?"

There are two main approaches to choosing a subset of interactions: individual-centric and population-centric. With the individual-centric approach, one individual at a time is considered, a set of interactions is chosen for that individual to take part in, and after the interactions are assessed, the individual's fitness is computed. These interactions may be reused (but generally are not) for computing the fitness of the other individuals that took part in them. These other individuals have historically been called *collaborators* in cooperative domains and *opponents* in competitive domains. In this context, the phrase *sample size* denotes the number of interactions used per fitness evaluation. Note that if there is no reuse, the number of interactions an individual takes part in may be greater than the number of interactions used for its evaluation. Thus, while the sample size may be (and usually is) the same for all individuals, the number of interactions that individuals are involved in may vary.

A simple and quite common approach is to use a single interaction per fitness evaluation and have the other individuals in this interaction be the best from their respective populations. When solving ideal team problems this has been termed *single-best collaboration method* by [108], while for domains considered competitive it is referred to as *last elite opponent (LEO) evaluation* [97]. More generally it is called the *best-of-generation* interaction scheme.

With the population-centric approach, a *topology* of interactions is picked such that any individual in the population(s) is used at least once, these interactions are assessed and then fitness is computed for all individuals. Tournaments, such as single-elimination or round-robin, are the most common examples for single-population algorithms. Both for single- and for multi-population models, shuffle-and-pair (also called bipartite pairing) is a population-centric method that simplifies the reuse of assessed interactions (meant to reduce computational time). With the single-elimination tournament, different individuals will participate in a different number of interactions.

Information available from previous evaluation rounds may be used to influence how interactions are chosen for the current evaluation round. We call this *fitness bias*. Usually, individual-centric approaches use such biases, while population-centric ones do not. Clearly, to be able to perform such biasing, the algorithm must store some information about previous evaluations. This information usually consists of individuals and their fitness, but can also be more complex in nature (e.g., include other properties of individuals or relationships between individuals). The

amount of information can range from remembering the last best individual to remembering all interaction assessments ever performed. With generational CoEAs, intermediary approaches include saving one or a few (usually the best) individuals from each previous generation, saving all individuals in the previous generation, or other memory or archive method [72, 25, 27, 39, 62].

Additional reviews and comparisons of various methods for selecting interactions can be found in [1, 69, 97].

With individual-centric methods, some of the additional choices (in particular the sample size) may be dynamic (i.e., they vary at run-time as a function of time) [70, 74] or adaptive (i.e., they vary at run-time as a function of the internal state of the algorithm, e.g. population diversity, operator success, etc.) [71, 72, 68].

Additional choices must be made when using spatially-embedded CoEAs. These tend to use individual-centric approaches and interactions are localized in space, namely neighborhood-based. The size of the neighborhood corresponds to the sample size, but the shape is an additional decision to be made. Fitness-biased selection of interactions can still be used (generally within the neighborhood).

3.2.2 Aggregation

If an individual participates in a single interaction, then it must obtain a value from it, and that value becomes the individual's fitness. But when an individual participates in multiple interactions and thus obtains multiple values, then a choice must be made about how to aggregate these values.

One approach is to input values from multiple interactions into some computations and output a *single value* per individual (to be used as fitness). The computations can simply use all values obtained by the individual and determine the *best*, the *worst* or the *average* of those values. A comparison of these three methods can be found in [109]. Other, more complex computations can take into account values from other individuals as well, as is the case for competitive fitness sharing [89, 90] or the biasing technique introduced by [75, 76]. The advantage of the single-value approach is that traditional parent-selection methods can then be used by the algorithm based on single-valued fitness. Unfortunately, different ways of computing a single value have different (and strong) biases, and it is not always straightforward to tell which bias is more or less helpful for the solution concept at hand. In particular, when dealing with the ideal team solution concept, the biases of the averaging method proved harmful [108] while choosing the best is helpful [68]. Even so, there has been some amount of controversy over the utility of biases [12, 71]; [8] goes so far as to argue that single-valued fitness assessments are to blame for a number of pathological algorithm dynamics.

The alternative is to have fitness be a *tuple* of values, usually the values obtained by the individual from multiple interactions. Selecting parents based on such fitnesses requires more specialized methods, in particular ones akin to multi-objective EAs. The tuples of different individuals of the same role must be somehow comparable, which imposes constraints on the interactions that generated the values in

the tuples. The advantage of this approach is that its bias may be more appropriate for solution concepts such as the Pareto Optimal Set or Simultaneous Maximization of All Outcomes, and it is mostly algorithms targeted at these solution concepts that use tuple fitness [25, 27, 38]. An example using tuple fitness for the ideal team solution concept can be found in [12].

3.2.3 Communication

When using a CoEA with multiple populations, in order for individuals in one population to interact with individuals from other populations, the populations must have access to one another's contents. This is an issue of communication, and thus entails choices typical of any distributed system, such as coordination, flow and frequency.

In terms of *coordination*, communication can be synchronous or asynchronous. In the *asynchronous* case, the populations evolve at their own pace and communicate with one another through shared memory. They decide independently when to write new information about their state to the shared memory and when to check the memory for new information about the other populations (which may or may not be available). Asynchronous CoEAs are uncommon. In the *synchronous* case, there is a centralized clock that dictates when the populations exchange information.

In terms of *flow*,[10] the asynchronous model is always *parallel*, in the sense that at any point in time there may be more than one population running[11]. The synchronous model can be either parallel or sequential (also called serial). In the *parallel* case, all populations run simultaneously for a certain period of time (dictated by the central clock), after which they all pause and exchange (communicate) information and then they all continue. In the *sequential* case, at any point in time there is a single population running and populations take turns in a round-robin fashion.

Frequency refers to the number of evaluation-selection-breeding cycles that a population goes through between two communication events. The frequency may be uniform across all populations or it may differ from one population to another.

4 Analysis of Coevolution

Particularly in the last decade, there has been a great deal of activity in analyzing coevolutionary algorithms and their use as co-optimizers. There are two major drivers of this field. One is the fact that most co-optimization problems pose difficulties not encountered in traditional optimization, namely measuring (and achieving) algorithmic progress and performance. The other is that coevolutionary algorithms as dynamical systems exhibit behaviors not akin to traditional evolutionary algorithms.

---

[10] In [108]'s hierarchy of CoEA properties, flow is called *update timing*.

[11] When a parallel CoEA is run on a single processor, this translates into the fact that there is no guarantee about the order in which the populations run.

The first half of this section clarifies the complex issues pertaining to interactive domains and co-optimization problems and places much extant research in the context of addressing such issues. The second half surveys the wide variety of analytical methods applied to understanding the search and optimization biases of coevolutionary algorithms, as well as the insights that were gained.

4.1 Challenges of Co-Optimization: Progress & Performance

For traditional optimization, judging the success of any algorithm attempting to solve a given problem instance is a fairly straightforward matter. There may be different criteria of success, such as the highest quality potential solution(s) found with a given budget of function evaluations or, conversely, what evaluation budget is needed to find potential solutions whose quality is above a certain threshold. Such success metrics also allow for easy comparisons of algorithm performance, even when aggregated over classes of problems.

Additionally, even when concerned with the behavior of a single algorithm on a single problem instance, we have a straightforward notion of progress through time. With each new function evaluation, we can easily keep track of the best quality potential solution found so far. Even if we do not know the quality of the overall best, we can know when we got closer to it. Moreover, should an algorithm actually discover a global optimum, whether or not it is recognized, there is no danger of "losing" it. The best-so-far quality metric is a monotonic function of time.[12]

All of this is largely possible because the quality of a potential solution can be obtained with a single function evaluation and thus immediately perceived by the algorithm, which in turn allows reliable comparison of any two potential solutions. As trivial as the above statements may appear, they are no longer a given in many co-optimization problems.

We shall start, however, with the one area that is most similar to traditional optimization: compositional problems using the ideal team solution concept. The unique feature of these problems is that the interaction set is actually the same as the potential solution set *and* the interaction function serves as the potential-solution quality function. Thus, the quality of a potential solution can be obtained with a single function evaluation and any two potential solutions can reliably be compared. This, of course, is not very surprising since many of these problems are actually obtained via decomposition from traditional optimization problems. Therefore, measuring performance and progress is not an issue.

Nevertheless, approaching these problems via coevolution raises new issues. For example, how do we assign credit to the components of potential solutions that are evolving in different populations? More generally, how do different algorithm design choices affect algorithm performance? Much research around the topic has shown that, even for these most traditional-optimization-similar problems, design

[12] The potential solution with the optimal quality may change, should the algorithm be deliberately implemented to replace the current best-so-far potential solution with a new one of equal quality.

heuristics for traditional evolutionary algorithms do not transfer as-is to coevolution [80]. Still, the ability to easily measure performance and progress greatly facilitates the study of coevolutionary algorithms for ideal team compositional problems.

