Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm

A Dissertation

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

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

by

Hugues Juillé

May, 1999

This dissertation, directed and approved by Hugues Juillé's Committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Dean of Arts and Sciences

Dissertation Committee

Prof. Jordan B. Pollack, Chair

Prof. Martin Cohn

Prof. Kenneth A. De Jong

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor, Professor Jordan B. Pollack. When he arrived at Brandeis, four years ago, I was just starting my Ph.D. program. He offered me an opportunity to address some fascinating problems and gave me the intellectual latitude to perform fundamental research. His scientific knowledge and creativity were a continual source of inspiration. Those years spent working with him were a very challenging and enlightening experience.

I would also like to thank Professor Martin Cohn. Without him, I might not even have entered a Ph.D. program. He was always available to listen to me and to give me some advice. His theoretical background was extremely helpful to my work. Today, I am very grateful to him and my advisor for putting their trust in me.

I am very honored that Professor Kenneth A. De Jong accepted to serve on my thesis committee and took the time to review my dissertation during a very busy period. His comments were very valuable for improving the quality of this dissertation.

I also wish to thank the Office of Naval Research for providing support for this work under grant N00014-96-1-0416.

I wish to give special thanks to my fellows at the DEMO lab: Alan Blair, Paul Darwen, Sevan Ficici, Pablo Funes, Greg Hornby, Simon Levy, Ofer Melnik, Elizabeth Sklar and Richard Watson. The lab was an environment in perpetual change and provided an inexhaustible source of new ideas, discussions and lively group meetings.

Thanks also to all the members of the computer science department and all the people who helped me during my thesis with their scientific insights. In particular, I would like to thank Professor Melanie Mitchell. My visit to the Santa Fe Institute in June 1998 was very exciting and the source of many interesting discussions with her and other researchers there.

Anne, my wife, was a wonderful source of moral support. She had faith in me from the beginning, and she was always here during periods of doubt and uncertainty. Her smile and her words have always brought so much to me.
Our lives were blessed when Marie was born on September 24, 1997. She gave me the motivation and strength to finish this thesis!

A mes parents qui ont toujours su par leur confiance me motiver pour poursuivre mes études et persévérer dans mes entreprises, je témoigne toute ma gratitude. Leur enseignement et leur exemple m'ont beaucoup apporté dans la vie.
Paul, je voudrais te dire aujourd'hui à quel point ton amitié a également été une importante source de soutien.
Enfin, je souhaite remercier toute ma famille et belle-famille en France et à Londres. Vous avez toujours eu confiance en moi et mon succès d'aujourd'hui est du en grande partie à vos encouragements continus.

Hugues JUILLE

ABSTRACT

Methods for Statistical Inference:

Extending the Evolutionary Computation Paradigm

A dissertation presented to the Faculty of

the Graduate School of Arts and Sciences of

Brandeis University, Waltham, Massachusetts

by Hugues Juillé

In many instances, Evolutionary Computation (EC) techniques have demonstrated their ability to tackle ill-structured and poorly understood problems against which traditional Artificial Intelligence (AI) search algorithms fail. The principle of operation behind EC techniques can be described as a statistical inference process which implements a sampling-based strategy to gather information about the state space, and then exploits this knowledge for controlling search. However, this statistical inference process is supported by a rigid structure that is an integral part of an EC technique. For instance, *schemas* seem to be the basic components that form this structure in the case of Genetic Algorithms (GAs). Therefore, it is important that the encoding of a problem in an EC framework exhibits some regularities that correlate with this underlying structure. Failure to find an appropriate representation prevents the evolutionary algorithm from making accurate decisions. This dissertation introduces new methods that exploit the same principles of operation as those embedded in EC techniques and provide more flexibility for the choice of the structure supporting the statistical inference process. The purpose of those methods is to generalize the EC paradigm, thereby expanding its domain of applications to new classes of problems.

Two techniques implementing those methods are described in this work. The first one, named SAGE, extends the sampling-based strategy underlying evolutionary algorithms to perform search in trees and directed acyclic graphs. The second technique considers coevolutionary learning, a paradigm which involves the embedding of adaptive agents in a fitness environment that dynamically responds to their progress. Coevolution is proposed as a framework in which evolving agents would be permanently challenged, eventually resulting in continuous improvement of their performance. After identifying obstacles to continuous progress, the concept of an "Ideal" trainer is presented as a paradigm which successfully achieves that goal by maintaining a pressure toward adaptability.

The different algorithms discussed in this dissertation have been applied to a variety of difficult problems in learning and combinatorial optimization. Some significant achievements that resulted from those experiments concern: (1) the discovery of new constructions for 13-input sorting networks using fewer comparators than the best known upper bound, (2) an improved procedure for the induction of DFAs from sparse training data which ended up as a co-winner in a grammar inference competition, and (3) the discovery of new cellular automata rules to implement the majority classification task which outperform the best known rules.

By describing evolutionary algorithms from the perspective of statistical inference techniques, this research work contributes to a better understanding of the underlying search strategies embedded in EC techniques. In particular, an extensive analysis of the coevolutionary paradigm identifies two fundamental requirements for achieving continuous progress. Search and machine learning are two fields that are closely related. This dissertation emphasizes this relationship and demonstrates the relevance of the issue of generalization in the context of coevolutionary races.

Contents

List of Figures

Chapter 1

Introduction

For many tasks in Artificial Intelligence (AI), problem solving is an activity that exploits search as the fundamental paradigm in order to isolate from a set of candidates a solution that exhibits some specific properties. As more difficult problems are addressed, the size of this search space grows quickly and it becomes important to take advantage of the available knowledge about the structural properties of the problem domain. This concept, formalized in the so-called "No Free Lunch" theorems [114], motivates the design of strong methods which embed such problem-specific knowledge in order to make search tractable. However, for many ill-structured and poorly understood problems, the knowledge available about the regularities of the problem domain concerns empirical observations, rules of thumb or intuitions. Efforts to formalize this knowledge in the framework of traditional search paradigms are often unsuccessful because this knowledge is better described in terms of *statistical properties* of the problem domain. This is the reason for the success of Evolutionary Computation (EC) techniques, like Genetic Algorithms (GAs), which seem more amenable to exploit this form of knowledge. The principle of operation behind Evolutionary Algorithms (EA) is to gather statistics about the structure of the state space and then to exploit that knowledge to control search. In the case of Genetic Algorithms, it is well accepted that this information gathering process is performed with respect to a specific structure constructed from *schemas* [39, 58]. Researchers are working on extending the schema theory to describe the search strategies embedded in other EC techniques like Genetic Programming (GP) [50, 51, 67, 78]. Therefore, when encoding a problem in the framework of EC techniques, it is important that the regularities of the state space correlate with the structure supporting the knowledge gathering process in the evolutionary algorithm. This is a necessary condition for the EA to exploit those regularities efficiently to drive search. The central contribution of this dissertation is to generalize this statistical inference strategy by proposing new methodologies that capture the same principles of operation as those embedded in evolutionary algorithms, thereby expanding their domain of applications. More precisely, the fundamental idea of this research work is to propose a framework for the representation and the exploitation of statistical properties of a problem domain that is more flexible than the rigid structures, like schemas, that underlie search in EAs. Since any sampling-based search strategy captures some specific statistical properties, this means that the techniques introduced in this dissertation are associated with specific classes of problems. The properties and features associated with those classes will be discussed in detail. By producing multiple tangible results that involved large instances of difficult combinatorial and inductive learning problems and resulted in significant improvements, this research work demonstrates the efficiency and flexibility of those methodologies. Moreover, by considering EC techniques from the perspective of algorithms that exploit statistical properties of a search space, this dissertation contributes to a better understanding of the search strategies embedded in those techniques.

The purpose of the following sections is to offer to the reader a background on the foundations of the fields of heuristic search and evolutionary search. The progression in those sections illustrates the different stages that lead to the research work presented in this dissertation. The purpose of the next chapter is to describe more formally how the methodologies introduced in this research work extend the EC paradigm by proposing two different frameworks to capture statistical properties of problem domains. The technical contribution of this dissertation consists in the description of new evolutionary-inspired search algorithms based on those frameworks. Since any search algorithm is associated with some specific classes of problems, the fundamental features that correlate with the statistical inference process implemented in those new algorithms are also discussed in this second chapter. In order to demonstrate the efficiency of the methods introduced in this dissertation, they have been evaluated against large instances of difficult combinatorial optimization and inductive learning problems. Many of those experiments performed against well-studied benchmark problems resulted in significant improvements.

1.1 Problem Solving is about Designing Abstract Objects

Problem solving involves the design of abstract objects that represent some solutions to a task. A fundamental activity which is inherent in problem solving concerns the search of a solution to the problem under consideration in a large space of candidate solutions. The formalization of this process can be decomposed into three components:

1. A representation to encode those objects.

2. A set of transformation operators that act on the representation of objects in order to construct new objects. The role of those transformation operators is to explore the space of candidate solutions.

3. A search strategy to explore the state space defined by the previous two elements in order to discover a solution that satisfies some desired properties (like the maximization of a criterion). The choices performed by this strategy are based, in particular, on the definition of an objective function used to evaluate and compare solutions.

The representation and transformation operators define a neighborhood structure over the space of solutions while the search strategy determines how this space is explored. One goal of the field of Artificial Intelligence is to provide methods to explore efficiently such spaces of candidate solutions. General purpose tree search algorithms like depth and breadth-first search or learning algorithms specific to a particular problem domain like classification are examples of such methods.

1.2 Efficient Problem Solving Means Exploiting Constraints and Regularities of the Problem Domain

In practice, there will always be problems that are just beyond our reach. However, as more knowledge is available about the properties of those problems and as more computational resource becomes available their status evolves and some of them become solved. The games of checkers and reversi are instances for which the amount of knowledge that has been gathered through experience and analysis allowed the implementation of strategies that achieve world class performance. The recent success in cracking some cryptographic codes using thousands of computers connected on the Internet is also a good illustration of this evolution in time of our ability to solve increasingly difficult problems.

However, as problems of increasing difficulty are addressed, the size of the space of candidate solutions grows quickly. In order for search algorithms to scale up, it

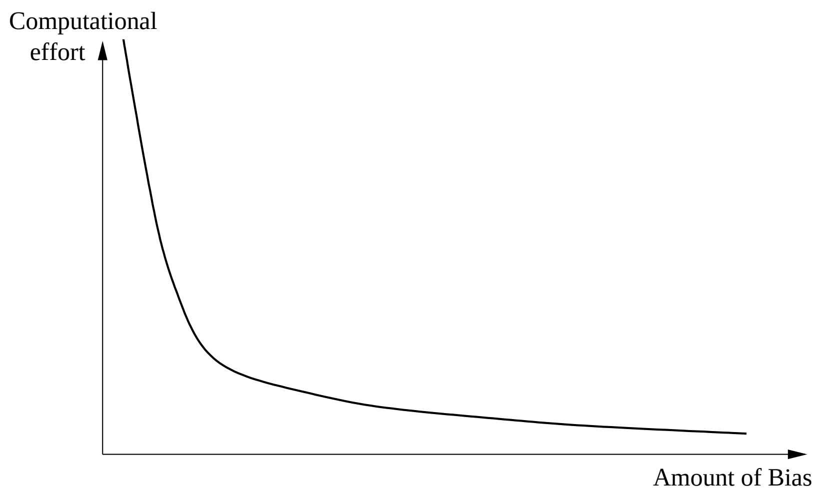


Figure 1.1: Distribution between the amount of computational effort and the amount of bias that are necessary to solve a specific problem.

is important to exploit the available knowledge about the regularities of the problem domain. That is, a search algorithm must be able to deduce information about unvisited domains of the space of candidate solutions from the evaluation of the samples that have been gathered. This process is possible only if the mechanisms that are embedded in the search algorithm correlate with the structural properties of the problem domain. Or, put in another way, the decision procedures embedded in the search algorithm must make informed choices with respect to the nature of the problem in order to limit the amount of effort involved in search. General purpose search strategies won't be able to achieve good performance in large state spaces without the introduction of problem specific knowledge. This concept is known as the *knowledge-search trade-off* and is illustrated with the diagram in Figure 1.1 where the notion of bias corresponds to the explicit encoding of knowledge about the problem. This notion has also been formalized in the so-called "No Free Lunch" theorems [114]. Put in a simple way, this theoretical result states that the operators of a search algorithm must correlate with the features associated with the intrinsic properties of the problem domain if this algorithm is to do better than random search. As a result, the goal of a general purpose search algorithm is in fact to propose a framework in which the knowledge about a problem domain can be expressed easily in a form that can be exploited efficiently by the search mechanisms embedded in that algorithm.

Therefore, solving increasingly difficult problems means that some properties about the problem domain must be identified and the search procedure must be able to exploit those regularities.

1.3 Manipulating Subsets of Candidate Solutions as a Search Paradigm

In response to the observation discussed in the previous section, most traditional Artificial Intelligence search algorithms consider a representation scheme for problems that involves the definition of a *structure* over the state space, along with heuristics as simple decision procedures to control search. The purpose of this structure is to provide the decision procedure with meaningful information about the structural properties of the problem domain.

In particular, a fundamental strategy followed by many AI search algorithms in order to capture structural properties of a problem domain is based on the ability to manipulate subsets of candidate solutions. Indeed, manipulating subsets of candidate solutions makes possible the exploitation of structural properties providing those subsets capture relevant information. A common example to illustrate this concept concerns a telephone directory. If one looks for an entry in this directory in order to find a street address and he is only given a phone number then, on average, no strategy is better than going through the entire directory until the phone number is found. However, if the name of the person is also given then the search becomes very simple by using a binary search strategy. By exploiting the ordering by names of the entries in the telephone directory, it is possible to eliminate at each step of the search all the entries that are before or after the current entry depending whether it is before or after the searched name in lexical order.

This simple example also illustrates that the different subsets that are under consideration must be meaningful. That is, they should capture some relevant information that can be exploited by the search procedure. Here, two subsets are considered at each step of the binary search, each of them corresponding to the sequence of entries in the directory that are before or after the current entry with respect to the lexical order. Then, a simple computation is performed in order to determine which subset should be conserved to continue the search. This example illustrates that this paradigm allows the elimination of large domains of the state space and efficiently drives search toward the goal solution.

However, in many problem domains, the search space is not as well structured as a telephone directory and the design of simple rules that drive efficiently the search toward the goal is not always possible. For such problems, the objective of the search procedure often shifts from the discovery of optimum solutions to the discovery of *satisficing* solutions. That is, solutions that provide a good trade-off between the computational cost to discover them and their quality. For such problems, the subset paradigm still provides an attractive framework to exploit some of the properties of the problem domain. In this framework, the decision procedures that exploit the structure defined over the search space by those subsets implement *heuristics* [75]. The purpose of heuristics is to capture some meaningful properties of the problem domain such that satisficing solutions can be discovered and the amount of computational effort involved in this process remains reasonable.

In order to illustrate the different issues relevant to the subset manipulation paradigm, consider the n-queens problem. The goal of this problem is to place n queens on a $n \times n$ chess board such that no two queens attack each other. Two basic approaches can be examined for the representation of this problem. The first one considers the space of all possible assignments of the n queens on the board. However,while being able to represent any valid solutions to the problem, most of the objects in this space correspond to non-valid solutions. In addition, this representation scheme doesn't provide simple transformation operators to explore efficiently the space of candidate solutions. For instance, consider the trivial transformation operator which moves a queen from its current position to another position on the board. This means that at each step, one has to make a choice among $n \times n \times n$ possible transformations (choose one among n queens and one among $n \times n$ board positions for the new assignment). This transformation operator doesn't exploit efficiently the constraints associated with the n-queens problem and, as a result, the sequence of transformations necessary to eventually reach a valid solution may have to traverse many non-valid configurations of the board.

A more efficient approach to this problem considers a sequential construction approach. That is, one first assigns a queen in the first column of the board and then iteratively assigns a queen in the next column at a position that doesn't violate any constraint. This strategy, illustrated in Figure 1.2, eliminates a large number of candidate solutions that were under consideration in the previous representation because the constraints of the problem are exploited more efficiently. However, there is still the issue of determining in which position among the unattacked ones the next queen should be assigned in order to eventually reach a configuration where all the queens have been assigned on the board. There is no trivial decision procedure like the one which was available for the telephone directory example. Instead, some heuristics have to be designed. A heuristic is a computational procedure which determines which choice among several alternatives should

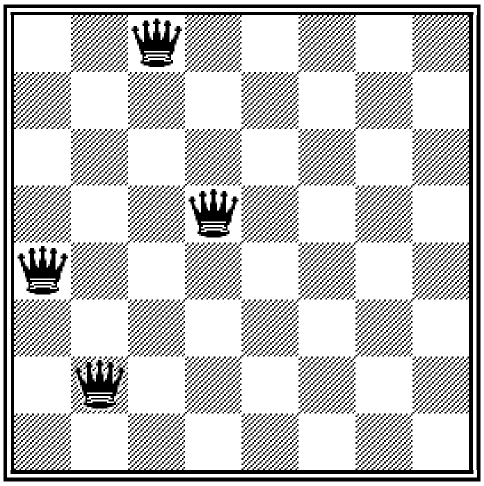


Figure 1.2: Board position for the 8-queens problem where queens have been assigned in the first four columns.

be selected to continue the search process [75]. The complexity of this computational procedure should be several orders of magnitude smaller than the complexity of finding the best solution with some systematic method. Since heuristics don't necessarily provide an exact measure but only an estimate of the true theoretical value attached to each decision, they don't always drive search in the correct direction. However, combined with some backtracking procedure, they usually result in the discovery of high performance solutions. The purpose of heuristics is to provide a trade-off between the complexity of search (in terms of the amount of computational effort involved) and the quality of the final solution. From the perspective of this second approach to the n-queens problem, the partial configuration composed of the first k assignments at a given stage of the construction procedure can be seen as a representation of the subset corresponding to all the board configurations that can be constructed from this "prefix". Then, the goal of a good heuristic is to evaluate which subset among the ones associated which each valid assignment for the $(k+1)^{th}$ queen is more likely to contain a solution representing a complete assignment for the n queens.

In summary, manipulating subsets of candidate solutions makes possible the representation and exploitation of structural properties of the problem domain. For instance, as illustrated with the n-queens problem, subsets can be used for the rapid identification of non-valid domains of the search space by exploiting efficiently the constraints associated with the problem. Subsets can also be used to define a structure over the search space. Such a structure organizes candidate solutions into a hierarchy with respect to the subset inclusion rule. The purpose of this structure is to capture some meaningful information with respect to the distribution of candidate solutions quality across the search space.

1.4 Interdependence between Heuristic and Structure Definition

From the description of the telephone directory and the n-queens examples, it appears that the design of the decision procedures (or heuristics) and the structure defined by the decomposition of the space of candidate solutions into subsets are closely intertwined. Indeed, the definition of a heuristic that allows efficient search is possible only if the subsets provide meaningful information with respect to the properties of the problem domain. That is, the structure must capture some information about the problem domain that can be exploited efficiently by the computational procedure that implements a given heuristic. In practice, the strategy for the hierarchical partitioning of the search space into subsets and the design of heuristics that exploit this partitioning comes from a good understanding by the human designer of the intrinsic properties of the problem domain.

1.5 Exploiting Statistical Properties of Problem Domains

From a general perspective, exploiting *statistical properties* of a problem domain means that valid inferences about the characteristics of the problem domain can be made from the processing of information obtained by randomly sampling the search space. If such properties of a problem domain can be accurately captured, a correct exploitation of this information is extremely useful to control a search procedure. Because the knowledge available about some problem domains is better stated in terms of statistical properties, this approach is especially relevant. This notion can be illustrated with the following application of a sampling approach to the n-queens problem. This problem has been used extensively as a standard Constraint Satisfaction Problem (CSP) benchmark for investigating the efficiency of search algorithms. Indeed, the board size provides a simple parameter to generate problem instances of increasing difficulty. This makes possible the analysis of the scalability of those search algorithms. Due to combinatorial explosion, traditional search algorithms (e.g. backtracking search) are limited to relatively small values of n (usually less than 100). Stone and Stone [102] performed an empirical study of search algorithms applied to the n-queens and provided evidence that this problem produces a tree that first expands exponentially with depth and then contracts as the branching factor rapidly decreases because the problem becomes more constrained as more queens are assigned on the board. Because the density of successful paths among this exponentially large number of initial alternatives is small, general purpose tree search algorithms are inefficient.

Now, consider an experiment in which the space of configurations for the queens on the $n \times n$ board is sampled according to the procedure described in Figure 1.3. This procedure sweeps the chess board from the first column to the right and randomly

```
/* Board size is n × n */
i = 0
do
   i  ←  i + 1
   S = list of positions in the i^th column which are not attacked
          by any of the queens assigned in columns [1 ... (i − 1)]
   if S ≠ ∅ then
      Assign a queen in the i^th column at a position selected
         uniformly randomly from S
   endif
while (S ≠ ∅)
/* i queens have been assigned on the board. */
```

Figure 1.3: Stochastic procedure to generate non-conflicting assignments of queens.

assigns a queen in a non-attacking position in each column. If at any stage in this procedure all positions are attacked in the current column, the procedure stops with an incomplete assignment. Figure 1.4 plots the evolution of the ratio of samples that resulted in a complete assignment of the n queens on the board as n increases from 100 to $1,000$. For each value of n, a total of $131,072(=2^{17})$ samples are performed. It appears that, using this simple sampling procedure for assigning queens, there is a non-negligible probability of generating a complete solution even for $n = 1,000$. On average, out of $1,000$ samples, at least one corresponds to a complete solution. The reason for this result is that the distribution of solutions, even if non uniform as it is shown by [102], is not localized in a small area of the search space. However, in the case of a deterministic tree search approach, an incorrect choice early in the search results in the exploration of an exponentially large domain of the search space that contains no solutions.

It should also be noted that, following a different approach, some large instances of the n-queens problem have been solved. For example, Sosic and Gu [101] designed a probabilistic algorithm using a form of gradient-based heuristic to find some solutions

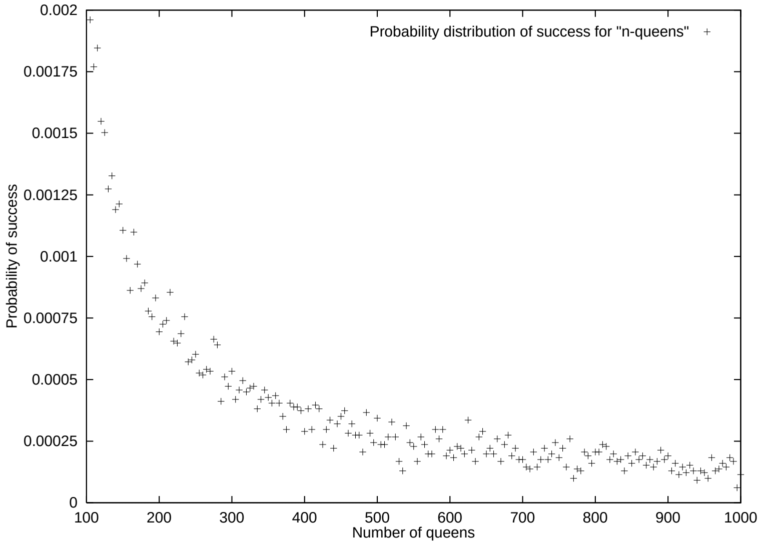


Figure 1.4: Evolution of the probability of success for generating a valid assignment for "n-queen".

for $n = 3,000,000$. Minton et al. [57] also proposed an algorithm using a min-conflicts heuristic to control the search. Using this approach, a solution with one million queens can be discovered in a few minutes. Both approaches start from an initial non-perfect solution constructed using a greedy algorithm that results in a small number of conflicts that are then resolved using a local search. This former work already shed some light that some regular structures can be exploited for that particular problem and that the density of solutions is relatively high.

This simple example exploits the property about the n-queens problem that solutions are distributed quite evenly in the search space and their density is far from negligible even for large values of n. Randomization techniques which exploit a similar idea have also been used in order to boost the performance of combinatorial optimization algorithms [35]. The underlying idea exploited by randomization techniques is that for many problems a large variance is observed for the running time of deterministic search algorithms. In some cases, the mean time to discover a solution even grows exponentially as more instances of the problem are processed. However, even for difficult problems, the expected time for discovering a solution can be considerably improved by performing multiple runs of the randomized version of the deterministic search algorithm. For such problems, the "hardness" is not intrinsic but relates to the search strategy.

The purpose of this section is to illustrate that stochastic approaches can be extremely efficient to address some problems. Usually, stochastic search algorithms are more elaborate than the simple sampling strategy implemented in this example with the n-queens problem. An iterative procedure is often involved for which the information gathered during the sampling stage is exploited to determine how to focus the search in the following stages.

An important motivation for exploiting statistical properties of problem domains as a search paradigm is that this information is sometimes the only knowledge that is available. This is particularly true for problems that are ill-structured and for which an analytical approach is extremely difficult. Defining an appropriate framework that allows the representation and the exploitation of statistical properties associated with such problems is the central issue that is addressed in this dissertation.

1.6 Evolutionary Computation as a Paradigm for Statistical Inference

Evolutionary Computation corresponds to a search paradigm which implements a simplified computational model of the mechanisms embedded in natural evolution. The central idea is to maintain a population of individuals which correspond to alternatives in a search space. A fitness is assigned to each individual with respect to some objective function. Then, following the rule of the survival of the fittest, individuals with higher fitness are selected and reproduce. The new offspring are introduced in the population and evaluated according to the current objective function. The operators involved in the reproduction stage introduce variability in the population, thereby allowing the exploration of the search space for improved solutions. Those operators perform transformations over an entity which is an abstraction of each individual. This abstraction is an encoding of the characteristics of an individual. Two principal transformation operators are usually implemented in evolutionary algorithms. The first one, called *crossover*, takes the encodings associated with the parents that reproduce and constructs a new entity by taking some pieces from each parent. The second operation, *mutation*, is usually performed after crossover and introduces some random variations in the encoding of an individual. There might be significant differences between different implementations of the EC paradigm with respect to those operators or the representation scheme used for the encoding of individuals. For instance, the field of Evolutionary Programming [28] considers only mutation as the search operator and doesn't implement crossover. In Genetic Algorithms [39], the encoding associated with each individual is called a chromosome and is represented as a string of bits, while in Genetic Programming [50, 51] a tree encoding is used to represent each individual.

1.6.1 Evolutionary Computation from the perspective of Global Random Search Methods

Evolutionary Algorithms can be described from different perspectives, depending on the particular features that one wants to emphasize. One approach considers the description of EAs in the framework of *Global Random Search* methods. The central idea in global random search methods consists in sampling iteratively the search space with respect to a probability distribution which is updated according to the evaluation of previous samples and some predefined strategy. More formally, a general scheme for global random search methods may be described with the algorithm in Figure 1.5 (from [117]), where

$\mathcal{X}$ represents the search space and $\mathbf{f} : \mathcal{X} \to \Re$ is the objective function. In typical applications of search and global optimization, the goal is to optimize the objective function $\mathbf{f}$, that is, to determine a point $x^* \in \mathcal{X}$ for which $\mathbf{f}(x^*)$ approximates the value of:

$$\mathbf{f}^* = \inf_{x \in \mathcal{X}} \mathbf{f}(x)$$

if the problem is to determine a global minimizer, or

$$\mathbf{f}^* = \sup_{x \in \mathcal{X}} \mathbf{f}(x)$$

if the problem is to determine a global maximizer.

```
1. Let P_1 be a probability distribution on X and set k = 1.
2. Generate N_k samples from X according to P_k:   S_k = {x_{1,k}, ..., x_{N_k,k}}
   Evaluate f for each of these samples.
3. Construct a new probability distribution P_{k+1} on X according to
     a fixed algorithm.
4. Check the stopping criterion.  If the algorithm doesn't terminate,
     let k <- k + 1 and return to step 2.
```

Figure 1.5: Formal scheme for global random search methods.

The algorithm that constructs the new probability distribution P_{k+1} can be more or less elaborate, depending on the underlying heuristics implemented by the global random search method. In the simplest instances, also called *passive* methods, this algorithm keeps the probability distribution unchanged ($P_{k+1} = P_k$). If P_k is the uniform distribution, this is pure random search. In *adaptive* methods, this algorithm exploits the information gathered from the different sampling stages to update the probability distribution. The algorithm implements a predetermined strategy whose design is usually based on some problem-specific knowledge.

In the framework of global random search methods, the state of the population maintained in a given EC implementation at the k^{th} generation corresponds to the set of samples $\mathcal{S}_k$. Following the same analogy, the search operators embedded in that EC implementation and the dynamics of the population evolution can be described in terms of strategies to control the evolution of the probability distribution $\{P_k\}$. For instance, Peck [77, 76] followed that methodology to analyze convergence conditions in Genetic Algorithms.

The point of this section is to focus the attention on the ability of EC techniques to exploit statistical properties of a state space. Indeed, exploiting the statistical properties of a problem domain can be described as a sampling strategy that can capture those properties in order to evaluate them and exploit that information to control the search process.

1.6.2 Genetic Algorithms and the Schema Theorem

Genetic Algorithms (GAs) originated in the 1960s and were first developed by Holland [39]. In GAs, the data structure undergoing evolutionary search is called a chromosome and is represented as a string of bits. The crossover operator selects substrings in a chromosome and exchanges them with substrings at the same location on a second chromosome. Multiple variations exist for the implementation of this operator depending on the strategy that is used to select the substrings that are exchanged (e.g., uniform crossover [103]). Figure 1.6 illustrates the one-point and the two-point crossover operators. Mutation operates at the level of bits and randomly flips those bits according to some probability.

Much work has been done to formalize the fundamental principles of operation behind GAs [39, 33, 85, 111, 112]. However, because of the multiple mechanisms involved in the search process embedded in a GA implementation, it becomes difficult to analyze in a formal framework the resulting dynamics of the population of chromosomes. The

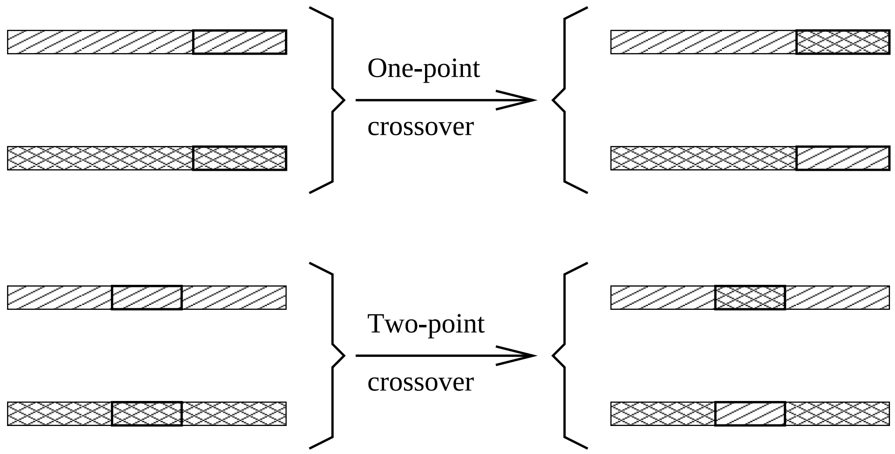


Figure 1.6: Description of the one-point and two-point crossover operators.

first theoretical analysis of GAs has been proposed by Holland [39] and resulted in the so-called *Schema Theorem*. This formulation is known to present some critical drawbacks to describe accurately the behavior of GAs because of some of the assumptions that underlie this theory. However, it is usually accepted that this formulation does capture some of the important principles of operation behind GAs. The purpose of this section is to present the fundamental ideas underlying this formulation.

Central to this theory is the notion of *schemas*. Schemas correspond to a hierarchical partitioning of the space of bit strings of length l into subsets. Each of those subsets is described by a template (or *schema*) which is represented as a string of same length l over the alphabet $\mathcal{A} = \{0, 1, *\}$, where "$*$" is the wild card character which represents a "0" or a "1". For instance, the schema $S =$"$*0***1**$" represents the subset of all binary strings of length eight whose second bit is a "0" and sixth bit is a "1". In the population, each individual's chromosome is an instance of 2^l schemas. The basic idea underlying the schema theorem is that, in parallel to the evolutionary search performed on the space of chromosomes, the GAs implicitly manage a population of schemas. Those schemas correspond to those which have at least one instance in the population of chromosomes. This population of schemas is *not* represented explicitly. However, the idea is to describe the search process underlying GAs by observing the evolution of this population of schemas. Indeed, while GAs evaluate explicitly the fitness of individuals, a fitness is also evaluated implicitly for the schemas that have some instances in the population. More precisely, a schema's fitness corresponds to the average fitness of individuals in the population that are an instance of this schema. This schema's fitness can be seen as an estimate of its "true" fitness defined as the average fitness of all the instances of that schema.

The central result of the Schema Theorem is an expression which describes for a given schema the expected evolution of the number of instances for that schema in the population. Taking the same notations as Mitchell [58], this result is stated as follows:

$$E(m(S,t+1)) \geq \frac{\hat{u}(S,t)}{\overline{f}(t)} m(S,t) \times \mathcal{S}_c(S) \times \mathcal{S}_m(S)$$

where:

- $m(S,t)$ is the number of instances of the schema S in the population at time t,
- $E(.)$ is the expected value,
- $\hat{u}(S,T)$ is the average fitness of individuals in the population at time t that are an instance of schema S,
- $\overline{f}(t)$ is the average fitness of the population at time t,
- $\mathcal{S}_c(S)$ represents the disruptive effect of crossover. Assuming the one-point crossover operator is used, a lower bound for $\mathcal{S}_c(S)$ is:

  $$\mathcal{S}_c(S) \geq 1 - p_c \frac{d(S)}{l-1}$$

  where:

- p_c is the probability that the one-point crossover operator is going to be applied when creating a new offspring,
- $d(S)$ is called the defining length of a schema S and corresponds to the distance between the outermost defined bits in the schema (i.e. those bits that are not the wild card character "$*$").
- l is the length of chromosomes,

- $\mathcal{S}_m(S)$ represents the disruptive effect of mutation. Assuming a bit mutation probability of p_m, a lower bound for $\mathcal{S}_m(S)$ is:

$$\mathcal{S}_m(S) \geq (1 - p_m)^{o(S)}$$

where $o(S)$ is the order of the schema S which is defined as the number of defined bits in S.

This formulation of the Schema Theorem assumes that selection follows a fitness proportionate rule. That is, the expected number of offspring of an individual x is equal to $\frac{f(x)}{\overline{f(t)}}$, where $f(x)$ is the fitness of x and $\overline{f(t)}$ is the average fitness of the population at time t.