Note that the equality of the potential solution set and the interaction set is not enough to guarantee ease of performance/progress assessment. Let us consider compositional problems using the pure Nash equilibria solution concept. This problem is not expressed as a co-optimization problem, but as a generic co-search problem. It is fairly obvious that the single evaluation of the interaction function for this potential solution is not enough to determine whether it constitutes a Nash-equilibrium. To determine that, we would need to, in turn, fix all but one component of the potential solution and evaluate interactions corresponding to, in the worst case, the entire set of entities for that component. While this does not mean exploring the entire interaction set, it does mean fully exploring all the component sets, which is to be assumed intractable. We might reach a negative decision sooner than that, but to reach a positive decision we must perform such exhaustive search.

For test-based problems, the potential solution set is always different from the interaction set and determining the quality of a potential solution requires evaluating interactions corresponding to entire entity sets (the test sets). Should this be tractable, the problem may be expressed as a traditional optimization problem. We assume that is not the case.

Consider a generic search algorithm that at every step evaluates a new interaction and outputs a potential solution.[13] How are we to judge whether the algorithm makes progress towards the solution to the entire problem? How are we to compare its output with the output of a different algorithm that has evaluated the same number of interactions? Since determining the true quality of a potential solution is generally computationally intractable for real problems, we are left with attempting to compare different potential solutions based on incomplete information.

The above observations paint a rather bleak view with respect to determining whether an algorithm could tackle one of these problems. However, the matter is not hopeless, though more can be said and done for some problems than for others.

For example, for Pareto-dominance solution concepts it has been shown that for any given set of n component entities (components), an *Ideal Evaluation Set* of at most $n^2 - n$ test entities (tests) exists that results in exactly the same dominance relations among the components as would result from using the set of all possible tests [32]. Of course, the guaranteed existence of this limited set of tests leaves open the question of how to identify them. However, an operational criterion is available to determine whether any given test should be included in such a limited test set. Specifically, the criterion of whether a test induces a *distinction* between two components, introduced in [38], can be used for this purpose; see [32].

The vast majority of research on these issues has concentrated on progress of (co-optimization) algorithms in individual runs and assessing performance on test problems. Only recently have aggregate performance comparisons between algorithms started receiving attention.

[13] Algorithms that may not be ready to output a potential solution after every evaluation can be interpreted as outputting their latest output again until ready.

With respect to progress in individual runs, one fact facilitates the analysis: the incomplete information that an algorithm sees, and based on which we should compare different potential solutions outputted by the algorithm, is increasing over time. The remarkable discovery of [34] is that some problems (solution concepts) permit the construction of algorithms that can properly leverage this incomplete but increasing information to guarantee that outputted potential solutions have monotonically increasing *global/true* quality over time. We say of any such algorithm that it guarantees *global monotonicity* and also call the said solution concepts *globally monotonic*.

Some problems of practical interest have solution concepts that are not globally monotonic. This makes it difficult or even impossible to devise algorithms that guarantee global monotonic progress as described above. Still, for some of these problems, algorithms can be built that guarantee a different form of monotonicity called *local monotonicity*, which entails that the quality of outputted potential solutions with respect to all information seen so far will increase with time.

Both globally and locally monotonic algorithms guarantee convergence to a globally optimal solution, if the interaction space is finite and any point in this space can be generated with non-zero probability at any time.

When neither global nor local monotonicity guarantees can be achieved, some surrogate techniques can be used to instrument what the algorithm is doing, though they do not inform about progress towards the problem solving goal. Finally, while all of the above assume the intractability of computing global quality, one research approach is to study various co-optimization algorithms on test problems, for which global quality can actually be computed. This metric is not made available to the algorithm, but used externally for instrumentation, and the hope is that the observed results will transfer to real problems.

In the remainder of this section we review, in turn, global and local monotonicity, surrogate measures of progress, test-bed analytical techniques and aggregate performance comparisons.

4.1.1 Monotonicity

Central to the work on monotonicity is the idea that the solution concept can be interpreted as more general than just a means of partitioning the entire set of potential solutions based on the entire set of interactions. Namely, it can be extended to apply to any context consisting of a subset of interactions and a corresponding subset of potential solutions (i.e., potential solutions that can be built with entities that participated in the given interactions). Thus, it can be viewed as a means for an algorithm to output a potential solution when queried, based on all or some of the entities it has generated and interactions it has assessed.[14]

In general, it may be difficult for an algorithm to determine the output of the solution concept when applied to the entire history of entities discovered and inter-

[14] Conversely, any mechanism for selecting a potential solution to be outputted from the algorithm can be viewed as a solution concept.

actions assessed if it has discarded some of the entities or some of the interaction outcomes. However, for some solution concepts this is possible. For example, for the best worst case solution concept, an algorithm needs to remember only the worst value seen for any potential solution and not all the interactions it took part in, the other entities involved in those interactions or their outcomes.

One question that arises is whether an algorithm able to report the application of the solution concept to its entire history (whether by means of never discarding any information or by cleverly storing parts or aggregates of this information) can guarantee that the *global* quality of the potential solutions it outputs will never decrease over time, regardless of the means of generating new interactions to be assessed. Ficici showed in [34] that this is possible for some solution concepts, which we call *globally monotonic*, but not for others. Out of the solution concepts presented in Sect. 2.4, the ones proven to be globally monotonic are: the Nash equilibrium, the Pareto-Optimal minimal equivalence set [34], maximization of all outcomes and ideal team [94]. Solution concepts proven not to be globally monotonic are: maximization of expected utility [34], best worst case [94, 84] and all Pareto-dominance ones beside the minimal equivalence set [34].

For the globally monotonic solution concepts, the challenge from an algorithmic perspective is being able to report the application of the solution concept to the entire history while limiting the amount of history that must be stored. The algorithms mentioned below are in many cases able to discard information without jeopardizing the guarantee of global monotonicity. Yet, all of the globally monotonic algorithms discussed below require unlimited memory.

For the Pareto-Optimal minimal equivalence set solution concept, [25] introduced an algorithm called *IPCA* (Incremental Pareto-Coevolution Archive) that guarantees global monotonicity [84] while discarding information along the way. The key to achieving that lies in a carefully designed archiving mechanism. A variant named the LAyered Pareto Coevolution Archive (LAPCA) [27] addresses the unbounded memory issue by maintaining a limited number of non-dominated layers. While it no longer provides the monotonicity guarantee, this algorithm achieves good progress on test problems. Similarly, the *Nash Memory* algorithm [39, 34], guarantees global monotonic progress for the solution concept of the Nash equilibrium in symmetric zero-sum domains while discarding information. As before, bounding the memory loses the theoretical monotonicity guarantee but still achieves good progress on test problems. Based on the Nash Memory, the *Parallel Nash Memory* algorithm [62] guarantees global monotonic progress for the solution concept of the Nash equilibrium in asymmetric games. For the maximization of all outcomes solution concept, the *Covering Competitive Algorithm* of [88] guarantees global monotonic progress while discarding information.

While the guarantee of increasing global quality may not be possible for solution concepts that are not globally monotonic, this does not mean that all hope is lost for problems with such solution concepts. For such problems it may still be possible to devise algorithms that guarantee that the quality of outputted potential solutions *with respect to the interactions seen so far* increases monotonically over time. Solution concepts with this property are called *locally monotonic*.

For example, while the Maximization of Expected Utility (MEU) solution concept introduced in Sect. 2.4.2 is not globally monotonic, a variant of it, namely maximum utility sum applied to domains with a positive-valued metric, allows algorithms guaranteeing locally monotonic progress [84]. As the name suggests, the MEU solution concept is an optimization criterion, where the function to be optimized is utility sum. When extended to any context consisting of a subset of interactions and a corresponding subset of potential solutions, the special property this function has (if the metric values are positive) is that the "local" value for a potential solution in some context can only increase when the context is expanded.

Based on this property, for maximum utility sum applied to domains with a binary-valued metric, [28] introduced an algorithm called *MaxSolve* that is guaranteed to output the application of the solution concept to the entire history (and therefore an increasing value with respect to an increasing context), while discarding information along the way. This property, coupled with unbounded memory, allows *MaxSolve* to achieve local monotonicity.

As [84] showed, for the best worst case solution concept even the local monotonicity guarantee cannot be fully achieved via algorithms that use the solution concept as an output mechanism (and probably not in general).

While monotonicity is an interesting property that has served in clarifying the behavior and goals of coevolutionary solution concepts and algorithms, it should be made clear that guarantees of monotonicity can have limited value for a practitioner. Three important things to keep in mind are that (a) the solution concept is most often a given, not a choice, and for some solution concepts there are no known algorithms with monotonicity guarantees; (b) all algorithms known to have monotonicity guarantees require unbounded memory; and (c) even if the quality of potential solutions can only improve over time, this provides no information about the *rate* at which progress will be achieved. For these reasons, an approach that may turn out to have more practical relevance is to consider the *performance* of coevolutionary algorithms. This topic is treated in the Sect. 4.1.4.