The common interpretation of the Schema Theorem is that schemas with short defining length and low order whose average fitness remains above the mean are represented over time by an exponentially increasing number of individuals in the population. While the effect of crossover is represented as disruptive in the Schema Theorem, this operator is believed to play an important role in the search process embedded in GAs. A central idea referred to as the *Building Block Hypothesis* states that one effect of crossover is to recombine instances of short, low-order schemas to create instances of new higher-order schemas of potentially higher quality. Following that interpretation, GAs can be viewed as a statistical inference procedure which is based on the exploitation of statistics

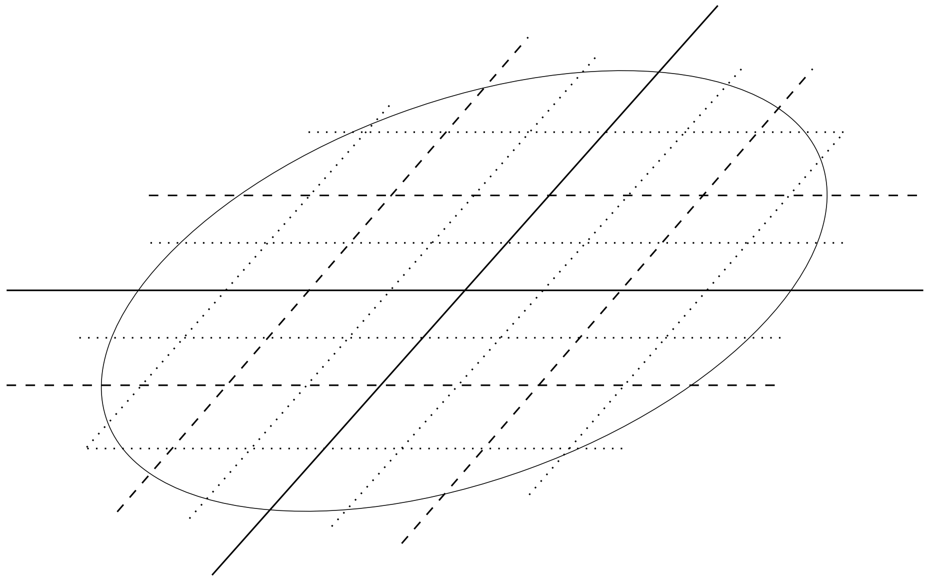


Figure 1.7: Illustration for the partitioning of the state space with respect to schemas.

about the distribution of candidate solutions in the state space with respect to a specific hierarchical structure derived from schemas. More precisely, this process first gathers information about short, low-order schemas and then, as higher-order schemas are considered over time via the selection process and the recombination of building blocks, focuses the search on more specific partitions (or domains) of the state space.

Thus, the knowledge acquisition process implemented in GAs is performed according to a rigid structure. Figure 1.7 illustrates this rigid partitioning by decomposing the space of candidate solutions according to a regular, hierarchical structure. An important consequence is that the representation of a problem in the framework of GAs should take that constraint into account so that the regularities of the problem domain correlate with the schema structure that underlies this knowledge acquisition process.

1.7 Evolutionary Computation as a "Black Art"

As discussed previously, Evolutionary Computation techniques implement a naïve model of natural evolution and can be described as sampling-based strategies that collect data about the state space in order to determine which regions of the space should be explored. However, the introduction of information about structural properties in an EC framework is often seen as a form of "black art". First, it requires a good understanding of which properties of a problem domain can be exploited by the intrinsic strategies embedded in the evolutionary search procedure. Then, the EC practitioner must have a good sense of how the problem should be represented and how the parameters and search operators of the algorithm should be set up so that those properties are indeed exploited. Successful applications are often the result of multiple tests and tuning. The important point is that the structural properties of the problem domain should correlate with the intrinsic search mechanisms of the Evolutionary Algorithm. Failure to exploit those structural properties prevents EAs from scaling up to address more difficult instances of the problem.

1.8 Summary and Objectives of this Dissertation

In this chapter, the importance of identifying and exploiting the structural properties of problem domains has been stressed. As a result, the design of search algorithms for addressing increasingly difficult problems should offer a framework for expressing knowledge about such regularities of a problem domain along with a specific strategy to exploit them. Following that idea, a well-known and effective paradigm to express structural properties associated with a problem domain has been discussed. This paradigm is based on the definition of a structure over the space of candidate solutions. This structure corresponds to a hierarchical decomposition of the state space into subsets which captures some properties which are relevant to the problem domain and that can be exploited with a specific search strategy. The methodology in traditional AI search algorithms consists of defining heuristics that exploit this structure in order to control search. However, the formalization of the properties of a problem domain in a scheme that allows the design of simple decision procedures based on the exploitation of this information is a difficult task for ill-structured problems. For many such problems, the available knowledge is stated more easily in terms of statistical properties. The field of Evolutionary Computation offers multiple techniques that seem more amenable to exploit such properties. However, the statistical inference process embedded in those algorithms is not always understood. Genetic Algorithms are a special case which has been the object of much research to provide a formal analysis of this inference process. The Schema Theorem, one of the theoretical achievements that resulted from that work, offers strong evidence that the knowledge acquisition process embedded in GAs is based on the gathering of statistics about the distribution of candidate solutions in the search space with respect to a specific hierarchical structure derived from the definition of *schemas*. As a result, when using GAs to address a problem, an appropriate representation must be designed so that the structural regularities of the problem correlate with that structure and that relevant information may be captured by the inference process.

EC techniques do provide the designer with a lot of flexibility when being applied to a new problem. However, despite this flexibility, experience shows that it is sometimes difficult to encode some form of knowledge about the structural properties of problem domains in their framework. The reason is that, depending on the implementation choices, the mechanisms embedded in EC techniques are more efficient to capture some specific classes of structural properties than others. Incorporating this knowledge is however important. In particular, it can prevent the search algorithm from spending too much effort exploring poor domains of the search space. The objective of this dissertation is to investigate and propose new techniques to address such situations. More precisely, the central idea is to expand the domain of applications of EC techniques by proposing a more flexible framework to represent and exploit efficiently statistical properties of problem domains. This research work is based on the exploitation of the subset manipulation paradigm in order to construct structures that capture some specific statistical properties of problem domains. Methodologies are introduced for the definition of such structures along with specific search algorithms that exploit the statistical properties they capture.

Chapter 2

Extending the Evolutionary Computation Paradigm

2.1 Problem Definition

2.1.1 Defining Structures to Capture Statistical Properties of Problem Domains

The association of a decomposition of the search space into subsets with some appropriate heuristics is an efficient paradigm to represent and then exploit structural properties of a problem domain. However, designing heuristics to exploit specific structural properties of a problem domain can be a difficult task. Indeed, the definition of heuristics requires a good understanding of the properties of the problem domain and also requires the formalization of this knowledge in a scheme that can be encoded as a simple computational procedure. If the knowledge available about the problem domain corresponds to some statistical properties exhibited by the components of a structure defined over the search space, the existence of such simple computational procedure is very unlikely.

Defining a structure over a search space provides a framework to gather statistics about subsets of candidate solutions. For instance, structures like the one introduced in the first chapter for addressing the n-queens problem can be used to capture statistical properties at different levels of granularity with respect to the hierarchical decomposition of the search space. Based on this source of information, one strategy to explore the search space would consist in evaluating statistical properties of the problem domain at a coarse level in order to identify promising regions. As the set of alternatives under consideration is restricted in response to some predetermined criteria, properties at a finer level of granularity could then be exploited to refine the quality of the current best solution.

In the n-queens example, each subset used in the definition of the structure over the search space is associated with an explicit entity in the representation scheme. Namely, an internal node in the search tree that represents the set of board configurations that can be constructed from a given partial solution. However, exploiting such an explicit representation scheme is not the only approach to define a structure. In particular, in some cases, an indirect approach may be more appropriate to capture the relevant statistical properties. To illustrate this notion, consider a simple problem for which each candidate solution can be evaluated against a number of properties to determine which one it satisfies and which ones it doesn't. Let $\mathcal{S}$ represent the space of candidate solutions and let $\mathcal{P} = \{p_1, \ldots, p_k\}$ be the set of properties against which solutions are evaluated. In practice, each property may correspond to a sub-goal that is achieved (or not) by a solution. Ultimately, the target would be a solution that satisfies all the properties in $\mathcal{P}$. In this example, each property p_i defines a decomposition of $\mathcal{S}$ into two subsets, S_i^1 and S_i^0, that correspond to the candidates that do (or don't) satisfy p_i. This set of properties can then be used to construct a structure over $\mathcal{S}$. For instance, in this structure, each

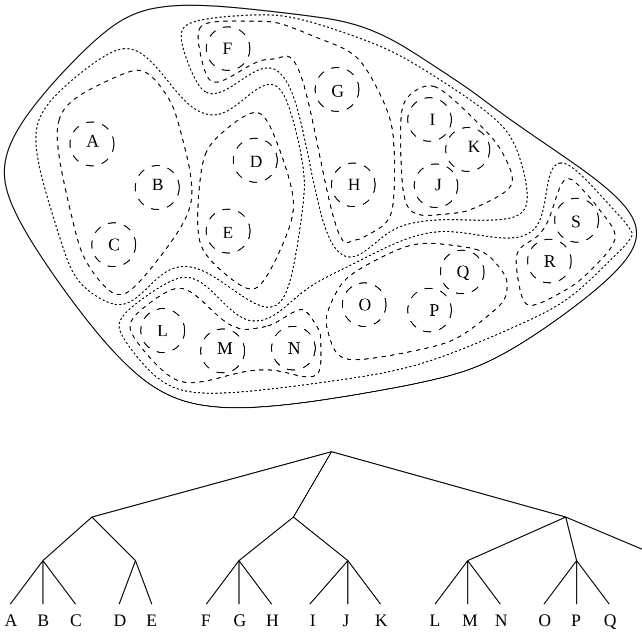


Figure 2.1: Partitioning of the state space with respect to an explicit procedure.

subset would correspond to solutions that satisfy all the properties of a specific subset of $\mathcal{P}$. Then, once such a structure is defined over the search space, it can be exploited to search for domains of the search space that exhibit some specific statistical properties with respect to this structure.

Therefore, different approaches are possible for the definition of structures over a space of candidate solutions. Two methodologies have been introduced to define such structures. The first one, called *explicit partitioning*, encodes explicitly the subsets associated with the structure in the representation scheme. As illustrated in Figure 2.1, the explicit partitioning of the state space corresponds to a mapping between subsets of candidate solutions and the nodes of a tree structure. The fact that an explicit encoding of the decomposition is defined means that, when considering an entity associated with the description of a specific subset, any member of that subset may be accessed. As a result, some specific properties related to the members of that subset may be evaluated

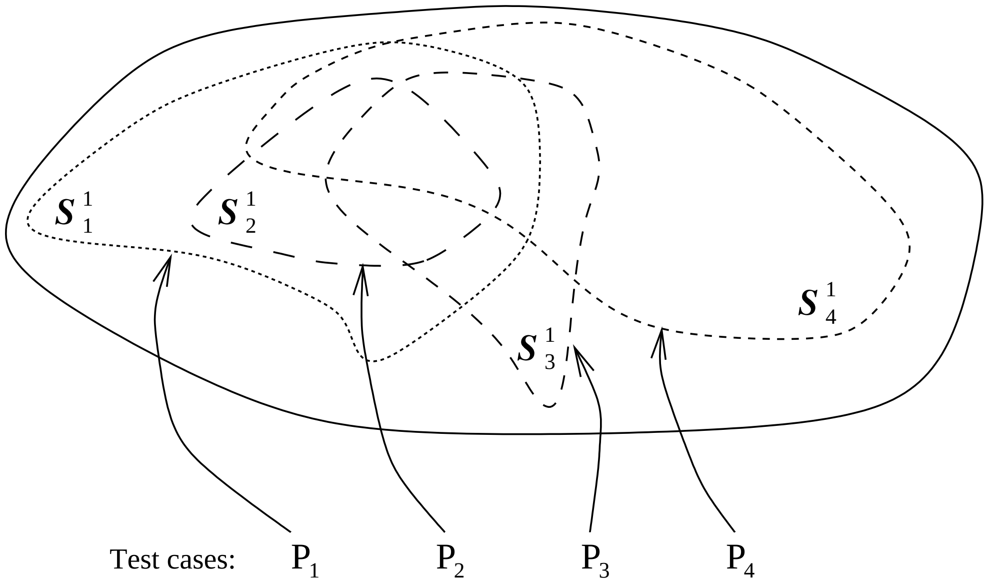


Figure 2.2: Partitioning of the state space with respect to an indirect methodology.

and assigned to that particular entity. In the second methodology, which will be referred to as *indirect partitioning*, the different subsets that define the structure are defined by intention; that is, no explicit object in the encoding scheme is associated with them. This is illustrated in Figure 2.2 where a set of properties is used to define subsets over the search space. Membership of a candidate solution to a particular subset can be evaluated directly however no procedure is available to construct the list of members that satisfy a particular property or a specific subset of properties (other than doing an exhaustive search). New strategies have to be designed in order to exploit statistical information related to the decomposition of the search space according to those subsets.

The choice of one methodology depends on the statistical properties that are exhibited by the problem domain. The goal of the next two sections is to illustrate with examples what kind of statistical properties can be captured with structures constructed following those methodologies. More precisely, two classes of problems are introduced in those sections, each one exhibiting some specific structural properties that can be represented in the framework of one of those two methodologies.

The central motivation underlying this idea of defining a structure over the search space is that stochastic techniques usually require a large sample, and therefore a significant amount of computational effort, in order to accurately infer properties about the problem domain. By defining a structure over the search space which correlates with the statistical properties of the problem domain it is however possible to considerably reduce the size of this sample.

2.1.2 Explicit Partitioning Methodology

Problems whose structural properties are captured by this framework will be referred to as *sequential construction problems*. In order to illustrate this class of problems, consider the problem of designing sorting networks with a minimum number of comparators. This problem is well-known in the EC community and has been addressed by many researchers since the pioneering work of Hillis [38]. A sorting network corresponds to a sorting algorithm for which the sequence of comparisons and swaps is independent of the input. As a result of this property and for a given number n of inputs, such an algorithm permits an implementation in hardware. The diagram in Figure 2.3 is an illustration of a convenient graphical representation for n-input sorting networks. Each horizontal line represents an input of the sorting network and each connection between two lines represents a *comparator* which compares the two elements and exchanges them if the one on the upper line is larger than the one on the lower line. The input of the sorting network is on the left of the representation. Elements at the output are sorted and the largest element migrates to the bottom line. More details about sorting networks can be found in [48].

A direct approach to this problem that would define the space of candidate solutions as the set of all possible sequences of comparators (eventually with an upper bound

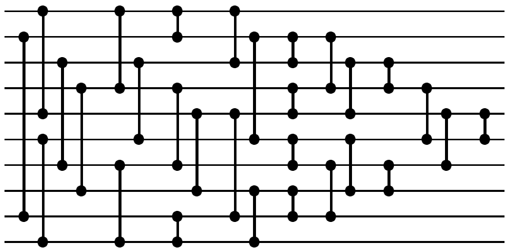


Figure 2.3: A 10-input sorting network using 29 comparators and 9 parallel steps.

for the length of this sequence) would make difficult the design of transformation operators to explore that space. The reason is that any modification of a comparator deeply transforms the function encoded by a network. Here, the difficulty comes from the strong interdependency between the different components of the representation. The complexity of the constraints involved in the description of the space of valid solutions prevent the design of search operators that can at the same time capture those constraints and capture some meaningful information about structural properties of the problem domain that could be used to control the search.

However, a sequential construction approach similar to the one described for the n-queens problem seems more appropriate to address this problem. This sequential approach implements an iterative procedure which starts from scratch and then appends one comparator after the other until a complete sorting network is constructed. This process always terminates with a sorting network because at each stage of the construction it is possible to compute the list of all *significant* comparators (those comparators that will perform a swap for at least one input vector) to extend the current partial solution. This construction process is monotonic in the sense that the number of unsorted vectors at the output of the partial solution don't increase as more comparators are appended. The construction ends with a sorting network when the list of significant comparators is empty.

For this particular problem, a sequential construction approach is more appropriate to capture the natural constraints of the problem. The sequential construction approach results in a search space represented as a search tree or, more generally, as a Directed Acyclic Graph (DAG). Each leaf in this DAG corresponds to a complete solution to the problem while each internal node represents the subset of all valid solutions that can be constructed from the corresponding partial solution.

At that point, the definition of heuristics would be a natural approach to drive the search over that DAG. However, when the properties of a problem are not well-understood or when those properties are ill-structured, the design of simple decision procedures to implement such heuristics is not always possible. In that case, alternative approaches must be considered to exploit the properties of the problem domain. For sequential construction problems, because of the strong dependence of each choice on the choices that have been made in the early stages of the construction process, those early choices usually have an important contribution to the quality of the solutions that are derived from them. To illustrate this concept, consider the game of chess. In that game, pertinent moves in the opening of the game usually provide a player with a strong advantage over his opponent and a good chance of winning the game. In chess and many similar games, heuristics have been identified in order to evaluate with good accuracy what makes a good opening by performing some analysis of the board. The experience resulting from the entire history of chess playing has made possible the representation of the relevant concepts in a formal framework and their encoding into such heuristics. The goal of such heuristics is to provide accurate information about the status of the game without waiting until the actual end of the game.

Following the sorting network example, the class of problems we propose to address concerns problems for which such knowledge is not available in a formal encoding.

In that case, the fundamental idea is to evaluate the quality of early choices by estimating the quality of solutions that are derived from them. Because of the explicit partitioning methodology and the corresponding DAG representation, candidate solutions are organized in a hierarchy where each node at a given level corresponds to a specific sequence of initial steps for the construction of solutions that belong to the sub-tree attached to that node. If the nature of the problem is such that early steps do contribute significantly to the quality of solutions that are derived from them, then good solutions are likely to be grouped together in this hierarchy. Therefore, it would be interesting to exploit this property of the hierarchy as a source of information to control the search. The motivation for applying this strategy to address the sorting network problem comes from the intuition that the choice for the first comparators in the construction does contribute significantly to the expected size of sorting networks constructed from this prefix and that an inappropriate prefix is likely to result in large sorting networks.

2.1.3 Indirect Partitioning Methodology

The domain of application of this methodology corresponds to problems for which candidate solutions are evaluated against a set of test cases or some arbitrary subset of a space of test cases. For the following, such problems will be referred to as *multi-objective problems*. The indirect partitioning methodology is appropriate especially when little information is available about the structural properties of the search space associated with the problem. In that case, the construction of an explicit structure is not likely to improve search significantly since no information is available about what this structure should be. Instead, some other source of information should be considered in order to drive the search. By providing a framework to capture some specific statistical properties, the indirect partitioning methodology proposes an alternative to address such ill-structured problems.

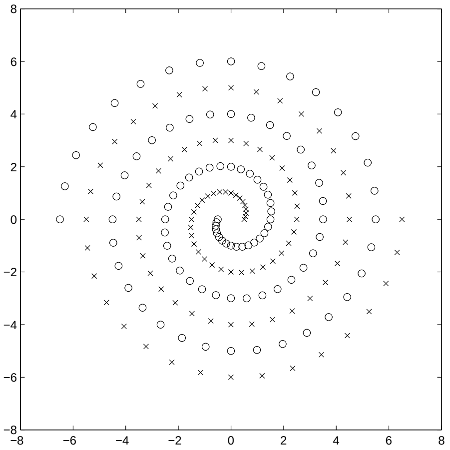


Figure 2.4: Training data for the intertwined spirals classification problem.

To illustrate this methodology, consider the following inductive learning problem which consists of learning to classify points on the plane into two classes according to two intertwined spirals. The data set is composed of two sets of 97 points, on the plane between -7 and +7. These two intertwined spirals are shown as "$\times$" and "$\circ$" in Figure 2.4. This learning problem, originated by Alexis Wieland, has been a challenge for pattern classification algorithms and has been the subject of much work in the AI community, in particular in the neural network field (e.g., [55, 26, 15]). In neural network classification systems based on linear, quasi-linear, radial, or clustering basis functions, the intertwined spirals problem leads to difficulty. When it is solved, the neural net solution often has a very "expansive" description of the spiral, i.e. the conjunction of many small regions, does not generalize outside the training regions, and is thus not particularly satisfying. In the EC community, Koza [50] and Angeline [5] have also investigated this problem using the Genetic Programming paradigm. In this section, the same paradigm and setup as this last work is considered to address the problem. In Genetic Programming, the search space is composed of Lisp S-expressions. A S-expression can be described as a tree in which internal nodes correspond to primitives (or functions) while leaves correspond to terminals (that is, a constant or a variable). A S-expression encodes a function that can be evaluated once the variables have been instantiated. The following function set is considered: $\{+, -, *, \%, iflte, sin, cos\}$, where $iflte$ is a conditional operator that takes four arguments arg_1, arg_2, arg_3 and arg_4. If arg_1 is lesser than or equal to arg_2 then it returns arg_3, otherwise it returns arg_4. The terminal set is composed of: $\{x, y, \Re\}$, where $\Re$ corresponds to the ephemeral random constant, that is a random constant which is instantiated the first time each instance of $\Re$ is evaluated and then keeps the same value. The goal is to discover an S-expression which returns a positive value when given the coordinates (x, y) of a point that belongs to the first spiral and a negative value when given the coordinates of a point in the second spiral.

Here, the search space is composed of all the S-expressions that can be constructed from the set of primitives and terminals. In a sense, each S-expression can be seen as an instance of a computer program. Because any modification of a S-expression usually results in the encoding of a completely different function, it is very difficult to identify some structural properties for this search space. As a result, the search space is ill-structured because of a large number of local optima and a lack of structural regularities. Usually, no *a-priori* explicit partitioning of the search space like the one introduced in the previous section can be constructed in order to drive search. Such a landscape is extremely difficult to search and is typical of needle-in-a-haystack type problems. However, if some domains of this state space exhibit some regularities that can be exploited by the search strategy implemented in Genetic Programming then it would be more efficient to focus the search on such domains because progress would be more likely to occur and a good solution might eventually be discovered. The indirect partitioning methodology proposes a framework to construct a structure over the state space that can be used to capture such statistical properties.

To implement the indirect partitioning methodology, each point p_i in the training

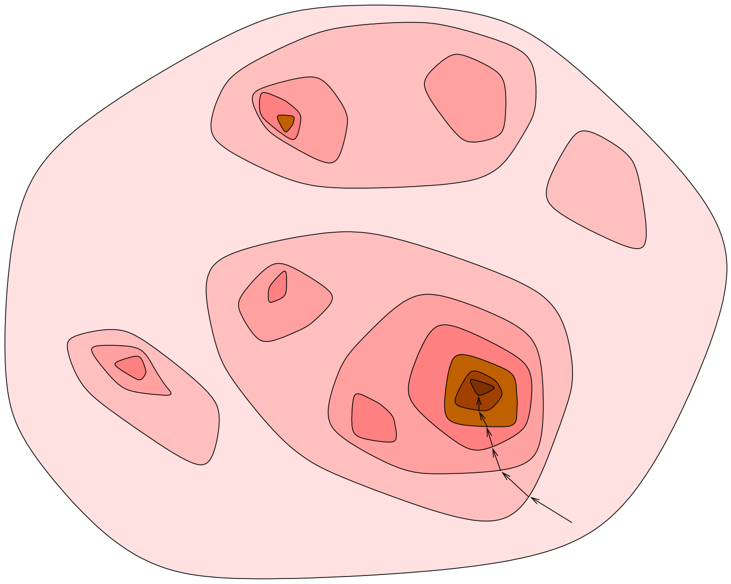


Figure 2.5: Representation of the search space: dark areas correspond to high quality solutions while light areas are associated with poor solutions. The search strategy consists in looking for domains of the state space over which continuous progress can be observed over a period of time.

set is used to decompose the space $\mathcal{S}$ of S-expressions into two subsets, $\mathcal{S}_i^1$ and $\mathcal{S}_i^0$, that correspond respectively to S-expressions which encode a function that classify correctly or fail to classify correctly the test case p_i. The goal is to discover an S-expression that belongs to the intersection of all the subsets $\mathcal{S}_i^1$. Since no procedure is available to construct directly those subsets and no explicit structure is known to capture the regularities of that problem domain, an alternative strategy to address this problem consists in searching for domains of the state space over which a gradient is available for the search algorithm. The underlying idea for that particular strategy is that if the path of continuous improvement along this gradient is long enough then there may be a good chance of discovering a high quality solution. This is illustrated in Figure 2.5 which pictures the number of subsets $\mathcal{S}_i^1$ that intersect for each state. Darker regions represent high quality solutions that cover multiple test cases while light regions correspond to poor solutions. In that picture, the notion of neighborhood between candidate solutions is defined with respect to the search operators implemented in the search algorithm (i.e., exchange of sub-trees between S-expressions in the case of the GP paradigm). The goal is to identify domains of that search space over which successive transformations of a solution by the search algorithm would result in continuous progress toward higher quality solutions.

As illustrated in the previous example, the motivation underlying the indirect partitioning methodology is to search for domains of the state space that correlate with the intrinsic mechanisms implemented in an arbitrary search algorithm. Such domains exhibit the property that progress can be observed consistently over a period of time. That is, when search is focussed in those domains, solutions of increasing quality are more likely to be discovered by the search algorithm. Therefore, if the problem under consideration can be stated as a multi-objective problem, the indirect partitioning methodology may be used as a very efficient tool to capture some specific statistical properties of the state space with respect to an arbitrary search procedure. Exploiting this source of information becomes particularly important when little knowledge is available about the structural properties of the problem domain. If some domains of the state space over which continuous progress may be observed does exist then, ultimately, a solution may be discovered that captures some underlying structure of the problem domain and generalizes well outside the training data. In the case of the intertwined spirals, such a solution could be like the one plotted in Figure 2.6 whose corresponding S-expression is presented in Figure 2.7.

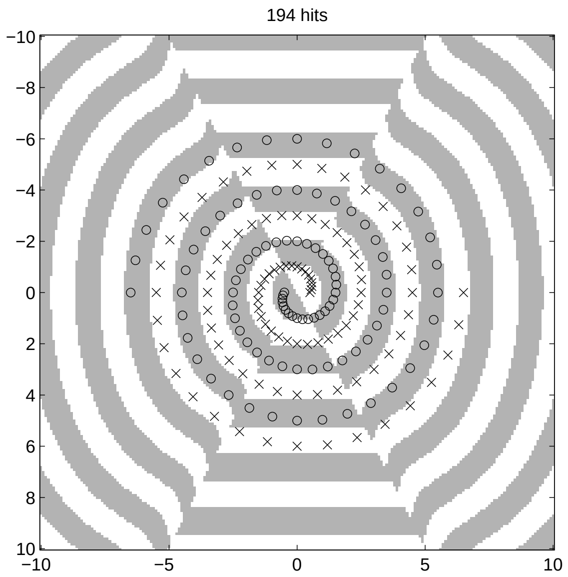


Figure 2.6: Perfect score generalizing classification of the two intertwined spirals.

```
(sin (% (iflte (- (- (- (* _A _A) (sin (% (iflte  -0.52381
                                                  _B
                                                  (sin -0.33333)
                                                  -0.33333)
                                          -0.33333)))
                     (* _B _B))
                  (% _A (% -0.33333 _A)))
               -0.80952
               _B
               (sin (% (% _A
                          (- (cos (sin (* (cos (sin -0.52381))
                                          (% _B (% _A (- (cos -0.33333)
                                                         0.04762))))))
                             0.04762))
                       (sin (sin -0.33333)))))
        -0.33333))
```

Figure 2.7: A 52-atom S-expression classifying correctly all 194 training examples for the intertwined spiral problem.

2.2 Proposed Solution

Two fundamental techniques are developed in this dissertation. The first one, named Self-Adaptive Greedy Estimate (SAGE), implements a search strategy that exploits some specific statistical properties that are captured by the explicit partitioning methodology. As discussed in section 2.1.2, the central idea for this strategy is to identify early choices in sequential construction problems that are more likely to result in high quality solutions. SAGE is an iterative search algorithm which proceeds level by level in the hierarchical partitioning of the space of candidate solutions. At each level, SAGE exploits a sampling-based strategy to evaluate some specific statistical properties about the corresponding subsets of this decomposition. Then, when enough information has been gathered, search proceeds to the next level, focusing its attention on the most promising subsets. An important feature of SAGE is that the statistical inference process it implements is supported by an evolutionary-inspired model. As a result SAGE exhibits many of the positive features attributed to Evolutionary Algorithms like robustness, flexibility and the ability to explore multiple alternatives in parallel. Moreover, such population-based models usually admit an efficient implementation on massively distributed architecture, making them an attractive tool to address large scale problems.

The second technique introduced in this dissertation is based on the concept of *coevolutionary learning* and is proposed as a solution to exploit statistical properties of problem domains that are captured with the indirect partitioning methodology. In coevolutionary learning, agents are evaluated in an environment that responds to their progress. The fundamental idea is that, instead of providing the learning agents with a fixed training environment, those agents coevolve with an adaptive environment that always challenges them in such a way that the performance of the learning agents improves continuously. From the perspective of this dissertation, this ability to observe continuous progress is seen as the search for domains of the state space that exhibit some specific properties. More precisely, our approach to coevolutionary learning is based on the idea of augmenting an evolutionary algorithm with a meta-level strategy that controls the evolution of agents toward domains of the state space that exhibit structural properties that correlate better with the search mechanisms embedded in that algorithm. The consequence of this meta-level strategy is that the evolutionary algorithm is less likely to converge to a local optimum and continuous progress has a better chance to occur.

2.3 Related Work

The classes of problems that are addressed by the techniques introduced in this dissertation have the common property that either the available information about their structural regularities cannot be encoded appropriately in the framework of Evolutionary Algorithms in such a way that those regularities are indeed exploited, or that very little information is available about such structural regularities. Those two issues are very common to the EC practitioner. In particular, a recurrent issue when applying EC techniques to a new problem concerns the design of an appropriate encoding to represent candidate solutions. Indeed, committing to an *a priori* representation is a difficult decision when it is based on little information about the properties of the problem. To go around that problem, different schemes have been proposed in order to adapt the encoding or the search operators during the evolutionary process. The following sections describe two different approaches that explore that direction.

2.3.1 Messy Genetic Algorithms

The central idea of "messy GAs" [34] is to perform evolutionary search on variable length chromosomes. To support this mechanism, a chromosome is in fact represented by a list of pairs *(index, value)*. Then, a binary string is constructed from that list by assigning the value specified in each pair to the corresponding index. As a result, the value of some positions in the binary string may be unspecified because their index is not represented in any pair in the chromosome or some may be specified more than once. Specific mechanisms have to be implemented to deal with those cases and the evaluation of chromosomes' fitness [34].

The motivation underlying the development of "messy GAs" is that short chromosomes may capture some important features relevant to the nature of the problem. Those features would involve, for example, a few important bits in the representation associated with some high-level properties. Therefore, the search process should continue by building up new chromosomes from those individuals, adding more information to their description (that is, specifying more bits). Progressively, such chromosomes would capture some finer and finer regularities of the problem domain and would end up as a complete encoding of all the bits.

Some important drawbacks are however associated with this strategy (e.g., see [58]). In particular, the design of a general purpose procedure for the evaluation of the fitness of incomplete chromosomes resulting in a consistent behavior across a large set of applications is a difficult technical challenge.

2.3.2 Adaptive Evolutionary Computation

Adaptive evolutionary computation denotes a class of evolutionary algorithms that modify the values for some of their operational parameters while performing search. The motivation underlying this strategy is that evolutionary search may improve its performance by automatically tuning its operational parameters to exploit more efficiently the regularities of the problem domain. This is usually achieved by incorporating an encoding of those parameters in the entities undergoing evolution.

Such operational parameters may be associated with different levels of control in the evolutionary process. Angeline [5] proposed a classification in three categories: population level, individual level and component level. Accordingly, a large variety of techniques have been proposed to implement this strategy. The work of Rosenberg [90], or Schaffer and Morishima [95] on evolving crossover positions in GAs is one example. This idea has been extended to the field of Genetic Programming to implement strategies that automatically adapt the probability of crossover points when exchanging sub-trees between S-expressions [41, 5]. Mechanisms adapting dynamically the variance of the mutational noise applied to real-valued vector representations have also been proposed very early in the field of Evolutionary Strategies [98, 10].

At the highest level, techniques that automatically adapt the representation of solutions for improving the performance of evolutionary search have been proposed. Paredis's work on symbiotic coevolution illustrates this strategy with promising results [72, 73].

Successful applications of adaptive evolutionary computation are usually the result of an appropriate trade-off between a larger search space and a more adapted strategy for exploring that search space. Indeed, the increase in flexibility procured by such adaptive techniques gives more opportunities to the designer for introducing informed biases.

2.3.3 Concluding Remarks

While methods to adapt the encoding or the search operators of an EA by incorporating those features as "parameters" of the evolutionary process seem very attractive at first, they are however still subject to one fundamental limit of search algorithms: the "No-free Lunch" theorem. Indeed, as discussed in [20], such attempts to improve performance of an evolutionary algorithm are generally fruitless unless the search mechanisms that control the evolution of those parameters correlate with some properties of the problem domain under consideration. Put in another way, this result means that the design of any method that implements some form of adaptation for controlling search should be driven by the identification of regularities of the problem domain. Failure to consider those properties in order to introduced "informed" biases prevents such adaptive methods from achieving any significant improvement.

As a corollary to this observation, when proposing new techniques for search, one should always attempt to describe what specific properties are captured and exploited by the underlying strategies. Indeed, identifying those properties makes possible a more accurate description of the appropriate domain of applications for those methods.

2.4 Original Contributions

The contribution of this dissertation concerns the investigation of methods to exploit statistical properties of problem domains. The central idea is to augment the search space with a structure that captures some specific statistical properties. Then, the idea is to use methods exploiting the same principles of operation as those embedded in Evolutionary Algorithms as a paradigm to perform statistical inferences based on the exploitation of this structure. The benefit of this approach is illustrated with the design of new search procedures that address two specific classes of problems. The first class is referred to as sequential construction problems and is addressed by SAGE, an algorithm that implements a sampling-based strategy to exploit statistical properties for the distribution of solutions in tree structures. The second class concerns a category of ill-structured multi-objective problems. Those problems are addressed with a specific coevolutionary strategy based on the implementation of an evolutionary race between two populations. Coevolution offers an attractive paradigm for search and learning when little knowledge is available about the properties of a problem domain or when this knowledge is difficult to introduce in the framework of a specific search strategy. However, coevolution also comes with several impediments. For this reason, only a limited number of successful applications based on coevolution can be found in the research literature. One important contribution of the research work presented in this dissertation is to provide more insights and a better understanding of the fundamental issues involved in this promising field. In particular, a central result introduced in this dissertation addresses directly the significance of coevolution as a paradigm for achieving open-ended evolution. That is, the emergence of agents that continuously improve their performance. More specifically, two fundamental conditions have been identified for achieving continuous progress: the need to maintain useful feedback from the training environment and the need for a meta-level strategy to ensure progress in the long term. Finally, based on the development of new techniques implementing those two requirements, the significance of the coevolutionary paradigm for addressing the issues of adaptability and generalization is demonstrated.