4.1.2 Surrogate Progress Metrics

For problems where the above monotonicity guarantees are unobtainable, we are left with two questions. One is how to design algorithms for tackling these problems. Another is how to instrument any such algorithm. The former is still an open research question. Here we address the latter and do so from the perspective of coevolutionary algorithms. However, while these techniques have been introduced in the context of coevolution, some of them may be adapted to be applicable to generic co-optimization algorithms.

Many techniques have been introduced over the years, and while they cannot tell how the algorithm is doing with respect to the goal, they may still provide useful information about algorithm behavior. The techniques involve computing metrics that are external to the algorithm (i.e. they do not influence its behavior) but rely on its history. They may measure all potential solutions considered by the algorithm or

only select ones (e.g. ones the algorithm outputs). Measuring all can provide more information, but it may make it difficult to extract the important patterns within and may also pose a challenge for visualization. One must beware that compressing information (e.g. by means of averaging) may actually eliminate important trends in the data. The trends to look for usually include: increase, decrease, stagnation, noise, repeated values.

The first such technique was introduced by [24] in the form of CIAO plots. The acronym stands for "current individual ancestral opponent". For a two-population CoEA, a CIAO plot is a matrix in which rows represent generations of one population and columns represent generations of the other population. Every cell represents an interaction between the best individual in each population (as reported by the algorithm) at the corresponding generations. Thus, individuals from later generations of one population interact with individuals from early generations of the other population (ancestors). The cells are color-coded on a gray scale, and can be constructed to reflect success from the perspective of either population.

The "master tournament" metric of [40] is basically a compression of the information in CIAO plots. Averages are taken along lines for one population and across columns for the other population.

Both these methods are computationally expensive, as they require the evaluation of n^2 interactions, where n is the number of generations. The master tournament metric makes it easier to identify "broadly-successful" individuals, but the averaging it performs may obscure some circularities that can be observed using the CIAO plots. An in-depth critique of CIAO plots is provided in [22].

While these metrics were introduced in the context of domains with two asymmetric roles, they are easily applicable to domains with two symmetric roles, whether approached by single-population or two-population CoEAs.

In the context of a domain with two symmetric roles and a two-population CoEA, [101] introduced a less costly technique called "dominance tournament". At each generation, the best individual in each population is determined and the two of them are paired; out of their interaction, the more successful one is designated the generation champion. The dominance property is then defined as follows. The champion from the first generation is automatically considered dominant. In every subsequent generation, the champion is paired only with previous dominant champions and it is itself labeled dominant only if it is more successful than all of them. The method is easily applicable to domains with two symmetric roles and a single-population CoEA. The paper also suggests a (less straightforward) extension to some domains with two asymmetric roles.

CIAO, master and dominance tournament are all techniques that track only the best of generation individual. This was contrasted by [3] that introduced a "population-differential" technique monitoring all individuals in each generation. Plots similar to the CIAO plots are produced, but now each cell is an aggregation over the results of all pair-wise interactions between individuals in the two populations at the generations corresponding to that cell. This is clearly more expensive, and a number of memory policies are introduced for reducing time complexity.

The unifying idea of these techniques is to determine whether an algorithm is making at least some sort of "local" progress away from the starting point.

All the metrics described so far require performing additional evaluations, rather than using the ones already performed by the algorithm. [42] introduced a technique that took the latter approach. Their metric, called "relative strength" is still subjective, as the value it returns for a particular individual depends on the current history of all interactions, and this history grows with time. The metric is based on paired comparison statistics and is applicable to domains with two symmetric roles.

4.1.3 Test-Bed Analysis

A means of judging both an algorithm's ability to make progress as well as its expected performance on some problem(s) is to test the algorithm on artificial domains for which true quality can be computed. The domains are constructed such that they exhibit properties thought to be present in real domains or such that they expose a range of different behaviors of the algorithm. The hope is that the results observed on the artificial domains will transfer to real domains.

A class of domains introduced to this effect are number games [106]. These are domains with two roles, a common entity set consisting of real-valued vectors and a binary-valued interaction function (based on comparisons between vector values). The roles can be interpreted as symmetric or asymmetric. The solution concept is Pareto-dominance. What makes these problems analysis-friendly is the fact that the interaction function is designed such that a genotype can be used as-is to represent its own true quality. There are a handful of such interaction functions in the literature and many progress-focused algorithm comparisons have been performed using them as a testbed [25, 27, 28, 29, 10, 43].

[82] introduced a set of four domains with two asymmetric roles, a common entity set consisting of single real values and a real-valued interaction function given in closed form. These domains were used to showcase the biases of symmetric last-elite-opponent evaluation, and how this technique generally does not optimize expected utility. The closed form of the interaction function allowed for easy computation of true quality.

A number of domains of the same kind (two asymmetric roles, real-valued entities, closed-form real-valued interaction function) have been introduced specifically for studying algorithms targeted at the best-worse-case solution concept [52], which has been shown to be non-monotonic and difficult in terms of achieving even very weak forms of progress [84].

Test-bed analysis has also been performed for algorithms targeted at the ideal team solution concept, yet with a different motivation (e.g. studying problem properties and algorithm design choices) since in this case assessing global quality is not an issue. Examples are given in section 4.2.4.

4.1.4 Comparing Performance

As emphasized throughout the previous section, instrumenting an algorithm from a progress-making perspective or, indeed, designing an algorithm such that it guarantees progress relies rather heavily on the fact that the amount of information seen by the algorithm increases over time. How an algorithm uses this information can be key to instrumenting/achieving progress. Therefore, the theoretical work on monotonicity has so far had nothing to say about how to compare algorithms since there is no connection between the information that different algorithms see. This line of research provided no generic advice on how to design efficient algorithms.

While test-bed analysis is useful for determining whether an algorithm makes progress or not, when global quality can be computed, one can also use test-beds to compare algorithms based on the expected rate of progress or expected level of solution quality over multiple runs. This kind of performance analysis, especially when backed by dynamics analysis of the kind reviewed in section 4.2.5, has provided insight into the biases of design choices by providing a means of comparison; however, it is unclear how well the results transfer to real-world problems, so the generality of such results is unclear.

From traditional optimization research, the famous no free lunch (NFL) theorem has approached the issue of performance generality [114, 93]. This result has led to a welcome shift in EC research towards identifying classes of problems where NFL does not hold and designing algorithms targeted at a specific class [48]. This is of particular current interest in co-optimization/co-evolution [115] since recent work states the opposite, namely that free lunches do exist for certain solution concepts.

Wolpert and Macready extended their formalism of traditional optimization algorithms to co-optimization algorithms (albeit without using this terminology) as having two components: a search heuristic and an output selection function. The search heuristic takes as input the sequence of interactions assessed so far and returns a new interaction to be assessed. The output selection function takes the same input and returns a potential solution. The need to explicitly model the output selection function comes from the algorithms' lack of access to a global quality function: in traditional optimization output selection is based on the quality values seen so far (usually picking the best of them); in co-optimization what an algorithm sees are values of the interaction function, which provide incomplete information about global quality. The use of archives in CoEAs is often targeted at implementing the output selection function, thus relieving the populations of this duty so they can focus on the search component.

Wolpert and Macready showed that aggregate performance advantages (free lunches) can be obtained both via the output selection function and via the search heuristic. The former means that if we fix the search heuristic, some output functions perform better than others. The optimal output function is the so-called Bayes output function, which outputs the potential solution with the highest estimated global quality, estimated over all possible problems consistent with the measurements of the interactions seen so far. The latter was showcased in the context of the best worst

case solution concept: fixing the output selection function to Bayes, two specific search heuristics were shown to have different average performance.

This line of work was extended by [96], which showed that the same two algorithms from [115] exhibit different aggregate performance for the maximization of all outcomes solution concept.

Generally, such free lunches are believed to exist in co-optimization whenever the performance of an algorithm depends on information not available in the algorithm's trace of interactions, in other words it explicitly depends on the interaction function (as opposed to implicitly, via the values in the trace). Therefore, the ideal team solution concept does not exhibit free lunches since the interaction set is actually the same as the potential solution set and the interaction function serves as the potential solution quality function.

The existence or lack of free lunches can be seen as a property of solution concepts, which has been named "bias" [94]. Since monotonicity is also a solution concept property, the natural question that arises is how these two properties relate to one another. [94] showed they are orthogonal to one another: solution concepts exist in all four classes combining monotonic/non-monotonic with biased/unbiased.

Especially for monotonic and biased solution concepts, the next natural question is whether the algorithms that can achieve monotonicity (known to exist) are also the best-performing. This question was investigated in [84], which showed that the output selection function used by monotonic algorithms is not equivalent to the best-performing Bayes, thus exposing a potential trade-off in algorithm design between monotonicity and performance.

4.2 Analysis of CoEA Biases: A Survey

The previous half of this section discussed the difficulties of measuring and achieving algorithmic progress for co-optimization problems, and reviewed the state of the art in understanding and dealing with these issues. However, when applying coevolutionary algorithms to such problems, an additional concern is understanding and harnessing the biases these algorithms have as dynamical systems. This section surveys the analytical research that contributed such understanding.

4.2.1 Model Analysis

A common algorithm analysis method is to *model* an algorithm mathematically, then to study the properties of that model. The advantage is that certain mathematical facts can sometimes be precisely learned from these models, facts that help us understand system behavior in ways that are simply not possible via traditional approaches. The disadvantage is that the model is *not* a real algorithm, and it is not always clear what can be transferred from model to algorithm. This step is typically bridged by empirical study of some kind.

The simplest way to approach modeling evolutionary computation as a dynamical system is the so-called "infinite population model" [105]. Here we assume that a population is infinitely large and focus on how the distribution of genotypes within the population changes over time. Questions about the existence (and sometimes the location) of stable, attracting fixed points, or their nature and quality, can sometimes be answered by applying traditional dynamical systems methods to such a model.

Evolutionary Game Theory (EGT) [45, 107, 58] is an appealing dynamical systems model of biological systems that is well-suited for studying coevolution. In EGT, interactions between genotypes are treated as a *stage game*, and the payoff from that game informs how the distribution is altered for the next play of the game (*replicator dynamics*). Because of its incorporation of game theory, a number of game-theoretic properties of such systems become useful points of consideration (e.g., Nash equilibria). Additionally, EGT has received a great deal of attention by the economics community [41]. Consequently, quite a bit is already known about the dynamical properties of these mathematical models under different conditions.

For the most part, EGT analysis has focused on the dynamics of CoEAs under selection only; however, variational operators in CoEAs can be (and have been) modeled as is typically done in dynamical systems models of traditional EAs, by constructing a matrix that transforms the probability distribution resulting from the selection operator into one that reflects the properties of the population after crossover and mutation are applied, the so-called *mixing matrix* [105]. To specify this, one must be explicit about representation. Despite these simplifications some interesting things are known about the system that inform our understanding of CoEAs.

In single population EGT models of CoEAs, for example, stable attracting *polymorphic* equilibria can exist when the underlying stage game is not a constant sum game. That is, aside from the stochastic effects of genetic drift, selection alone can drive the systems to states where the population is a mixture of different kinds of genotypes, and these stable, mixed strategies correspond with a Nash equilibrium of the stage game. Moreover, examining this system using other selection methods has uncovered some fascinating results. Common EC selection methods (e.g., truncation, (μ, λ), linear rank, Boltzman) can sometimes lead to systems in which there are stable cycles and even chaotic orbits [37]. Indeed, the dynamics of CoEAs can be much more complex than traditional EAs in that stable attractors of infinite population models can be quite unrelated to the specific values in the game payoff matrix itself. Empirical experiments on real algorithms verify that these fundamental dynamical pressures of the selection operator do, in fact, affect how real CoEAs perform on real problems.

Additionally, by considering certain game-theoretic properties of the stage game (a particular variation of the notion of constant-sum), one can show that the behavior of certain single population CoEAs using non-parametric selection operators will, in principle, behave dynamically like a comparable EA on some unknown fitness function [57], while this cannot be said for other systems *even when the payoff is fully transitive*. This reinforces the idea that certain co-optimization problems are fundamentally different from traditional optimization problems.

EGT-based models have also been used to study two-population compositional CoEAs. From these we learn that any pure Nash equilibrium of the underlying payoff game is a stable, attracting fixed point. This suggests selection alone can drive compositional CoEAs toward points that are globally very suboptimal in terms of their actual payoff value. Experimentation confirms this is so in real, analogous compositional CoEAs even to the extent that highly inferior local optima draw significantly more populations than do the true global optima. Indeed, complete mixing implies a solution concept wherein "optimality" is defined solely based on how individual strategies do on average over all possible partners. If one is using such methods to try to find a globally maximal payoff value, this can be seen as a kind of pathology (*relative generalization*). This dynamic may well be consistent with obtaining solutions that meet certain specific definitions of *robustness* [111].

But most compositional CoEAs use best-of-generation interaction methods, not complete mixing, and [73] noted that these more common methods bias compositional coevolutionary search of projections of the payoff space to favor aggregate projections that suit identifying the global optimum. Further, by incorporating archival notions that help retain *informative* partners in terms of these projection estimates, we can further improve the compositional CoEAs optimization potential. These analytical lessons have been applied to both real compositional CoEAs, as well as other RL-based co-optimization processes [68].

From these studies, we see that interaction and aggregation have profound impacts on the dynamical influences of selection in compositional coevolution. More specifically, the complete mixing interaction pattern may be ill-suited for problems with an ideal team solution concept. Another, altogether different dynamical systems approach by Subbu and Sanderson [104] provides convergence results for a particular class of distributed compositional CoEAs to ideal team style solutions under conditions where complete mixing is not even possible.

In addition to the infinite population models described above, several works have focused on finite populations. EGT itself has been applied to finite population situations [58, 36], but typically under very constrained circumstances (two-strategy games). Outside of using EGT-based methods, there have been attempts to model the dynamics of finite, two-population CoEAs using Markov models [56]. While these methods also restrict the size of the problem spaces for obvious reasons, they have important implications for infinite population models in the sense that they clearly demonstrate that the long-term behavior of these systems can differ quite significantly from infinite population models.

4.2.2 Runtime Analysis

EC has made good use of classical algorithm analysis methods to find bounds on the expected number of evaluations necessary before a global optimum has been obtained as a function of the size of the (typically, but not always, combinatorial) genotype space [63]. These methods have the advantage of providing precise and correct performance bounds on real algorithms for real problems, but their chief

disadvantage is that it is often difficult to gain crucial insight into the behavior of an algorithm. This concern is typically addressed by selecting problem classes that help identify important properties in the problem and falsify hypotheses regarding how EAs address them.

Indeed, runtime analysis has been quite useful in understanding how simple compositional coevolutionary algorithms approach problems with ideal team solution concepts. A common hypothesis about such algorithms is that they tend to perform better when the representation choices result in a linear separation of problem components with respect to the objective function. Runtime analysis of simple $(1+1)$ [49, 50] and steady-state-like [51] variants of compositional CoEAs revealed this hypothesis to be conclusively false: though non-linear relationships between components can have profound effects on the relative performance differences between a coevolutionary algorithm and a similarly structured EA, separability itself is neither a sufficient nor necessary property with which to predict problem difficulty [51]. Additionally, there exist certain problem classes that essentially *cannot be solved* by a large class of compositional coevolutionary algorithms. Finally, this work suggests that compositional CoEAs solve such optimization problems by leveraging the decomposition to partition the problem *while* increasing the focus of the explorative effects of the variational operators (so-called *partitioning and focusing* effect).

Runtime analysis of other CoEAs is more challenging and has not yet been done. Fortunately, recent research provides several needed pieces to overcome these challenges. First, there's been a great deal of recent progress in analyzing increasingly more complex multi-objective methods [55], and much of this may help study CoEAs that employ multi-objective mechanisms (e.g., archives). Second, the formalisms for explicit definition of solution concepts and the guarantee of monotonicity in certain concepts provide clear global goals needed for runtime analysis.

4.2.3 Probabilistic / Convergence Analysis

Another way to investigate algorithms theoretically, is to consider the dynamical behaviors of the real algorithms (as opposed to abstractly modeling them). Typically such analyses are focused on providing convergence guarantees, but often careful consideration of problem and algorithm properties can provide greater insight.

[91, 92] provide a strong type of convergence analysis. For the maximization of all outcomes solution concept, the proposed algorithm is guaranteed not only to find a solution, but to have its populations converge to containing only genotypes corresponding to the solution(s). Markov-chain analysis is used for the proofs. Bounds on convergence speed are not given; however, the work provides some useful general advice for how to apply rates of genetic operators to ensure such convergence.

Additionally, [43] conducts a probability theory based investigation into the relationships between problem properties, algorithm properties, and system dynamics. This work focuses on test-based CoEAs, and demonstrates that the interplay of the considered properties cause trajectories through the domain-role sets to either make progress or resemble random-walk behaviors.

4.2.4 Empirical Black-Box Analysis

Analysis of coevolution can be approached empirically by viewing the system as a black box whose inputs are the algorithm and the problem, and the output is observed performance (e.g. quality of potential solution outputted given a certain amount of time). The algorithm and / or the problem are then varied and the output of the system re-observed, with the goal of determining the rules governing the dependency between inputs and outputs. Such studies are performed for problems with computable global quality, such as ideal team problems or test-beds of the kind reviewed in Sect. 4.1.3.