In addition, the search algorithms described in this dissertation have made possible the discovery of new results for large instances of difficult combinatorial optimization and inductive learning problems. The following is a list of some of those achievements:

- new constructions for 13-input sorting networks using fewer comparator-swaps.
- a new cellular automata rule that implements the majority classification task with significantly higher accuracy.
- a procedure for the induction of minimum DFAs from positive and negative examples requiring sparser training data than the previous best algorithms for that task. This procedure ended up as a co-winner in the "Abbadingo" challenge, a competition in grammar induction.

2.5 Outline of Chapters

The body of this dissertation is organized as follows. Chapter 3 describes SAGE, a search procedure for the exploration of trees and directed acyclic graphs (DAGs) based on a sampling-based strategy. The central idea in SAGE is to exploit the information about the distribution of solutions gathered by sampling the tree or the DAG to determine on which domain(s) of the state space to focus the search. The application of SAGE to three different problems is then presented: the construction of compact sorting networks, the induction of finite state automata from a training set composed of positive and negative examples, and a game of Solitaire.

Chapters 4, 5 and 6 investigate coevolutionary methods and their application to search and learning. First, Chapter 4 describes the concept of coevolution and the motivations for exploiting this paradigm in the framework of the indirect partitioning methodology. Chapter 5 states two fundamental requirements for achieving continuous progress in the context of coevolutionary learning and introduces the "Ideal" trainer as a paradigm implementing those requirements. Finally, Chapter 6 describes two applications of the "Ideal" trainer paradigm. The first one consists in the discovery of cellular automata rules that implement the majority classification task. The second application presents a system implementing a modular approach to inductive learning based on the "Ideal" trainer paradigm.

Chapter 7 summarizes the different issues and achievements that resulted from this work. Then, a presentation of the contributions of this research and its significance to the EC community concludes this dissertation.

Chapter 3

SAGE: a Sampling-Based Strategy for Search in Trees and Directed Acyclic Graphs

As discussed in chapters 1 and 2, capturing the intrinsic regularities of a problem domain is important in order to address difficult instances of that problem. The main reason is that such instances usually involve a large search space. In chapter 2, a specific class of problems referred to as *sequential construction problems* has been introduced. Such problems are combinatorial optimization problems that are highly constrained and sequential in nature. As a result, those problems are difficult to describe in the framework of existing EC techniques. The reason is that the information gathering process in those techniques is supported by a structure that may not be compatible with an appropriate encoding of the problem. That is, an encoding that would allow the intrinsic search mechanisms embedded in the Evolutionary Algorithm to exploit the statistical properties of the problem.

By definition, sequential construction problems are associated with an iterative procedure which defines a hierarchical partitioning of the space of candidate solutions into a tree or a Directed Acyclic Graph (DAG) structure. The internal nodes of the tree correspond to partial solutions that are associated with some intermediate stage in the construction process. Each internal node (and its associated partial solution) can then be seen as the encoding of a specific subset composed of all the candidate solutions that can be constructed by extending this partial solution. This principle was described as the *explicit partitioning methodology* in chapter 2.

Addressing sequential construction problems in the context of Evolutionary Algorithms is difficult because it requires the design of a representation and some search operators that capture the constraints that apply for the construction of valid solutions and at the same time exhibit some regularities that can be exploited by the search strategy. If the representation and the operators can't capture the construction constraints or if the state space doesn't exhibit some regularities, search becomes trapped in local optima. In the EC community, such undesirable property of a representation is sometimes referred to as *epistasis*, meaning that the degree of interdependence among the components of the representation is high and makes the problem deceptive with respect to the search operators. In fact, an important part of the work involved when applying an EC technique to a particular problem is to design a representation for states that is compatible with the search operators embedded in the evolutionary algorithm.

On the other hand, sequential construction problems would seem a typical application for traditional artificial intelligence backtracking search algorithms. However, to face the problem of combinatorial explosion, those search algorithms usually require the design of heuristics in order to control the search. Such heuristics use problem-specific knowledge that has been identified for the problem at hand. This knowledge can be an estimate of the cost to reach one valid solution from a partial solution (e.g. Manhattan distance for the "8-puzzle" problem [84]) or an ordering of the children of the current node in order to define a priority for the branches to explore and/or how to prune the search tree (e.g. a chess program). However, such heuristics cannot use any information about the distribution of the solutions in the space unless this information has been provided explicitly. For example, following an idea first introduced by Palay [68], Baum [11] used an evaluation function for game tree search which returns a probability distribution rather than a single number to estimate the "value" of a position. Then, these distributions are propagated upward to the root of the tree and the best move corresponds to the branch whose distribution has highest mean. The distribution returned by this evaluation function is a model of the distribution of the values of children nodes for the current leaf and can be the result of a combination of hand designed features and of a training with statistics data from experiments. However, little work has been done to explore the idea of using the distribution of solutions when an evaluation function such as the one described above is not know or difficult to design.

This chapter presents an algorithm that captures the same principles of operation as Evolutionary Algorithms in order to gather information about the distribution of candidate solutions in the search space. This knowledge acquisition process is supported by a specific hierarchical partitioning of the search space defined by the construction procedure associated with the problem. The motivation for the development of this algorithm is to combine some of the fundamental search strategies embedded in Evolutionary Algorithms with a framework that can capture efficiently the constraints associated with the definition of valid solutions. More precisely, the central idea for that algorithm is to collect information about the distribution of values assigned to the leaves of the sub-trees associated with the alternatives under consideration by performing a random sampling. Then, this information is used to focus the search on the most promising alternatives.

This result is achieved by using a model of local competition between the elements of a distributed model, which allow the algorithm to allocate more "resources" (i.e., processing elements) to the most promising alternatives in a self-adaptive manner. Such a model provides this algorithm with the advantage of allowing a highly distributed implementation because of a loose central control. Because of this architecture, we named this algorithm *Self-Adaptive Greedy Estimate* (SAGE) search procedure. So far, random sampling techniques on search trees have been used to predict the complexity of search algorithms [49, 16], but never as a strategy to control the search.

This chapter describes the application of SAGE to three difficult problems. The first one is a *grammar induction* problem and originated from the Abbadingo DFA learning competition [54] which took place between March and November 1997. This competition proposed a set of difficult instances for the problem of DFA learning as a challenge to the machine learning community and to encourage the development of new algorithms for grammar induction. The problem instances proposed in that competition were expected to be just beyond the current state of the art. Each instance consists in a set of training examples and a set of test examples extracted from a randomly generated DFA. The goal is to discover a model of the training set which has a predictive error over the test set less than one percent. The labeling of the test examples and the underlying DFAs of the different problem instances were disclosed only at the end of the competition. Therefore, the performance of a model could be tested only by submitting a labeling candidate for the test examples to an "Oracle" implemented on a server at the University of New Mexico. The complexity of the different instances evolves in two dimensions: the size of the underlying DFA and the sparsity of the training data. This feature introduces a partial order over problems which makes possible multiple winners. The SAGE search algorithm was able to solve difficult problems along the dimension of sparser data and was one of the two winners of this competition. The second problem is a discrete optimization problem: the discovery of *sorting networks* with a minimal number of comparators. Several computer scientists and mathematicians worked on that problem in the '60's, trying to construct by hand minimal solutions for sorting networks up to 16 inputs. An extensive presentation of that problem is presented in [48]. Since that time, no improvement has been made for that problem. Using SAGE, new constructions have been discovered whose size matches all the upper bounds already known and even improves by one comparator the best upper bound for the 13-input case. The third problem is a one-player game named *Solitaire*. This game can be stated as a combinatorial optimization problem for which the player wants to maximize the number of moves. Little knowledge is available for the design of good strategies to play that game. However, SAGE exhibits a performance similar to a highly-skilled human player.

3.1 The Self-Adaptive Greedy Estimate (SAGE) Search Algorithm

3.1.1 Principle

As discussed in the previous section, the search space associated with sequential construction problems is represented as a tree or a directed acyclic graph. An approach that has proven to be effective to illustrate the search strategy implemented in SAGE is to compare this algorithm with a well-known AI algorithm for search in DAGs and trees called *Beam search* [113]. Beam search examines in parallel a number of nearly optimal alternatives (the *beam*). This search algorithm progresses level by level in the tree of states and it moves downward only from the best w nodes at each level. Consequently, the number of nodes explored remains manageable: if the branching factor in the tree is at most b then there will be at most wb nodes under consideration at any depth. This parallel search strategy is illustrated in Figure 3.1: at each level, the alternatives under consideration are evaluated with respect to some objective function (denoted h_i), then the search moves on to the next level, considering only the children of the best $w = 3$ nodes. SAGE is similar to this algorithm in the sense that it proceeds level by level

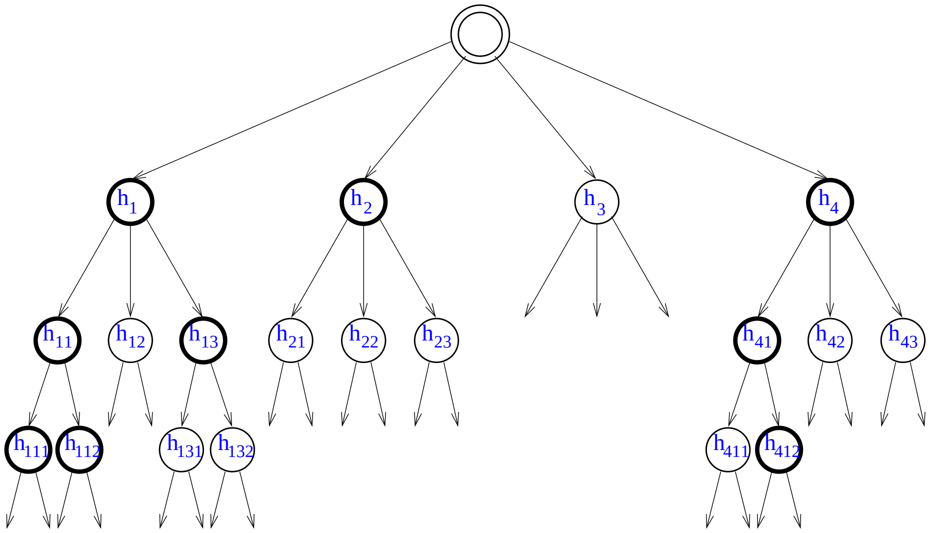
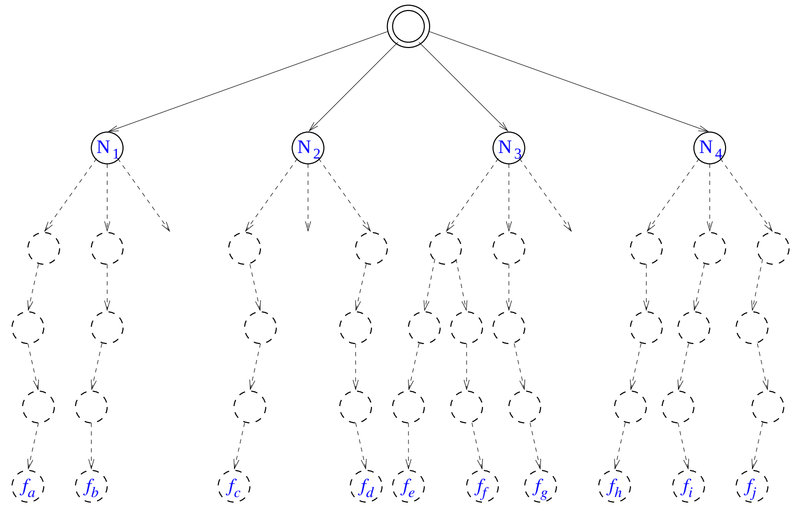


Figure 3.1: Illustration for the parallel search strategy implemented in Beam search.

in the search tree and it implements a distributed-model that performs multi-threaded search. SAGE is composed of a population of *processor elements*. Each of them is playing the role of an elementary search algorithm and is seen as one alternative in the beam. In tree search algorithms, a *fitness* (or score) is assigned to internal nodes in order to determine which nodes will be explored further and which nodes will be ignored. Beam search uses heuristics to score the different alternatives and to select alternatives that are most promising, i.e. the nodes with largest scores. However, this approach assumes the existence of an evaluation function used to score the nodes. The new idea proposed by SAGE is to estimate the score of internal nodes by performing a random sampling from this node. That is, a path is constructed incrementally and randomly until a leaf (or valid solution) is reached. Then, the score of this solution is directly computed with respect to the problem objective function and it is assigned to the initial node. This random-sampling strategy is illustrated in Figure 3.2 -*(a)* where $\mathrm{N_i}$ represents the number of processor elements associated with a specific internal node (i.e., alternative) and f_i represents the evaluation of a leaf (i.e., solution).

(a)



(b)

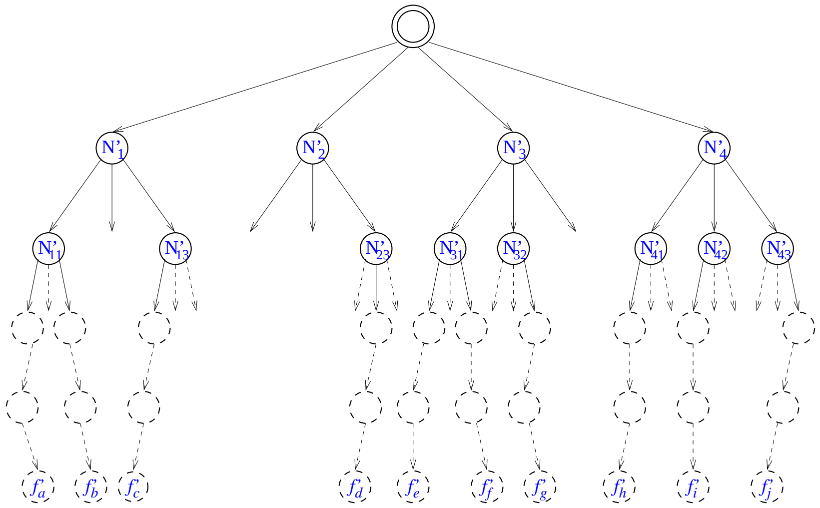


Figure 3.2: Illustration for the sampling-based strategy exploited by SAGE to evaluate nodes and the adaptive strategy which focuses search on most promising alternatives.

More precisely, SAGE is an iterative search procedure for which each iteration is composed of two phases, a *construction* phase and a *competition* phase. SAGE implements a population of elementary randomized search algorithms and a *meta-level strategy* which controls the search procedure by distributing the alternatives under consideration among this population of processor elements. For any iteration, all the alternatives represented in the population have the same depth in the search tree and they are represented by a number of processor elements which depends on their relative performance. At the beginning of the search, this depth is null and it increases with the number of iterations according to a meta-level strategy which is described in section 3.1.4. During each iteration, the construction and the competition phases perform the following operations:

- *construction phase*: during this phase, each processor element calls the construction procedure designed for the current problem. This procedure starts the construction from the internal node which represents the alternative assigned to the calling processor and thereafter makes each decision by randomly selecting one choice from the list of choices available at each step. Each random selection is performed with respect to a uniform probability distribution. This phase ends when all the processor elements have constructed their own solution.

- *competition phase*: the purpose of this phase is to focus the search on most promising alternatives by assigning more representatives to them. This result is achieved by assigning those better alternatives to the processor elements that are representative of poor alternatives. The details of that phase are described in section 3.1.3. In Figure 3.2 *-(a)*, this process updates the value of N_i's so that most promising alternatives are represented by a larger number of processor elements.

In summary, SAGE is a population-based model in which each processor element is the representative of one alternative for the current level of search in the tree. That alternative determines the initial node from which the random sampling is performed by that processor element during the construction phase. Then, SAGE controls the exploration of the search space according to the following strategy:

1. Initially, the search is restricted to the first level of the tree and each processor element in the population randomly selects one of the first-level nodes.

2. Each processor element scores its associated node (or alternative) by performing a random sampling. This is the construction phase.

3. Then, the competition phase is operated. The purpose of this phase is to focus the search on most promising alternatives.

4. A test is performed by the meta-level strategy and the result determines whether the level of search is increased by one or not. In the affirmative, each processor element selects uniformly randomly one of the children of its associated node and this node becomes the new alternative assigned to the processor element. In Figure 3.2 -*(b)*, this translates with the fact that $\sum_j \mathrm{N'}_{ij} = \mathrm{N}_i$.

5. The search stops if no new node can be explored (because the search reached the leaves of the tree); otherwise it continues with step 2.

In the SAGE model, the level of search in the tree is called the *commitment degree* since it corresponds to a commitment to the first choices of the incremental construction of the current best solution.

The following sections describe in detail the different implementation issues for the construction phase, the competition phase and the management of the commitment degree.

3.1.2 Construction Phase

The construction procedure used during this phase plays an important role since it determines the distribution of solutions when sampling the search tree. The procedure that generates the list of possible extensions at each step of the incremental construction of a solution to a problem can be implemented in several ways. It is possible to introduce some problem-specific heuristics to drive the search in a particular direction, to introduce some constraints in order to reduce the size of the search tree, or to use some strategies that modify the distribution of solutions. The *multiple-sampling* strategy is an example of a technique that adapts this distribution. This strategy simply evaluates the score of an alternative by taking the best out of n samples, where n is a parameter of SAGE. In the limit (as $n \longrightarrow \infty$), one of the samples will correspond to the best score for the alternative under consideration and thus this technique would give an exact measure of the score of partial solutions. In practice, n cannot be too large since the computational resource required increases linearly with its value. However, increasing n can be an interesting alternative to a larger population size since the amount of memory required to store the members of the population can quickly become overwhelming for complex problems.