Two different approaches can be distinguished within the black-box analysis category. One approach focuses on *properties* of the algorithms, of the problems or of both. When algorithm properties are involved, this approach has been referred to as component analysis [108]. The other approach varies the algorithm and / or the problem and compares the resulting performance without a clear isolation of the properties responsible for the differences in performance.

We review this latter approach first. Most works introducing new algorithms are of this kind. This includes some of the early research comparing CoEAs with traditional EAs, such as [44, 1] for test-based problems and [86] for compositional problems. Later on, the approach was used to compare CoEAs amongst themselves, usually "enhanced" algorithms with basic ones. Examples include [38, 39, 34, 32, 25, 27, 28] for test-based problems and [12, 72, 71, 68] for compositional problems. Some of the algorithms introduced by these works [25, 28, 32] are backed-up by theoretical results.

Property-focused approaches address problem properties, algorithm properties, or the interactions between them. We discuss these cases in order.

Some of the CoEA-versus-EA comparisons were extended to determine the classes of problems for which one type of algorithm would provide performance advantages over the other. Such work was performed for compositional problems by [85] and [111]. The scrutinized problem properties were *separability* [108], *inter-agent epistasis* [15], *dimensionality*[15], *noise* and *relative sizes of basins of attraction of local optima*.

For test-based problems, empirical black-box analysis was used to investigate problem properties such as *role asymmetry* [64], *intransitivity* [26] and *dimensionality*[16] [30]. These works introduced specialized algorithms intended to target the respective problem property.

Studies focusing only on algorithm properties investigated either the mechanism for *selecting interactions* [74, 70, 78, 6] or the mechanism for *aggregating* the result of those interactions [75, 109, 19]. All but the last cited work were concerned with compositional problems.

Finally and perhaps most importantly, some black-box empirical studies analyzed the effects on performance of the interdependency between algorithm prop-

[15] Here dimensionality refers to the number of roles.

[16] Here dimensionality refers to the number of underlying objectives implicitly defined by the test role(s).

erties and problem properties. Most studied has been the mechanism for selecting interactions, and most of those targeted compositional problems [85, 109, 110, 108, 14, 16]. The launched hypothesis was that the performance effects of the interaction method are tightly related to the (previously mentioned) problem property separability and the kind of *cross-population epistasis* created when representation choices split up inseparable pieces of the problem. This dependency however proved not to be as straightforward as thought, as it was later shown in [81] via empirical dynamics analysis. For test-based problems, two different methods for selecting interactions were analyzed by [69], suggesting that the amount of noise in the problem affects their influence on performance.

For compositional problems, [76] extended their previous work on the mechanism for aggregating results from multiple interactions by studying how its performance effects are affected by the problem's local optima and their basins of attraction. [112] studied the potential benefits of *spatially-distributed schemes for selecting interactions and / or selecting parents* on domains with role asymmetry.

[15] studied the effects of mutation and crossover in the context of "mixed-payoff" domains ([54]'s NKC landscapes) and found them to be sensitive to inter-agent epistasis.

4.2.5 Empirical Dynamics Analysis

While black-box analysis can provide some heuristics for improving performance, it cannot explain the causes for the observed dependencies between the inputs and the outputs of a CoEC setup. To understand these causes, one needs to observe the system while running and track some of its time-varying properties. Instead, dynamics analysis is used to explain the connection between algorithm and problem properties on one side and performance on the other side. Individual studies connect different subsets of the pieces, analyzing:

- dynamics in isolation [24, 101, 3, 40, 2, 22];
- problem properties and dynamics [17];
- problem properties, algorithm properties and dynamics [18, 54];
- dynamics and performance [67, 53, 35, 59, 42, 77];
- problem properties, dynamics and performance [106];
- algorithm properties, dynamics and performance [113, 66, 89, 90, 20, 21, 10];
- how the combination of algorithm properties and problem properties influences dynamics and how dynamics influence performance [79].

These analyses used some of the progress metrics described in Sect. 4.1.2, and/or various techniques for tracking genotypic/phenotypic changes. Studies involving the latter were performed for domains and representations that are easily amenable to visualization.

[24] were also the first to use techniques for tracking genotypic changes in order to analyze the run-time behavior (dynamics) of CoEAs. They introduce "elite

bitmaps", "ancestral Hamming plots" and "consensus distance plots" as tools complementary to CIAO plots. They all work on binary representations. Elite bitmaps simply display the genotype of best-of-generation individuals next to each other in temporal sequence. Some additional processing can reveal interesting patterns. Ancestral Hamming plots display the Hamming distance between elite individuals from different generations. Consensus distance plots monitor the genotypic make-up of the whole population through the distribution of Hamming distances from the genotype of each individual to the population's "consensus sequence". All three techniques are applicable to EAs in general, not just CoEAs.

[79] extensively used best-of-generation trajectory plots for understanding the effects of various CoEA design choices and exposed best-response curves as a problem property that is a strong driver of coevolutionary algorithm behavior.

It is via the empirical dynamics analysis approach that most knowledge about the pros and cons of using CoEAs for co-optimization was generated. And just as traditional canonical evolutionary algorithms can be used for optimization, yet they are not intrinsically optimizers [33], canonical coevolutionary algorithms may be useful in co-optimization, yet they are not intrinsically co-optimizers.

5 The Future of Coevolution

The bulk of this chapter has focused on how coevolutionary algorithms, co-search problems, and interactive domains have been studied. This section is dedicated to the future of coevolution, including current lines of work, research questions, and results with intriguing but so-far-untapped potential.

5.1 Open Issues

Recent research has helped establish a firm, formal foundation for coevolutionary problem solvers, and it has provided some principled ways to think about, design, and apply such systems. Nevertheless, many issues remain quite unresolved. We group these into two categories: those relating disparate theoretical analyses and those relating our analytical understanding of CoEAs to their practical application.

Relating Theories:

Dynamical systems analysis of compositional coevolution has demonstrated that certain robustness properties in the problem can attract populations [111]. But there has yet to be any attempt to formalize this idea as a solution concept, and it remains to be seen whether such a concept is monotonic. Similarly, Lothar Schmidtt's convergence analysis [92] provides useful feedback about how to manage the param-

eters of operators; however, it makes very specific assumptions about the solution concept, and no work has yet approached the task of generalizing his ideas to include CoEAs as applied to problems with different concepts of solution. Runtime analysis for compositional approaches has yielded precise bounds for those algorithms on certain optimization problems [51], and there has been a lot of success using similar methods to analyze performance of EAs on multi-objective optimization problems. This suggests it should be possible to apply such analyses to CoEAs for test-based problems, due to their underlying multi-objective nature.

Though in a handbook of natural computing, this chapter has tried to present co-evolutionary algorithms in the broader context of co-optimization. As pointed out in Sect. 4.1, some of the difficulties CoEAs face, such as the difficulty to monitor and achieve progress, are specific to the nature of certain co-optimization problems and not the evolutionary aspect of these algorithms. Any other algorithms attempting to solve such problems will have to deal with these issues. Formalizing co-optimization algorithms as having two components, a search heuristic and an output selection function, allows for easy transfer of results pertaining to monotonicity and performance to algorithms using non-evolutionary search heuristics, such as the ones introduced in [95].

Last but not least, many research questions on how algorithms should be designed have been opened by the recent confluence of the two generic co-optimization theories concerning monotonicity and performance [94, 84]. We expound below.

Relating Theory to Practice:

As discussed earlier in the chapter, the notion of monotonic solution concept not only provides a way to characterize problems but also allows engineers to construct principled algorithms for which consistent progress toward a solution can be expected [34]. Still, on a practical level, several questions remain. Many algorithms that implement monotonic solution concepts have unrealistic memory requirements. Researchers are only beginning to try to understand what the impact of practical limits on mechanisms such as archive size have on performance and theoretical guarantees about progress. Similarly, hidden in the free lunch proofs of [115] was advice on how to obtain some performance advantage, such as avoiding unnecessary evaluations, exploiting biases introduced by the solution concept and using a Bayes output selection function. Still, determining how to use this advice in real algorithms for real problems is very much an open research issue. Further, the design choices required by monotonicity can conflict with those required for performance [84] and additional research is needed to determine when/if this tradeoff can be avoided or at least how to minimize it. Indeed, recent work relating monotonicity, solution concepts, and the NFL suggest this relationship could be quite complex [94, 84].

The same types of questions can be raised with respect to methods used for the discovery of informative dimensions of co-optimization problems: Can heuristic methods be developed that can approximate this search process in the general case, and what are the precise tradeoffs of such an approximation? Finally, under certain

conditions, it is theoretically possible for some CoEAs to prefer components considered later in the run, which in turn will tend to require larger supporting test sets [8]. This implies the possibility of a kind of bloating that practitioners may need to address in practical application.

5.2 Discovery of Search Space Structure

[32] presents empirical results suggesting that a Pareto coevolutionary algorithm could find what were dubbed the *underlying objectives* of a problem. These are hypothetical objectives that determine the performance of components without the need to interact them with all possible tests. [32] applies a two-population Pareto coevolutionary algorithm, DELPHI, to instances of a class of abstract interactive domains. Figs 13 and 15 of that work suggest that evaluator individuals[17] evolve in a way that tracks the underlying objectives of the domain. The results suggest that the algorithm is sensitive to the presence of underlying objectives even though it is not given explicit information about those objectives, exhibiting what has been called an "emergent geometric organization" of test individuals [8]. [10] makes a similar observation, also empirical though using a different algorithm; Fig. 5 of that work suggests a similar sensitivity to underlying objectives. In both cases, clusters of individuals, rather than single individuals, move along or collect around the known objectives of the domain. The problems considered, namely numbers games [106], were designed to have a known and controllable number of objectives, but the algorithms used in these two studies did not rely on that fact. The work therefore raises the question of whether underlying objectives exist in all domains, and whether algorithms can discover them.

A partial answer to this question is found in the notion of *coordinate system* defined in [13]. Coordinate systems, which were defined for a class of test-based problems,[18] can be viewed as a formalization of the empirically-observed underlying objectives of [32]. To elaborate, a coordinate system consists of several *axes*. Each axis is a list of tests ordered in such a way that any component can be placed somewhere in the list. A component's placement is such that it does well against all tests before that spot and does poorly against all tests after it. For this reason, an axis can be viewed as measuring some aspect of a component's performance: a component that places high on an axis is "better than" one that sits lower in the sense that it does well against more tests (more of the tests present in the axis, not more tests of the domain as a whole). Formally, an axis corresponds to a numerical function over the components, in other words to an objective function. It can be proven [13] that every domain of the considered class possesses at least one coordinate system, meaning it has a decomposition into a set of axes. In short, every such domain has

[17] What we have called tests.

[18] Specifically, problems with a finite number of components and a finite number of binary-outcome tests.

some set of objective functions associated with it, one for each axis in a coordinate system for the problem.

Besides defining coordinate systems formally, [13] gives an algorithm that finds a coordinate system for an interactive domain in polynomial time. The algorithm, though fast, is not guaranteed to produce the smallest-possible coordinate system for the domain, however. Finite domains must have a minimal coordinate system, but in general even finite domains can have distinct coordinate systems of different sizes. The algorithm is not coevolutionary per se, as it examines the outcomes of tests on components. It is therefore applicable to the entire class of test-based problems and agnostic about the specific algorithm used to solve the problem.

The above work leaves open the question of whether underlying objectives or coordinate systems are simply mathematical curiosities with no practical or theoretical utility. [31] addresses that question in two ways. First, it provides an exact coordinate system extraction algorithm that, unlike the approximate algorithm of [13], is slow but is guaranteed to return a minimal-sized coordinate system for a finite interactive domain.[19] Second, it applies the exact extraction algorithm to several small instances of the game of Nim, and observes that for all instances tested, the axes of extracted coordinate systems contain significant information about how a strategy performs at the game. Thus, far from being theoretical peculiarities, at least in this case (minimal) coordinate systems intuitively relate to real structure in the domain.

Given this last fact and that certain coevolutionary algorithms seem to be sensitive to the underlying dimensions of a domain, we are led to an intriguing possibility: that appropriately-designed coevolutionary algorithms could discover and extract meaningful structure from interactive domains in the process of solving problems over them. Besides solving problems, algorithms might be able to simultaneously output useful knowledge about domains. For complex, ill-understood domains, the knowledge output might be at least as useful as a solution.

5.3 Open-Ended Evolution and Novelty

An important possibility that should not be lost to our solution-based, problem-oriented view is the idea that the goal of evolutionary systems needn't necessarily be to find a particular solution (be it a set of points or otherwise) at all. In particular, in cases where a representation space is not explicitly limited, evolutionary systems can be seen as dynamical methods to explore "interesting" spaces. Here "interesting" might correspond with some problem-oriented view or relate to aspects of some simulation that involves the evolutionary process itself, but it may also simply correspond with notions of novelty.

In traditional, non-coevolutionary systems, an objective fitness measure in conjunction with selection methods will tend to drive the systems in particular directions. Explicit construction of fitness functions that encourage some direct notion

[19] It is unclear whether interactive domains can have more than one minimal-sized coordinate system, whence *a* and not *the*.

of novelty can certainly be developed; however, coevolution offers some very natural and interesting possibilities here. Indeed, some of the earliest forms of artificial evolution involved game-playing as means of exploring some of basic mechanisms of evolution and self-maintenance themselves [5].

Many of the concepts discussed above (e.g., *informativeness*, *distinctions*, etc.) are very natural measures of novelty, and algorithms that literally function by moving in directions that work to maintain these are compelling for this purpose. Even more compelling is the idea just discussed above: Since CoEAs can be used to discover geometries of comparisons, to construct and report dimensions of informativeness in a space, they are very appealing mechanisms for novelty search in open-ended evolution because they may be capable of providing much more than new, unusual individuals — they may well be able to place such individuals in relationship with things already seen.

References

1. Angeline, P.J., Pollack, J.B.: Competitive environments evolve better solutions for complex tasks. In: Proceedings of the 5th International Conference on Genetic Algorithms ICGA-1993, pp. 264–270 (1993)
2. Axelrod, R.: The evolution of strategies in the iterated prisoner's dilemma. In: L. Davis (ed.) Genetic Algorithms and Simulated Annealing, pp. 32–41. Morgan Kaufmann (1989)
3. Bader-Natal, A., Pollack, J.B.: A population-differential method of monitoring success and failure in coevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004, *Lecture Notes in Computer Science*, vol. 3102, pp. 585–586. Springer Verlag (2004)
4. Barbosa, H.: A coevolutionary genetic algorithm for constrained optimization. In: Proceedings of the Congress on Evolutionary Computation, CEC-1999. IEEE Press (1999)
5. Barricelli, N.: Numerical testing of evolution theories. Part II. Preliminary tests of performance. Symbiogenesis and terrestrial life. Acta Biotheoretica **16**(3–4), 99–126 (1963)
6. Blumenthal, H.J., Parker, G.B.: Punctuated anytime learning for evolving multi-agent capture strategies. In: Proceedings of the Congress on Evolutionary Computation, CEC-2004. IEEE Press (2004)
7. Branke, J., Rosenbusch, J.: New approaches to coevolutionary worst-case optimization. In: Parallel Problem Solving from Nature, PPSN-X, *Lecture Notes in Computer Science*, vol. 5199. Springer-Verlag (2008)
8. Bucci, A.: Emergent geometric organization and informative dimensions in coevolutionary algorithms. Ph.D. thesis, Michtom School of Computer Science, Brandeis University, Waltham, MA (2007)
9. Bucci, A., Pollack, J.B.: Order-theoretic analysis of coevolution problems: Coevolutionary statics. In: W.B. Langdon, *et al.* (eds.) Genetic and Evolutionary Computation Conference Workshop: Understanding Coevolution. Morgan Kaufmann (2002)
10. Bucci, A., Pollack, J.B.: Focusing versus intransitivity: geometrical aspects of coevolution. In: E. Cantu-Paz, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2003. Springer (2003)
11. Bucci, A., Pollack, J.B.: A mathematical framework for the study of coevolution. In: K.A. De Jong, *et al.* (eds.) Foundations of Genetic Algorithms Workshop VII, pp. 221–235. Morgan Kaufmann, San Francisco, CA (2003)
12. Bucci, A., Pollack, J.B.: On identifying global optima in cooperative coevolution. In: H.G. Beyer, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005. ACM Press, New York, NY (2005)