3.1.3 Competition Phase

The SAGE search algorithm uses a population of geographically distributed processor elements. This architecture makes it amenable to a simple implementation on several models of computer networks or of parallel computers since different topologies can be implemented for the adjacency structure over the population of processor elements. For the sake of simplicity, we will consider a model in which the population of processor elements is mapped on a wrap-around mesh (i.e. a torus). A processor element is assigned to each node of the mesh and, therefore, has four adjacent nodes. In such a setup, the competition phase can be performed in the following way:

- Each processor element compares its score with the score of processor elements in its neighborhood. This neighborhood is composed of the nodes in the mesh whose Manhattan distance from the central node is less than a given value of the parameter *radius*.

- If one processor element in this neighborhood has a better score than all the others then the central processor element becomes a representative of the same alternative as that node. If several processor elements in the neighborhood have identical scores which are better than all the others then the central processor element becomes a representative of the alternative associated to one of them, uniformly randomly selected. If no processor element in the neighborhood is better than the current processor element, then it remains a representative of the same alternative.

So, if a given alternative for the current level of search in the tree is more likely to lead to a high score solution then more copies of that alternative will be represented by the population of processor elements as a result of this competition phase (thus focusing the beam). This process is illustrated in Figure 3.3 which presents the state of the population at different stages of the search. The problem consists in sorting integers $\{0 \dots 9\}$ in ascending order. The search tree represents all the permutations of those numbers. Each node at a given level in this search tree corresponds to the addition of one element to the prefix associated with its parent. In the simulation of the population evolution presented in Figure 3.3, each processor is associated with a node in the first level in the search tree (which corresponds to a single number). During the construction phase, each processor element constructs a random permutation of the numbers $\{0 \dots 9\}$ by extending the current prefix (in that example, the prefix corresponds to the first element of the sequence). The fitness is defined as the number of ascents in the resulting sequence. As a result of the local competition, nodes associated to 0 are represented by a growing number of processor elements, thereby focusing the search on that particular integer as the first element for the final sequence.

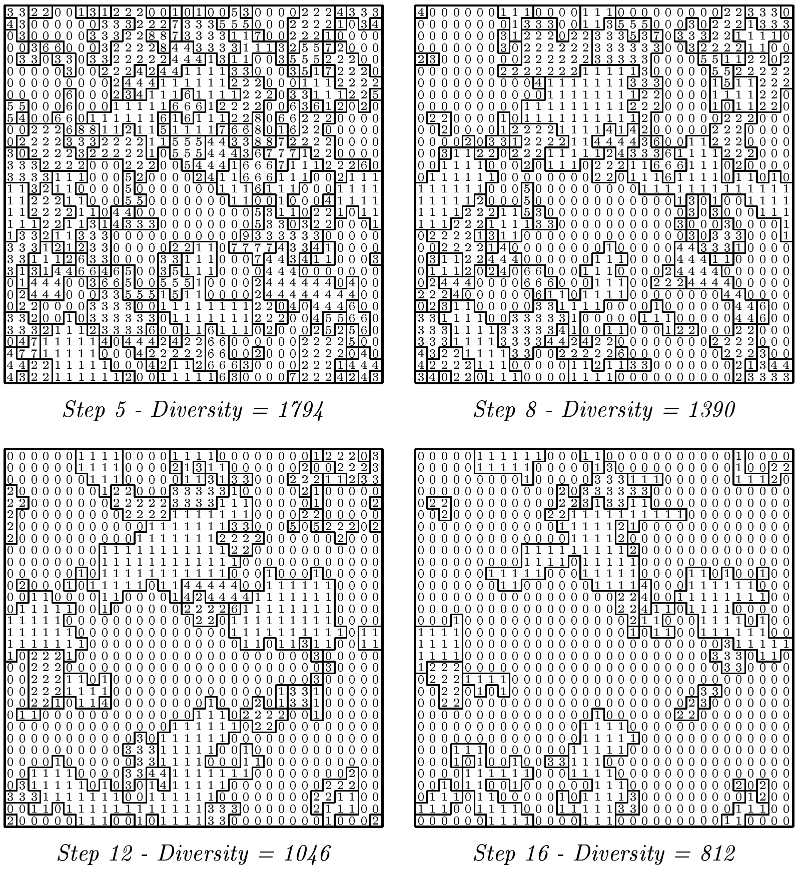


Figure 3.3: As a result of the local competition among members of the population, search focuses on most promising alternatives.

Such a geographically distributed model with local interactions has already been used in the past to implement evolutionary search [38]. This model usually results in multiple niches that are maintained in the population because local interactions induce a delay for the diffusion of better niches. From a search strategy point of view, this feature can be seen as the simultaneous exploration of different sub-domains of the state space.

3.1.4 Management of the Commitment Degree

Successive competition phases result in most promising alternatives being represented by larger and larger numbers of processor elements. Ultimately, if one waits until all the population agrees on the same alternative before continuing the search on the next level, the beam would be reduced to the exploration of a single alternative. On the contrary, if the level of search is increased too frequently, the beam would become very wide. In that case, many different alternatives are represented in the population and each alternative is represented by a small number of processor elements. So, the likelihood of a promising alternative disappearing during a competition phase because of non-favorable random sampling would be non-negligible.

Clearly, for the search to be efficient, these two extremes must be avoided and one wants to find a balance between the number of alternatives represented in the population and a number of representative for each alternative large enough to allow a reliable evaluation of their score in the random sampling operation. This goal is achieved by using a strategy that controls the depth of the alternatives represented in the population and, therefore, the level of search. The purpose of this meta-level strategy is to decide when the commitment degree should be incremented. As described previously, this results in each processor element moving downward by one level in the search tree by selecting randomly one of the child nodes of the current alternative it represents. In our work, we have been using two different strategies to control the commitment degree. The first one simply increases the commitment degree every n iterations, where n is a parameter set at the begin of the run. The drawback of this strategy is that n has to be chosen astutely in order to maintain the right balance between exploration and exploitation. Thus, this strategy is brittle since the value of n has to be chosen *a priori* and depends highly on the problem under consideration and the actual value of the commitment degree.

To correct those drawbacks, a second strategy has been designed. This strategy performs a measure of the state of the population called the *diversity measure* which is an estimate of the width of the beam. To compute this measure, each processor element in the population counts the number of nodes in its neighborhood that correspond to a different alternative than itself. Then, this number is summed over all the members of the population and the result is the diversity measure. This measure reflects the degree of convergence of the population. If only a few alternatives are represented in the population then this measure is small because most of the processor elements represent the same alternative as their neighbors. On the other hand, if many alternatives are explored, and each alternative is represented by a few processor elements, then the diversity measure is large. As the population focuses on most promising alternatives because of the competition phase, the diversity measure decreases (Figure 3.3 illustrates this dynamics for the evolution of diversity). When this measure reaches an arbitrarily fixed threshold (which is a parameter of the model), the meta-level strategy considers that the width of the beam is small enough and the search can continue on the next level of the tree. The drawback of this strategy is that it can take a long time for the diversity measure to reach the given threshold if several alternatives lead to equivalent solutions. This problem doesn't appear with the first strategy. A hybrid approach combining those two strategies offers a good compromise, the first strategy preventing deadlocks by fixing a limit on the number of iterations for any value of the commitment degree.

3.1.5 Parameters of SAGE

This section is a summary of the different parameters of the SAGE search algorithm that have been defined in the previous sections.

Population size : If the number of processor elements in the population increases then the width of the beam can also be larger without decreasing the efficiency of the search. Therefore, the size of the explored state space is directly related to the size of the population.

Number of iterations for the multiple-sampling strategy : This number modifies the distribution of solutions returned during the construction phase by selecting the best out of n successive calls to the construction procedure.

Neighborhood used in the competition phase : The size of the neighborhood used in the competition phase controls the dynamics of evolution of the population. Indeed, if the radius of this neighborhood is large then best alternatives will propagate faster in the population and the search will focus quickly on the alternatives with a high score, discarding apparently less promising alternatives. On the contrary, a small radius allows the search to converge slowly. In the current implementation, the distance between members of the population is defined as the Manhattan distance. The effect of the value of this parameter on the performance of the search depends a lot on the reliability of the score returned by the construction phase.

Management of the commitment degree : As discussed in the previous section, this parameter plays an important role in controlling the balance between exploration and exploitation.

3.2 Application 1: The Abbadingo DFA Learning Competition

3.2.1 Presentation

The aim of inductive inference is to discover an abstract model which captures the underlying rules of a system from the observation of its behavior and thus to become able to give some prediction for the future behavior of that system. This ability is fundamental for learning and underlies the process of scientific discovery or the process by which an animal can interpret its environment and exhibit planning activity in order to achieve a goal. In the field of grammar induction, observations are strings that are labeled "accepted" or "rejected" and the goal is to determine the language that generates those strings. For example, given the binary strings in Table 3.1, one could conjecture that they are instances of the language composed of strings of even length while strings of odd length are not recognized by the language. Angluin and Smith [7] have presented an excellent survey of the field, covering in particular the issue of computational complexity and describing some inference methods for inductive learning. Several representations have been proposed to describe the abstract models for grammar induction like deterministic finite state automata, boolean formula or propositional logic. More recently, Pollack [79] proposed dynamical recognizers as an interesting alternative to those symbolic approaches, leading to a wide range of Recurrent Neural Network (RNN) architectures [109, 115, 23, 29] that had been employed for similar tasks. However, none of them could compete in the Abbadingo competition because of the proposed problems size.

The application of the SAGE search algorithm to this problem has been motivated by the Abbadingo competition organized by Lang and Pearlmutter [54]. It is a

Accept	Reject
01	0
10	1
0010	010
1001	111
010110	01001
00001111	10110
10111010	0111001

Table 3.1: A tiny training set of labeled strings for grammar induction.

challenge proposed to the machine learning community in which a set of increasingly difficult DFA induction problems have been designed. Those problems are supposed to be just beyond the current state of the art for today's DFA learning algorithms and their difficulty increases along two dimensions: the size of the underlying DFA and the sparsity of the training data. Gold [32] has shown that inferring a minimum finite state automaton compatible with given data consisting of a finite number of labeled strings is NP-complete. However, Lang [53] empirically found out that the average case is tractable. That is, randomly generated target DFAs are approximately learnable even from sparse data when this training data is also generated at random. One of the aims of the Abbadingo competition is to estimate an empirical lower bound for the sparsity of the training data for which DFA learning is still tractable for the average case. The competition has been set up as follows. A set of DFAs of various size have been randomly generated. Then, a training set and a test set have been generated from each of those DFAs. Only the labeling for the training sets have been released. Training sets are composed of a number of strings which varies with the size of the target DFA and the level of sparsity desired. The test sets are composed of a set of 1800 unlabeled strings. The goal of the competition is to discover a model for the training data that has a predictive error rate smaller than one percent on the test set. That is, at most 18 mismatches are

Problem name	Target DFA states	Target DFA depth	Strings in training set
1	63	10	3478
2	138	12	10723
3	260	14	28413
R	499	16	87500
4	68	10	2499
5	130	12	7553
6	262	14	19834
S	506	16	60000
7	65	10	1521
8	125	12	4382
9	267	14	11255
T	519	16	32500

Table 3.2: Parameters associated with the data sets for the Abbadingo DFA learning competition.

allowed between the prediction and the true labeling. Since the labeling for the test sets has not been released, the validity of a model can be tested only by submitting a candidate labeling to an "Oracle" implemented on a server at the University of New Mexico [54] which returns a "pass/fail" answer. Table 3.2 presents the different problems that compose this competition. The size and the depth of the target DFA are also provided as a piece of information to estimate how close a DFA hypothesis is from the target.

In the next sections, two algorithms are presented to address this grammar induction problem. First, the construction procedure to be used in the SAGE search algorithm is described. Then, a heuristic is presented which was first discovered by Rod Price during the competition. This heuristic uses a counting technique which results in an algorithm with much better generalization ability than the Trakhtenbrot-Barzdin algorithm [105].

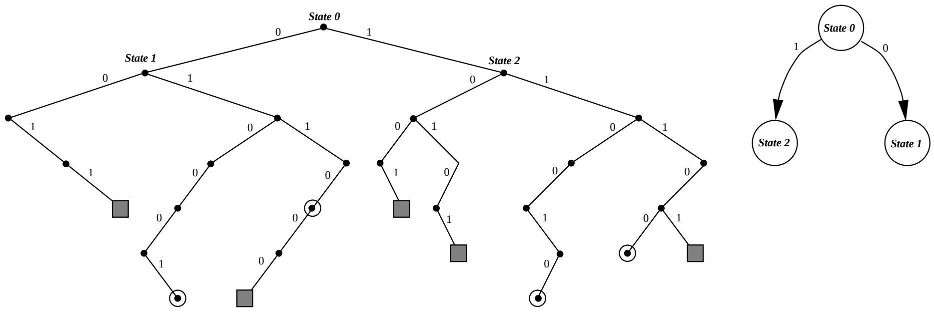
3.2.2 Description of the Implementation for SAGE

The construction procedure makes use of the state merging method described in [105]. It takes as input the prefix tree acceptor constructed from the training data. Then, a finite state automaton is iteratively constructed, one transition at a time until a valid finite automaton is generated (i.e., until every state has a "0" and a "1" outgoing transition).

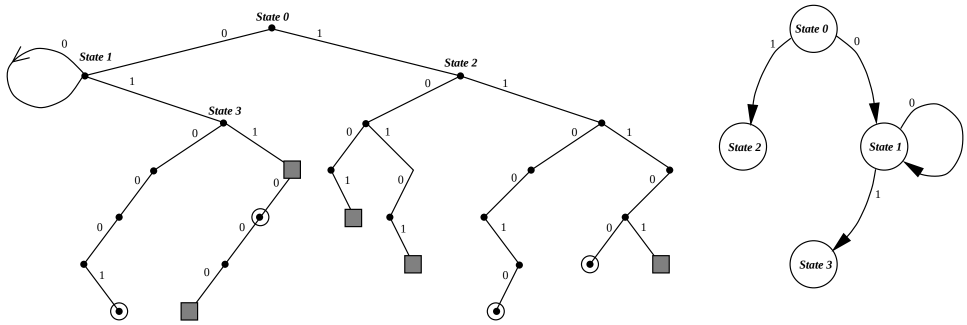
The construction procedure begins with a single state (the initial state of the DFA) which is attached to the root of the prefix tree. Then, at each step of the construction, one of the states that has no outgoing transition is selected at random and the "0" and "1" transitions are created for that state. Two cases are possible when considering a new transition: either it goes to an existing state or it goes to a newly created state. As the hypothesis DFA is constructed, states are mapped with nodes in the prefix tree and transitions between states are mapped with edges. When a new transition is created going to an existing state, corresponding nodes in the prefix tree are merged. When two nodes in the prefix tree are merged, the labels in the tree are updated accordingly and the merging of more nodes can be recursively triggered so that the prefix tree reflects the union of the labeled string suffixes that are attached to those nodes. Thus, as the DFA is constructed, the prefix tree is collapsed into a graph which is an image of the final DFA when the construction procedure is finished. This merging procedure provides the mechanism to test whether a transition between two existing states is consistent with the labeling and should be considered as a potential choice in the construction procedure. Indeed, if there is an inconsistency in the labeling when trying to merge two nodes, this means that the corresponding transition is not valid. The merging is then undone in order to restore the state of the prefix tree before the operation was performed. Figure 3.4 gives an example of the iterative construction of a DFA consistent with the training data. It shows the different steps of the collapsing of the prefix-tree into a digraph image of the hypothesis DFA. In this example, the *accepted* strings are {"0011", "011000", "1001", "10101", "11101"} and the *rejected* strings are {"010001", "0110", "110010", "11100"}.

In stage (a), the prefix-tree constructed from the training data is represented before any operation has been performed on it. Nodes corresponding to accepted strings are represented by a shaded square and nodes corresponding to rejected strings are circled. The construction of a DFA is represented on the right of the prefix-tree and starts by assigning *State 0* to the root. Then, the construction procedure creates two new states (*State 1* and *State 2*) corresponding to transitions on characters "0" and "1" respectively. In stage (b), the outgoing transitions for *State 1* are considered. The construction procedure makes the transition on character "0" loop onto *State 1*, which results in a new node labeled "accepted" (the one after transitions "11", starting from *State 1*) after the merging of the node labeled "state 1" and its left child. A new state (*State 3*) is also created for the transition on character "1". Then, in stage (c), the construction procedure creates the transition: *State 2* $\longrightarrow$ *State 3* on character "0". This results in the merging of the left child of the node corresponding to *State 2* with the node corresponding to *State 3*. The prefix-tree is then updated, taking into account the labels in the left subtree of *State 2*. Stage (d) represents the prefix-tree after the transition on character "1" from *State 2* (which loops onto itself) has been added. In stage (e), a transition from *State 3* to *State 0* on character "0" is selected by the construction procedure. Once again, this results in the merging of the left child of the node corresponding to *State 3* with the one corresponding to *State 0* and the updating of the labels by propagation. Finally, stage (f) represents the prefix-tree after the last transition has been added from *State 3* to *State 2*. At the end of the construction, states 1 and 2 are the accepting states while states 0 and 3 are rejecting. The pseudo-code describing this construction procedure is given in Figure 3.5.

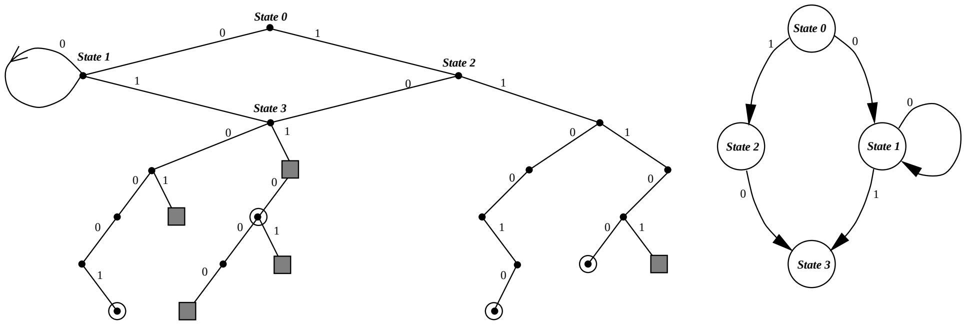
(a)


State 0
State 1
State 2

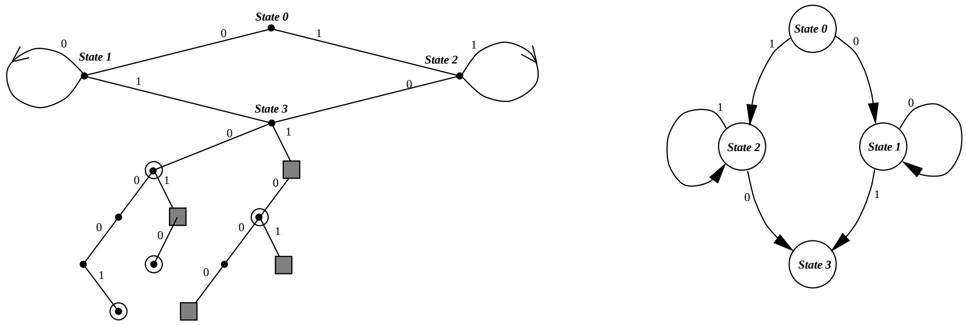
(b)


State 0
State 1
State 2
State 3

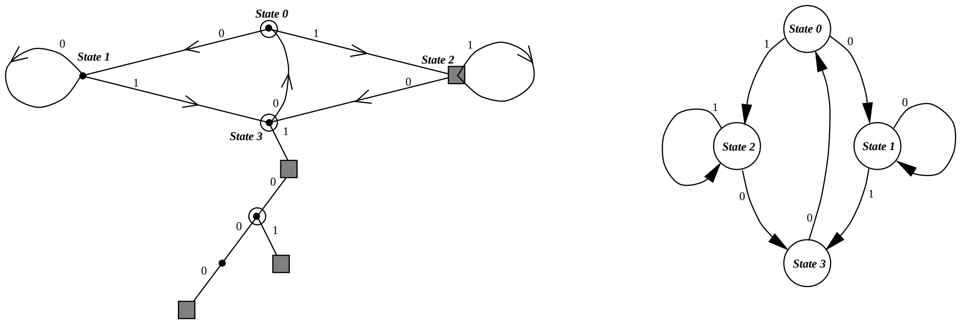
(c)


State 0
State 1
State 2
State 3

(d)


State 0
State 1
State 2
State 3

(e)



(f)

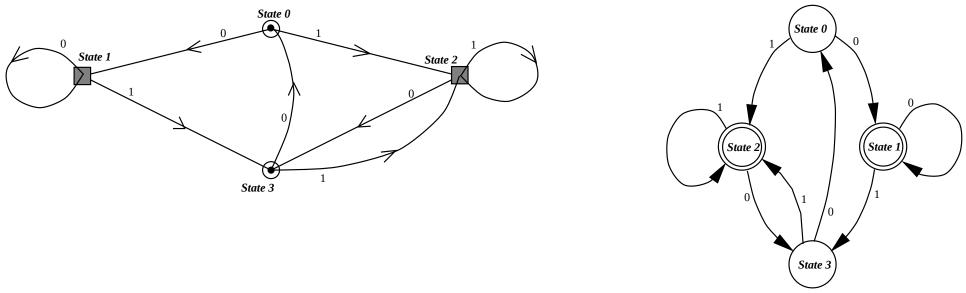


Figure 3.4: Illustration of the state merging method for DFA induction. Section 3.2.2 describes the different steps in the merge process.

```
Begin with a single state mapped to the root of the prefix tree
The list ℒ of unprocessed states consists of that single state
do
    Pick randomly a state S from ℒ

    Compute the set T_0 of valid transitions on "0" from state S
    Pick randomly a transition t_0 from T_0
    if t_0 goes to an existing state then
        Merge corresponding nodes in the prefix tree
    else
        Create a new state, map it to the corresponding node in the prefix
            tree and add it to ℒ
    endif

    Compute the set T_1 of valid transitions on "1" from state S
    Pick randomly a transition t_1 from T_1
    if t_1 goes to an existing state then
        Merge corresponding nodes in the prefix tree
    else
        Create a new state, map it to the corresponding node in the prefix
            tree and add it to ℒ
    endif
until (ℒ is empty)
/* The output is a DFA consistent with the training data */
```

Figure 3.5: Randomized construction procedure for DFA learning.

3.2.3 Description of the Evidence-Driven Heuristic

The state merging method implemented in [105] considers a breadth-first order for merging nodes, with the idea that a valid merge involving the largest sub-trees in the prefix tree has a higher probability of being correct than other merges. The evidence-driven heuristic doesn't follow that intuition and considers instead the number of labeled nodes that are mapped over each other and match when merging sub-trees in the prefix tree. Different control strategies can be designed to explore the space of DFA constructions exploiting this heuristic. Our implementation maintains a list of valid destinations for each undecided transition for the current partial DFA and, as a policy, always gives priority to "forced" creation of new states over merge operations. The pseudo-code for this algorithm is presented in Figure 3.6.

3.2.4 Experimental Results

Results for Problems in the Competition In a first stage, we have been using a sequential implementation of SAGE to address this learning problem since small populations were enough to solve the smallest instances of the Abbadingo competition. Because of the large amount of memory required to store the prefix tree and to perform operations on it, a coarse grained architecture (for which each node has usually more memory available compared to fine-grained architectures) is more appropriate for a parallel implementation. We used a network of workstations to scale the population size and address the most difficult problem instances in the competition. In particular, the solution to problems 5 and 7 involved around 16 workstations on average. This parallel implementation uses a server that manages the population of partial solutions and distributes the work load among several clients. This architecture presents the advantage that clients can be added or removed at any time.

Begin with a single state mapped to the root of the prefix tree
The list $\mathcal{S}$ of existing states in the DFA construction consists of that state
The list $\mathcal{T}$ of unprocessed transitions consists of the two outgoing transitions
    from that state, on "0" and "1"
For each $t \in \mathcal{T}$, compute:
    . the subset $\mathcal{S}_{dest}(t)$ from $\mathcal{S}$ of valid destinations for t
    . the merge count for each destination in $\mathcal{S}_{dest}(t)$

do
  Construct the subset $\mathcal{T}_0$ of transitions $t \in \mathcal{T}$ for which $\mathcal{S}_{dest}(t) = \emptyset$
  /* Transitions in $\mathcal{T}_0$ cannot go to any existing state */
  if ($\mathcal{T}_0$ is not empty) **then**
    Select $t_0 \in \mathcal{T}_0$ outgoing from the shallowest node (break ties at random)
    Remove t_0 from $\mathcal{T}$
    Create a new state S_0 mapped to the destination node for t_0 in the prefix tree
    Add S_0 to $\mathcal{S}$
    Add the two outgoing transitions from S_0, t'_0 and t'_1, to $\mathcal{T}$
    Compute $\mathcal{S}_{dest}(t'_0)$ and $\mathcal{S}_{dest}(t'_1)$ along with the corresponding merge counts
    For each transition $t \in \mathcal{T}$, add S_0 to $\mathcal{S}_{dest}(t)$ if it is a valid destination for t
      and compute its merge count
  else
    /* Operate a merge */
    Select $t_0 \in \mathcal{T}$ with the highest merge count (break ties at random)
    Merge the destination node for t_0 in the prefix tree with the destination
      state corresponding to this highest merge count
    Remove t_0 from $\mathcal{T}$
    For each $t \in \mathcal{T}$, update $\mathcal{S}_{dest}(t)$ and the merge counts
  endif
until ($\mathcal{T}$ is empty)

Figure 3.6: A DFA learning algorithm exploiting the evidence-driven heuristic.

SAGE has been able to solve problems 1, 2, 4, 5 and 7 from Table 3.2. To solve problem 7, we proceeded in two steps. First, the construction procedure described in section 3.2.2 has been extended with the evidence-driven heuristic in order to prune the search tree. The construction procedure switches to this heuristic when the number of states in the current DFA has reached a given size. Before that threshold size is reached, the construction procedure remains unchanged. After about 10 runs, a DFA with 103 states has been discovered early. Then, in a second step, more experiments were performed using the original construction procedure but starting with the same first few choices as those that had been made for the 103-state DFA. This resulted in a DFA of size 65. This second step uses SAGE for local search, starting from a good prefix for the DFA construction. The appropriate size for the prefix has been determined experimentally. It is clear from those experiments that the density for the training data available for problem 7 is at the edge of what SAGE can manage. This observation is confirmed by the analysis presented in the following section.

Table 3.3 presents the experimental setup along with the results for some of the DFAs that passed the "Oracle" test. We decided to report in this table the values for parameters that had been used when each problem was solved for the first time. Those values could be tune to improve the performance (on average up to a fourfold factor). Experiments for problems 1, 2 and 4 have been performed on a Pentium PC 200MHz. For problems 5 and 7, a network of Pentium PCs and SGI workstations has been used. The evidence-driven heuristic can solve all the problems in the first and the second group in Table 3.2 except problem 5 for which the heuristic is misled.

Procedure for Generation of Problem Instances For the experimental analysis of our algorithms, we constructed a set of problem instances, using the same procedure as for the generation of Abbadingo challenges. First, a target DFA is randomly constructed.

Problem name	1	2	4
Population size	64	64	256 (+ best of 2 samples)
Neighborhood radius for competition phase (defined with respect to Manhattan distance)	1 unit	1 unit	1 unit
Threshold value for commitment degree increment (radius neighborhood = 1 unit)	200	200	800
Number of generations	200	1600	100
Results (size of DFA model)	63 states	150 states	71 states
Execution time	1 hour (sequential)	40 hours (sequential)	4 hours (sequential)

Problem name	5	7 (step 1)	7 (step 2)
Population size	576 (+ best of 8 samples)	1024	4096
Neighborhood radius for competition phase (defined with respect to Manhattan distance)	1 unit	1 unit	1 unit
Threshold value for commitment degree increment (radius neighborhood = 1 unit)	2400	2000	10000
Number of generations	250	20	100
Results (size of DFA model)	131 states	103 states	65 states
Execution time	40 hours (parallel) (16 workstations)	2 hours (parallel) (16 workstations)	4 hours (parallel) (16 workstations)

Table 3.3: Experimental results for the SAGE search algorithm applied to problems 1, 2, 4, 5 and 7 of the Abbadingo competition.

Then, it is used to generate two sets of labeled strings (the training set and the testing set). This procedure is performed as follows:

- *Generation of target DFAs*: To construct a random DFA of nominal size n, the following procedure is performed: A random digraph with $\frac{5}{4}n$ nodes is constructed, each vertex having two outgoing edges. Then, each node is labeled "accept" or "reject" with equal probability, a starting node is picked, nodes that can't be reached from that starting node are discarded and, finally, the Moore minimization algorithm is run. If the resulting DFA's depth isn't $\lfloor(2\log_2 n) - 2\rfloor$, the procedure is repeated. This condition for the depth of DFAs corresponds to the average case of the distribution. It is a design constraint which allows the generation of a set of problems whose relative complexity remains consistent along the dimension of target size.

- *Generation of training and testing sets*: A training set for a n-state target DFA is a set drawn without replacement from a uniform distribution over the set of all strings of length at most $\lfloor(2\log_2 n) + 3\rfloor$. The same procedure is used to construct the testing set but strings already in the training set are excluded.

Comparative Performance Analysis In a comparative study, the performance of the three approaches: Trakhtenbrot-Barzdin algorithm, evidence-driven heuristic and SAGE has been evaluated against a set of random problem instances generated using the procedure described in section 3.2.4. For each target DFA, the three algorithms were evaluated across a range of density for the training data in order to observe the evolution of each approach when working with sparser data. For the first two algorithms, 1000 problems were used while only 100 problems were used to evaluate SAGE because of the requirement in computational resources. Indeed, while it takes at most one minute for the Trakhtenbrot-Barzdin algorithm or the evidence-driven heuristic to return a solution

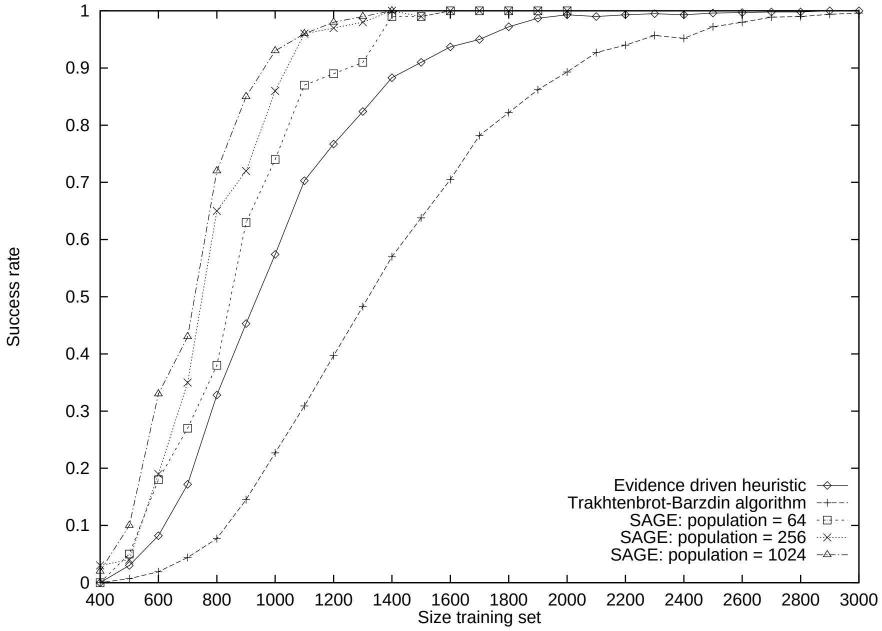


Figure 3.7: Comparison of the performance for the evidence-driven heuristic, the Trakhtenbrot-Barzdin algorithm and SAGE on the task of grammatical inference with randomly generated target DFAs of nominal size 32.

with a target DFA of size 64, three hours on average are necessary for SAGE with a population of size 1024. This comparison has been performed for three values of the population size for SAGE: 64, 256 and 1024 and for two values of the target nominal size: 32 and 64 states (Figures 3.7 and 3.8 respectively).

In those experiments, the performance is the ratio of problems for which the predictive ability of the model constructed by the algorithm is at least 99% accurate (i.e., error rate smaller than 1%). This threshold is the same as the success criterion for solving problems in the Abbadingo competition. Figures 3.7 and 3.8 show the dependence of SAGE on the population size for its performance. Indeed, a larger population results in a better reliability for the control of the focus of the search because of a larger sample. For the set of problems generated for the purpose of this analysis, SAGE and

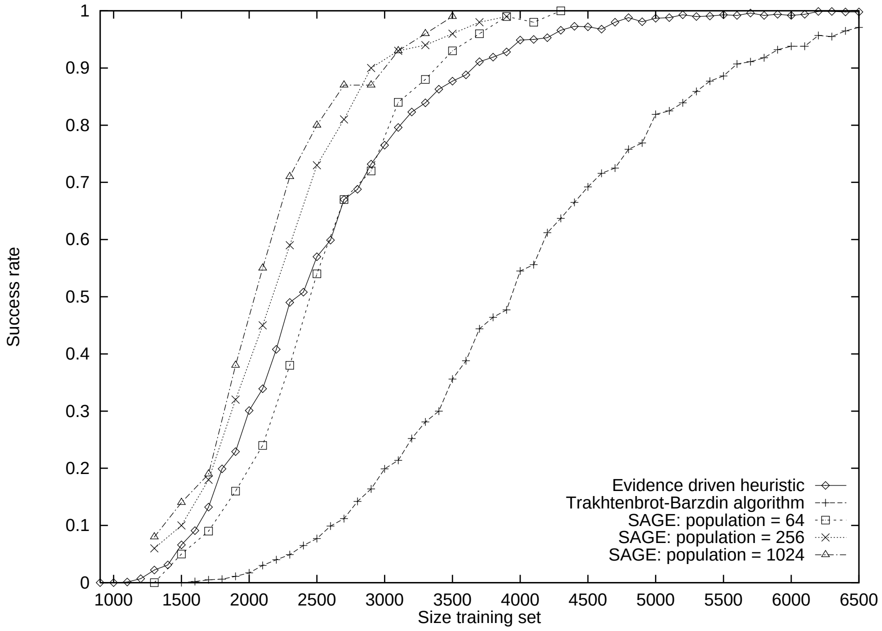


Figure 3.8: Comparison of the performance for the evidence-driven heuristic, the Trakhtenbrot-Barzdin algorithm and SAGE on the task of grammatical inference with randomly generated target DFAs of nominal size 64.

the evidence-driven heuristic clearly outperform the Trakhtenbrot-Barzdin algorithm. With a population size large enough, SAGE also exhibits a performance consistently better than the evidence-driven heuristic. However, it is difficult to compare those two approaches since SAGE is a general purpose search algorithm while the other is a greedy algorithm using a problem-specific heuristic. For this reason, SAGE doesn't scale up as well as the evidence-driven heuristic (or the Trakhtenbrot-Barzdin algorithm) for larger target DFAs. The introduction of problem-specific heuristics in the construction procedure becomes necessary for SAGE to address this scaling issue.