13. Bucci, A., Pollack, J.B., de Jong, E.D.: Automated extraction of problem structure. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004, *Lecture Notes in Computer Science*, vol. 3102. Springer Verlag (2004)
14. Bull, L.: Evolutionary computing in multi-agent environments: Partners. In: T. Baeck (ed.) Proceedings of the 7th International Conference on Genetic Algorithms, pp. 370–377. Morgan Kaufmann (1997)
15. Bull, L.: Evolutionary computing in multi-agent environments: Operators. In: D. Wagen, A.E. Eiben (eds.) Proceedings of the 7th International Conference on Evolutionary Programming, pp. 43–52. Springer Verlag (1998)
16. Bull, L.: On coevolutionary genetic algorithms. Soft Computing **5**(3), 201–207 (2001)
17. Bull, L.: Coevolutionary species adaptation genetic algorithms: A continuing saga on coupled fitness landscapes. In: M. Capcarrere, *et al.* (eds.) Proceedings of the 8th European Conference on Advances in Artificial Life, ECAL-2005, pp. 845–853. Springer (2005)
18. Bull, L.: Coevolutionary species adaptation genetic algorithms: growth and mutation on coupled fitness landscapes. In: Proceedings of the Congress on Evolutionary Computation, CEC-2005. IEEE (2005)
19. Cartlidge, J., Bullock, S.: Learning lessons from the common cold: How reducing parasite virulence improves coevolutionary optimization. In: Proceedings of the Congress on Evolutionary Computation, CEC-2002, pp. 1420–1425. IEEE (2002)
20. Cartlidge, J., Bullock, S.: Caring versus sharing: How to maintain engagement and diversity in coevolving populations. In: W. Banzhaf, *et al.* (eds.) Proceedings of the 7th European Conference on Advances in Artificial Life, ECAL-2003, *Lecture Notes in Computer Science*, vol. 2801, pp. 299–308. Springer (2003)
21. Cartlidge, J., Bullock, S.: Combating coevolutionary disengagement by reducing parasite virulence. Evolutionary Computation **12**(2), 193–222 (2004)
22. Cartlidge, J., Bullock, S.: Unpicking tartan ciao plots: Understanding irregular coevolutionary cycling. Adaptive Behavior **12**(2), 69–92 (2004)
23. Chellapilla, K., Fogel, D.B.: Evolving neural networks to play checkers without expert knowledge. IEEE Transactions on Neural Networks **10**(6), 1382–1391 (1999)
24. Cliff, D., Miller, G.F.: Tracking the red queen: Measurements of adaptive progress in coevolutionary simulations. In: Proceedings of the 3rd European Conference on Advances in Artificial Life, ECAL-1995, pp. 200–218 (1995)
25. De Jong, E.D.: The incremental Pareto-coevolution archive. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004, *Lecture Notes in Computer Science*, vol. 3102, pp. 525–536. Springer Verlag (2004)
26. De Jong, E.D.: Intransitivity in coevolution. In: X. Yao, *et al.* (eds.) Parallel Problem Solving from Nature, PPSN-VIII, *Lecture Notes in Computer Science*, vol. 3242, pp. 843–851. Birmingham, UK (2004)
27. De Jong, E.D.: Towards a bounded Pareto-coevolution archive. In: Proceedings of the Congress on Evolutionary Computation, CEC-2004, pp. 2341–2348. IEEE Press (2004)
28. De Jong, E.D.: The MaxSolve algorithm for coevolution. In: H.G. Beyer, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005. ACM Press, New York, NY (2005)
29. De Jong, E.D.: Objective fitness correlation. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2007, pp. 440–447. ACM Press, New York, NY (2007)
30. De Jong, E.D., Bucci, A.: DECA: Dimension extracting coevolutionary algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2006. ACM Press, New York, NY (2006)
31. De Jong, E.D., Bucci, A.: Multi-Objective Problem Solving From Nature: From Concepts to Applications, chap. Objective Set Compression: Test-based Problems and Multi-objective Optimization. Natural Computing Series. Springer-Verlag, Berlin (2007)
32. De Jong, E.D., Pollack, J.B.: Ideal evaluation from coevolution. Evolutionary Computation Journal **12**(2), 159–192 (2004)
33. De Jong, K.A.: Genetic algorithms are not function optimizers. In: L.D. Whitley (ed.) Foundations of Genetic Algorithms II, pp. 5–17. Morgan Kaufmann (1992)

34. Ficici, S.G.: Solution concepts in coevolutionary algorithms. Ph.D. thesis, Brandeis University Department of Computer Science, Waltham, MA (2004)
35. Ficici, S.G., Pollack, J.B.: Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In: C. Adami, *et al.* (eds.) Artificial Life VI Proceedings, pp. 238–247. The MIT Press, Cambridge, MA (1998)
36. Ficici, S.G., Pollack, J.B.: Effects of finite populations on evolutionary stable strategies. In: D. Whitley, *et al* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2000, pp. 880–887. Morgan Kaufmann (2000)
37. Ficici, S.G., Pollack, J.B.: Game–theoretic investigation of selection methods used in evolutionary algorithms. In: D. Whitley (ed.) Proceedings of the 2000 Congress on Evolutionary Computation, pp. 880–887. IEEE Press (2000)
38. Ficici, S.G., Pollack, J.B.: Pareto optimality in coevolutionary learning. In: Proceedings of the 6th European Conference on Advances in Artificial Life, ECAL-2001, pp. 316–325. Springer-Verlag, London, UK (2001)
39. Ficici, S.G., Pollack, J.B.: A game-theoretic memory mechanism for coevolution. In: E. Cantu-Paz, *et al.* (eds.) Genetic and Evolutionary Computation Conference, GECCO-2003, pp. 286–297. Springer (2003)
40. Floreano, D., Nolfi, S.: God save the red queen! competition in co-evolutionary robotics. In: J.R. Koza, *et al.* (eds.) Proceedings of the 2nd Genetic Programming Conference, GP-1997, pp. 398–406. Morgan Kaufmann, San Francisco, CA (1997)
41. Friedman, D.: On economic applications of evolutionary game theory. Journal of Evolutionary Economics **8**, 15–43 (1998)
42. Funes, P., Pollack, J.B.: Measuring progress in coevolutionary competition. In: From Animals to Animats 6: Proceedings of the 6th International Conference on Simulation of Adaptive Behavior, pp. 450–459. MIT Press (2000)
43. Funes, P., Pujals, E.: Intransitivity revisited coevolutionary dynamics of numbers games. In: Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO-2005, pp. 515–521. ACM Press, New York, NY (2005)
44. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. In: CNLS '89: Proceedings of the 9th International Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks on Emergent Computation, pp. 228–234. North-Holland Publishing Co. (1990)
45. Hofbauer, J., Sigmund, K.: Evolutionary Games and Population Dynamics. Cambridge University Press (1998)
46. Horn, J.: The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL (1995)
47. Husbands, P., Mill, F.: Simulated coevolution as the mechanism for emergent planning and scheduling. In: R. Belew, L. Booker (eds.) Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 264–270. Morgan Kaufmann (1991)
48. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. Information Processing Letters **86**(6), 317–321 (2003)
49. Jansen, T., Wiegand, R.P.: Exploring the explorative advantage of the CC (1+1) EA. In: Proceedings of the 2003 Genetic and Evolutionary Computation Conference. Springer (2003)
50. Jansen, T., Wiegand, R.P.: Sequential versus parallel cooperative coevolutionary (1+1) EAs. In: Proceedings of the Congress on Evolutionary Computation, CEC-2003. IEEE (2003)
51. Jansen, T., Wiegand, R.P.: The cooperative coevolutionary (1+1) EA. Evolutionary Computation **12**(4), 405–434 (2004)
52. Jensen, M.T.: Robust and flexible scheduling with evolutionary computation. Ph.D. thesis, Department of Computer Science, University of Aarhus, Denmark (2001)
53. Juillé, H., Pollack, J.B.: Coevolving the ideal trainer: Application to the discovery of cellular automata rules. In: J.R. Koza, *et al.* (eds.) Proceedings of the 3rd Genetic Programming Conference, GP-1998, pp. 519–527. Morgan Kaufmann (1998)

54. Kauffman, S., Johnson, S.: Co-evolution to the edge of chaos: coupled fitness landscapes, poised states and co-evolutionary avalanches. In: C. Langton, *et al.* (eds.) Artificial Life II Proceedings, vol. X, pp. 325–369. Addison-Wesley (1991)
55. Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Transactions on Evolutionary Computation **8**(2), 170–182 (2004)
56. Liekens, A., Eikelder, H., Hilbers, P.: Finite population models of co-evolution and their application to haploidy versus diploidy. In: E. Cantú-Paz, *et al* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2003, pp. 344–355. Springer (2003)
57. Luke, S., Wiegand, R.P.: Guaranteeing coevolutionary objective measures. In: K.A. De Jong, *et al.* (eds.) Foundations of Genetic Algorithms VII, pp. 237–251. Morgan Kaufman (2003)
58. Maynard Smith, J.: Evolution and the Theory of Games. Cambridge University Press (1982)
59. Miller, J.H.: The coevolution of automata in the repeated prisoner's dilemma. Journal of Economic Behavior and Organization **29**(1), 87–112 (1996)
60. Monroy, G.A., Stanley, K.O., Miikkulainen, R.: Coevolution of neural networks using a layered Pareto archive. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2006, pp. 329–336. ACM Press, New York, NY (2006)
61. Noble, J., Watson, R.A.: Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection. In: L. Spector, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, pp. 493–500. Morgan Kaufmann, San Francisco, CA (2001)
62. Oliehoek, F.A., de Jong, E.D., Vlassis, N.: The parallel Nash Memory for asymmetric games. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2006, pp. 337–344. ACM Press, New York, NY (2006)
63. Oliveto, P., He, J., Yao, X.: Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. International Journal of Automation and Computing **04**(3) (2007)
64. Olsson, B.: Co-evolutionary search in asymmetric spaces. Information Sciences **133**(3-4), 103–125 (2001)
65. Osborne, M.J., Rubinstein, A.: A course in game theory. The MIT Press, Cambridge, MA (1994)
66. Pagie, L., Hogeweg, P.: Information integration and red queen dynamics in coevolutionary optimization. In: Proceedings of the Congress on Evolutionary Computation, CEC-2000, pp. 1260–1267. IEEE Press, Piscataway, NJ (2000)
67. Pagie, L., Mitchell, M.: A comparison of evolutionary and coevolutionary search. International Journal of Computational Intelligence and Applications **2**(1), 53–69 (2002)
68. Panait, L.: The analysis and design of concurrent learning algorithms for cooperative multiagent systems. Ph.D. thesis, George Mason University, Fairfax, VA (2006)
69. Panait, L., Luke, S.: A comparison of two competitive fitness functions. In: W.B. Langdon, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002, pp. 503–511. Morgan Kaufmann (2002)
70. Panait, L., Luke, S.: Time-dependent collaboration schemes for cooperative coevolutionary algorithms. In: AAAI Fall Symposium on Coevolutionary and Coadaptive Systems. AAAI Press (2005)
71. Panait, L., Luke, S.: Selecting informative actions improves cooperative multiagent learning. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2006. ACM Press, New York, NY (2006)
72. Panait, L., Luke, S., Harrison, J.F.: Archive-based cooperative coevolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2006. ACM Press, New York, NY (2006)
73. Panait, L., Luke, S., Wiegand, R.P.: Biasing coevolutionary search for optimal multiagent behaviors. IEEE Transactions on Evolutionary Computation **10**(6), 629–645 (2006)

74. Panait, L., Sullivan, K., Luke, S.: Lenience towards teammates helps in cooperative multiagent learning. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS-2006. ACM Press, New York, NY (2006)
75. Panait, L., Wiegand, R.P., Luke, S.: Improving coevolutionary search for optimal multiagent behaviors. In: G. Gottlob, T. Walsh (eds.) Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI-2003, pp. 653–658 (2003)
76. Panait, L., Wiegand, R.P., Luke, S.: A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004, *Lecture Notes in Computer Science*, vol. 3102. Springer Verlag (2004)
77. Paredis, J.: Coevolving cellular automata: Be aware of the red queen. In: T. Bäck (ed.) Proceedings of the 7th International Conference on Genetic Algorithms, ICGA-1997. Morgan Kaufmann, San Francisco, CA (1997)
78. Parker, G.B., Blumenthal, H.J.: Comparison of sample sizes for the co-evolution of cooperative agents. In: Proceedings of the Congress on Evolutionary Computation, CEC-2003. IEEE (2003)
79. Popovici, E.: An analysis of two-population coevolutionary computation. Ph.D. thesis, George Mason University, Fairfax, VA (2006)
80. Popovici, E., De Jong, K.A.: Understanding competitive co-evolutionary dynamics via fitness landscapes. In: S. Luke (ed.) AAAI Fall Symposium on Artificial Multiagent Learning. AAAI Press (2004)
81. Popovici, E., De Jong, K.A.: A dynamical systems analysis of collaboration methods in cooperative co-evolution. In: AAAI Fall Symposium Series Co-evolution Workshop (2005)
82. Popovici, E., De Jong, K.A.: Relationships between internal and external metrics in coevolution. In: Proceedings of the Congress on Evolutionary Computation, CEC-2005. IEEE (2005)
83. Popovici, E., De Jong, K.A.: Understanding cooperative co-evolutionary dynamics via simple fitness landscapes. In: H.G. Beyer, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005. ACM Press, New York, NY (2005)
84. Popovici, E., De Jong, K.A.: Monotonicity versus performance in co-optimization. In: Foundations of Genetic Algorithms X. ACM Press, New York, NY (2009). (to appear)
85. Potter, M.: The design and analysis of a computational model of cooperative coevolution. Ph.D. thesis, George Mason University, Computer Science Department (1997)
86. Potter, M., De Jong, K.A.: A cooperative coevolutionary approach to function optimization. In: Parallel Problem Solving from Nature, PPSN-III, pp. 249–257. Springer, Jerusalem, Israel (1994)
87. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1), 1–29 (2000)
88. Rosin, C.D.: Coevolutionary search among adversaries. Ph.D. thesis, University of California, San Diego, CA (1997)
89. Rosin, C.D., Belew, R.K.: Methods for competitive co-evolution: Finding opponents worth beating. In: Proceedings of the 6th International Conference on Genetic Algorithms, ICGA-1995, pp. 373–381. Morgan Kaufmann (1995)
90. Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. Evolutionary Computation **5**(1), 1–29 (1997)
91. Schmitt, L.M.: Coevolutionary convergence to global optima. In: E. Cantu-Paz, *et al.* (eds.) Genetic and Evolutionary Computation Conference, GECCO-2003, pp. 373–374. Springer (2003)
92. Schmitt, L.M.: Theory of coevolutionary genetic algorithms. In: M. Guo, *et al.* (eds.) Parallel and Distributed Processing and Applications, International Symposium, ISPA-2003, pp. 285–293. Springer (2003)
93. Schumacher, C., Vose, M., Whitley, L.: The no free lunch and description length. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, pp. 565–570. Morgan Kaufmann (2001)

94. Service, T.C.: Unbiased coevolutionary solution concepts. In: Foundations of Genetic Algorithms X. ACM Press, New York, NY (2009). (to appear)
95. Service, T.C., Tauritz, D.R.: Co-optimization algorithms. In: M. Keijzer, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2008, pp. 387–388. ACM Press, New York, NY (2008)
96. Service, T.C., Tauritz, D.R.: A no-free-lunch framework for coevolution. In: M. Keijzer, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2008, pp. 371–378. ACM Press, New York, NY (2008)
97. Sims, K.: Evolving 3D morphology and behaviour by competition. In: R. Brooks, P. Maes (eds.) Artificial Life IV Proceedings, pp. 28–39. MIT Press, Cambridge, MA (1994)
98. Spears, W.: Simple subpopulation schemes. In: Proceedings of the 1994 Evolutionary Programming Conference. World Scientific (1994)
99. Stanley, K.O.: Efficient evolution of neural networks through complexification. Ph.D. thesis, The University of Texas at Austin, Austin, Texas (2004)
100. Stanley, K.O., Miikkulainen, R.: Continual coevolution through complexification. In: W.B. Langdon, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002, pp. 113–120. Morgan Kaufmann, San Francisco, CA (2002)
101. Stanley, K.O., Miikkulainen, R.: The dominance tournament method of monitoring progress in coevolution. In: W.B. Langdon, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002. Morgan Kaufmann (2002)
102. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research **21**, 63–100 (2004)
103. Stuermer, P., Bucci, A., Branke, J., Funes, P., Popovici, E.: Analysis of coevolution for worst-case optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2009. ACM Press, New York, NY (2009). (to appear)
104. Subbu, R., Sanderson, A.: Modeling and convergence analysis of distributed coevolutionary algorithms. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **34**(2), 806–822 (2004)
105. Vose, M.: The Simple Genetic Algorithm. MIT Press (1999)
106. Watson, R.A., Pollack, J.B.: Coevolutionary dynamics in a minimal substrate. In: L. Spector, *et al.* (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, pp. 702–709. Morgan Kaufmann, San Francisco, CA (2001)
107. Weibull, J.: Evolutionary game theory. MIT Press, Cambridge, MA (1992)
108. Wiegand, R.P.: An analysis of cooperative coevolutionary algorithms. Ph.D. thesis, George Mason University, Fairfax, VA (2004)
109. Wiegand, R.P., Liles, W., De Jong, K.A.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: L. Spector (ed.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, pp. 1235–1242. Morgan Kaufmann (2001). Errata available at http://www.tesseract.org/paul/papers/gecco01-cca-errata.pdf
110. Wiegand, R.P., Liles, W.C., De Jong, K.A.: The effects of representational bias on collaboration methods in cooperative coevolution. In: Proceedings of the 7th Conference on Parallel Problem Solving from Nature, pp. 257–268. Springer (2002)
111. Wiegand, R.P., Potter, M.: Robustness in cooperative coevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2006. ACM Press, New York, NY (2006)
112. Wiegand, R.P., Sarma, J.: Spatial embedding and loss of gradient in cooperative coevolutionary algorithms. In: X. Yao, *et al.* (eds.) Parallel Problem Solving from Nature, PPSN-VIII, pp. 912–921. Springer, Birmingham, UK (2004)
113. Williams, N., Mitchell, M.: Investigating the success of spatial coevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005, pp. 523–530. ACM Press, New York, NY (2005)
114. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
115. Wolpert, D., Macready, W.: Coevolutionary free lunches. IEEE Transactions on Evolutionary Computation **9**(6), 721–735 (2005)