3.3 Application 2: Discovery of Short Constructions for Sorting Networks

3.3.1 Description

An *oblivious comparison-based algorithm* is such that the sequence of comparisons performed is the same for all inputs of any given size. This kind of algorithm has received much attention since it admits an implementation as circuits: comparison-swap can be hard-wired. Such an oblivious comparison-based algorithm for sorting n values is called an n-input *sorting network* (a survey of sorting network research is in [48]).

There is a convenient graphical representation of sorting networks as shown in Figure 3.9, which is a 10-input sorting network (from [48]). Each horizontal line represents an input of the sorting network and each connection between two lines represents a *comparator* which compares the two elements and exchanges them if the one on the upper line is larger than the one on the lower line. The input of the sorting network is on the left of the representation. Elements at the output are sorted and the largest element migrates to the bottom line.

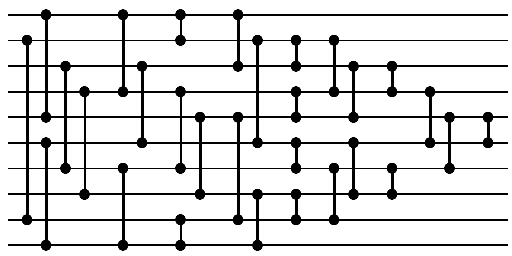


Figure 3.9: A 10-input sorting network using 29 comparators and 9 parallel steps.

Performance of a sorting network can be measured in two different ways:

1. Its *depth* which is defined as the number of parallel steps that it takes to sort any

input, given that in one step disjoint comparison-swap operations can take place simultaneously. Current upper and lower bounds are provided in [69]. Table 1 presents these current bounds on depth for $n \leq 16$.

Inputs	1	2	3	4	5	6	7	8
Upper	0	1	3	3	5	5	6	6
Lower	0	1	3	3	5	5	6	6
Inputs	9	10	11	12	13	14	15	16
Upper	7	7	8	8	9	9	9	9
Lower	7	7	7	7	7	7	7	7

Table 3.4: Current upper and lower bounds on the depth of n-input sorting networks.

2. Its *length*, that is the number of comparison-swap used. Optimal sorting networks for $n \leq 8$ are known exactly and are presented in [48] along with the most efficient sorting networks to date for $9 \leq n \leq 16$. Table 2 presents these results.

   The 16-input sorting network has been the most challenging one. [48] recounts its history as follows. First, in 1962, Bose and Nelson discovered a method for constructing sorting networks that used 65 comparisons and conjectured that it was best possible. Two years later, R. W. Floyd and D. E. Knuth, and independently K. E. Batcher, found a new method and designed a sorting network using 63 comparisons. Then, a 62-comparator sorting network was found by G. Shapiro in 1969, soon to be followed by M. W. Green's 60 comparator network (see [36] and [48]).

3.3.2 Related work

Since [48], no improvement has been made regarding the upper bound for the length of sorting networks for $n \leq 16$. Some attempts have been made to improve the results in Tables 1 and 2, but these attempts are limited by the number of computer resource

Inputs	1	2	3	4	5	6	7	8
Comparators	0	1	3	5	9	12	16	19
Inputs	9	10	11	12	13	14	15	16
Comparators	25	29	35	39	45*	51	56	60

Table 3.5: Best upper bounds currently known for length of sorting networks. Previously, the best known upper bound for the 13-input problem was 46.

available.

A well-known work in the field of evolutionary computation is Hillis' [38] which addressed the problem of finding 16-input sorting networks of short length. However, despite a large amount of computer resource (a few hours on a 64K processors CM2) and a strong bias (the search algorithm was initialized with the first 32 comparators of Green's construction), the best result achieved was a 61-comparator sorter, one more than Green's solution.

Parberry [69] addressed the problem of determining the smallest depth for $n = 9$ and $n = 10$. His work used an exhaustive search and required 200 hours on a Cray-2.

3.3.3 Implementation

A randomized construction procedure (for which a pseudo-code is presented in Figure 3.10) is run by each processor element in the population to generate valid sorting networks. This iterative procedure determines at each step the list of *significant* comparators. A *significant* comparator is one that swaps inputs for at least one input sequence of the network. Then, one of them is randomly selected and the procedure is repeated until the list of significant comparators is empty, meaning that a valid sorting network has been constructed. A run of this algorithm corresponds to the construction of a path in the tree representing all valid and fair sorting networks; that is, valid sorting networks with no useless comparators. The score of processor elements is defined as the

Begin with an empty or a partial network N
do
  Compute the set $\mathcal{S}$ of significant comparators
  if $\mathcal{S}$ is not empty **then**
    Pick randomly a comparator from $\mathcal{S}$ and append it to N
  endif
until ($\mathcal{S}$ is empty)
/* Now N is a valid and fair sorting network */

Figure 3.10: Randomized construction procedure run by each processor element.

length of the sorting network that has been constructed during the construction phase. Eventually, ties are broken using the number of parallel steps in the sorting networks.

Valid sorting networks are built using the *zero-one principle*. This principle tells us that one only needs to consider all 2^n possible binary inputs instead of the $n!$ permutations of n distinct numbers. The algorithm in Figure 3.11 describes the details of the principal operations introduced in the pseudo-code of Figure 3.10. This last algorithm uses a direct implementation of the zero-one principle (i.e. manages a list of the 2^n vectors). Since the complexity of computing the set of significant comparators and of updating the list of unsorted vectors is a function of the size of the representation for the set of unsorted vectors, if the structure used to represent this set is the explicit list of those vectors (using a list or an array structure), this makes the complexity for the randomized construction algorithm a function of 2^n. In fact, we have been able to improve the complexity of the construction algorithm by making it a function of Φ^n instead of 2^n where Φ is the *golden ratio* for Fibonacci numbers: $\Phi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$. This has been done by representing the set of unsorted vectors as a list of classes instead of an explicit list of those vectors. The details of this algorithm and the proof for the complexity are presented in [47]. The complexity remains exponential but it is unlikely that an algorithm exist that can construct *any* valid sorting network with a better performance (i.e. smaller than exponential) since verifying the validity of sorting

Init

/* A sorted vector is a sequence of 1's starting from the least significant bit and all the remaining bits are 0's */

$unsorted\_vectors = \{1, 2, \ldots, 2^n - 1\} -$
$\{2^0, 2^0 + 2^1, 2^0 + 2^1 + 2^2, \ldots, 2^0 + 2^1 + \ldots + 2^{n-1}\}$

Compute the set $\mathcal{S}$ of significant comparators

$\mathcal{S} = \emptyset$
for $i = 1$ **to** $n - 1$ **do**
  for $j = i + 1$ **to** n **do**
    if there is an element $e \in unsorted\_vectors$ for which
        the i^{th} bit is a 0 and the j^{th} bit is a 1 **then**
      /* The pair (i, j) corresponds to a significant comparator */
      $\mathcal{S} = \mathcal{S} \bigcup \{(i, j)\}$
    endif
  endfor
endfor

Update of *unsorted_vectors* after comparator (i,j) has been selected from $\mathcal{S}$

$new\_unsorted\_vectors = \emptyset$
for $e \in unsorted\_vectors$ **do**
  /* Apply comparator (i,j) to vector e */
  $e' \leftarrow e$
  if the i^{th} bit of e is a 0 **and** the j^{th} bit is a 1 **then**
    $e' \leftarrow$ switch the i^{th} and the j^{th} bits of e
  endif
  if e' is not a sorted vector **and** $e' \notin new\_unsorted\_vectors$ **then**
    $new\_unsorted\_vectors = new\_unsorted\_vectors \bigcup \{e'\}$
  endif
endfor
$unsorted\_vectors \leftarrow new\_unsorted\_vectors$

Figure 3.11: Detail of the principal operations performed by the non-deterministic incremental algorithm, using a direct implementation of the zero-one principle.

	13-input problem	16-input problem
Population size	65,536 each processor implements 16 processor elements	65,536 each processor implements 16 processor elements
Neighborhood radius for competition phase (defined with respect to Manhattan distance)	2 units	2 units
Threshold value for commitment degree increment (radius neighborhood = 1 unit)	60000 or 80000	60000 or 80000
Number of generations	350 to 450	500 to 650
Results	Number of runs: 10 For 6 runs: 45 comparators For 4 runs: 46 comparators	Number of runs: 10 For 5 runs: 60 comparators For 3 runs: 61 comparators For 2 runs: 62 comparators
Execution time	about 3 hours for each run	about 12 hours for each run

Table 3.6: Experimental results for the SAGE search algorithm applied to the 13-input and the 16-input sorting network problem.

networks is intractable [45]. However, this improvement for the base of the exponent resulted in a significant speed up for our experiments (e.g., a factor of about four in the construction procedure for 16-input sorting networks).

3.3.4 Experimental Results

Experiments were performed on a Maspar MP-2 parallel computer. The configuration of our system is composed of $4K$ processors. The peak performance of this system for 32-bit integer computation is $17,000$ Mips. In the maximal configuration a MP-2 system has $16K$ processors and a peak performance of $68,000$ Mips. Each processor implements one or several of the elementary processor elements of the SAGE search algorithm, allowing a population size of $4K$ or larger (we have been working with population size up to $64K$).

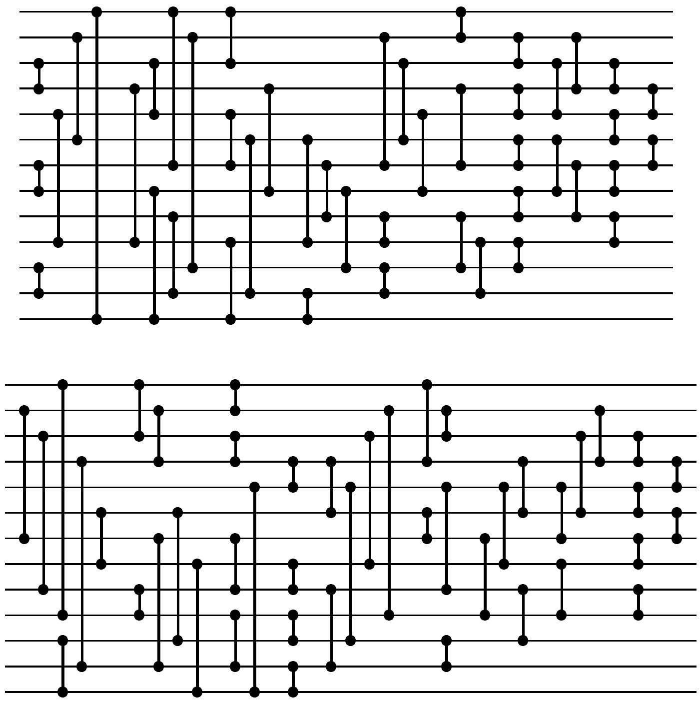


Figure 3.12: Two 13-input 45-comparator sorting networks using 10 parallel steps.

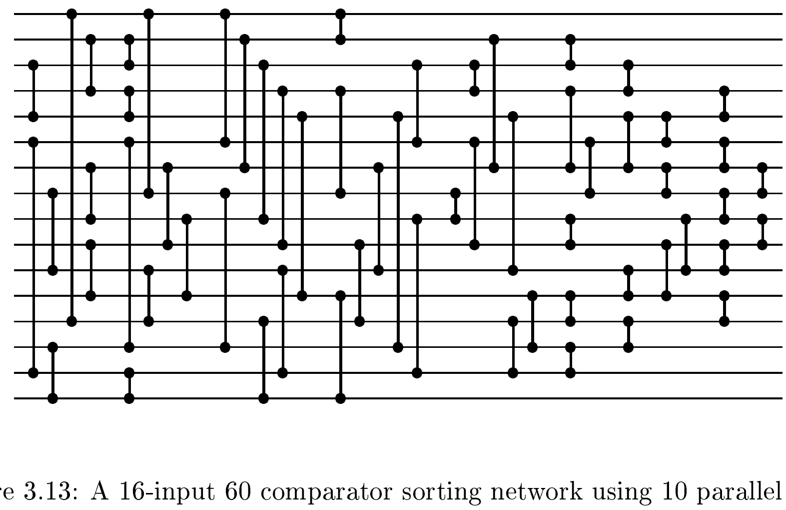


Figure 3.13: A 16-input 60 comparator sorting network using 10 parallel steps.

SAGE has been able to find all best known upper bounds for the length of sorting networks (for a number of inputs up to 16) and it even improved the upper bound for the 13-input problem. Table 3.6 presents parameters and results of experiments for the 13-input and the 16-input problems.

For the 13-input problem, SAGE discovered a few sorting networks using 45 comparators (one comparator less than the best current known) and 10 parallel steps. Two of them are presented in Figure 3.12. For the 16-input case, a few 60 comparator sorting networks have also been discovered, each of them using 10 parallel steps. This is as good as the best human-built sorting network designed by M. W. Green. Figure 3.13 presents one of these constructions.

3.4 Application 3: The Solitaire Game

3.4.1 Presentation of the game

This third problem is a one-player game which can also be stated as a combinatorial optimization problem. The purpose of the game is to discover a sequence of moves which is as long as possible. We decided to study this game because it is an interesting example of a problem for which no heuristic is known to prune efficiently the search tree and it is not immediately amenable to search by evolutionary strategy algorithms like GAs or GP. There is no previous research work in the literature about that problem that could be compared to the experimental results presented in this section. Therefore, the performance of SAGE on that particular problem has been evaluated against the performance of "experimented" human players.

To play this game, one only needs a piece of paper with a grid layout and a pencil. First, the player draws a fixed initial pattern composed of crosses, like the left picture in Figure 3.14. The rule is that a cross can be added at an intersection of the

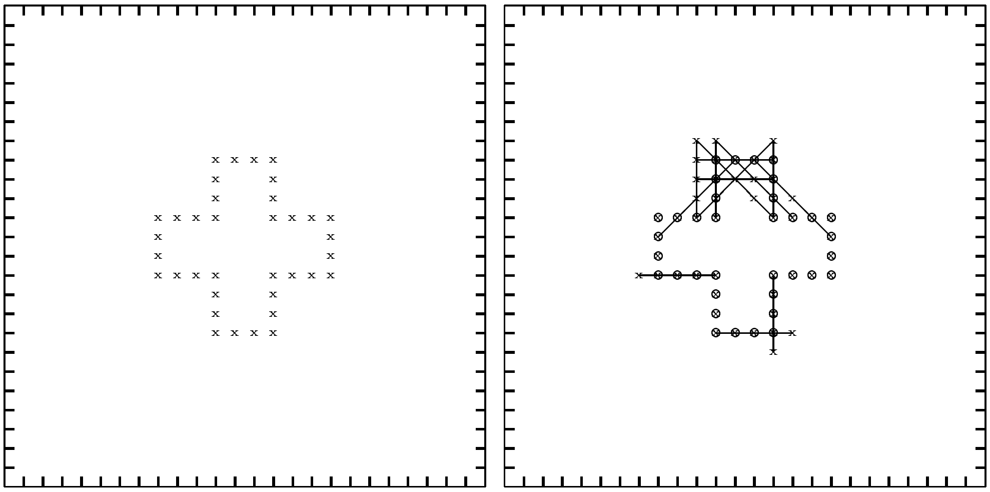


Figure 3.14: The initial configuration and a possible configuration after 13 moves for the Solitaire game. For a clearer picture, the grid layout is not drawn but is represented by the rules on the border.

grid layout if and only if this cross allows the drawing of a line composed of 5 consecutive crosses that do not overlap another line. This line may however be the continuation of another line or may cross another line. That is, the new line can share at most one cross with any other line. This new line can be drawn vertically, horizontally or diagonally. The right picture in Figure 3.14 shows a possible configuration of this game after a few moves. Crosses of the initial pattern are circled in order to distinguish them from the ones added by the player. Now, the goal of this game is simply to draw as many lines as possible! If this game is played by hand, one can see that a good strategy is difficult to elaborate. After a few games, a score of 70-80 lines is relatively common. However, to reach 90 lines is less obvious and a score greater than 100 lines is exceptional. Also, it can be proved that the maximum number of moves for this game is finite. However, no tight upper bound has been established for this optimum.

Population size	4,096
Neighborhood radius for competition phase (defined with respect to Manhattan distance)	1 units
Value of the threshold for commitment degree increase (radius neighborhood = 1)	4 values have been tested: 4500, 5000, 5500, 6000
Number of generations	from 2000 (threshold = 6000) up to 5000 (threshold = 4500)
Results	Number of runs: 5 for each value of the threshold For threshold = 4500: 103, 111, 111, 112, 112 For threshold = 5000: 108, 112, 113, 117, 118 For threshold = 5500: 113, 117, 118, 119, 122 For threshold = 6000: 103, 104, 106, 111, 115
Execution time	from 6 hours (threshold = 6000) to 12 hours (threshold = 4500)

Table 3.7: Experimental results for the SAGE search algorithm applied to the Solitaire game.

3.4.2 Implementation

The construction procedure is a direct implementation of the rules of the game. At each step of the construction, a choice is made from the list of all valid moves given the current configuration of the board.

3.4.3 Experimental Results

Table 3.7 presents the parameters and the results that have been achieved for a set of experiments. Those experiments have been performed on the MasPar computer described in section 3.3.4. So far, the best result is a 122 lines game configuration. Figure 3.15 shows the final configuration along with the configuration of the game after the first 40 moves. It is interesting to notice that most of the moves at the beginning of the game are located in the same area of the board. This strategy is one that we also discovered

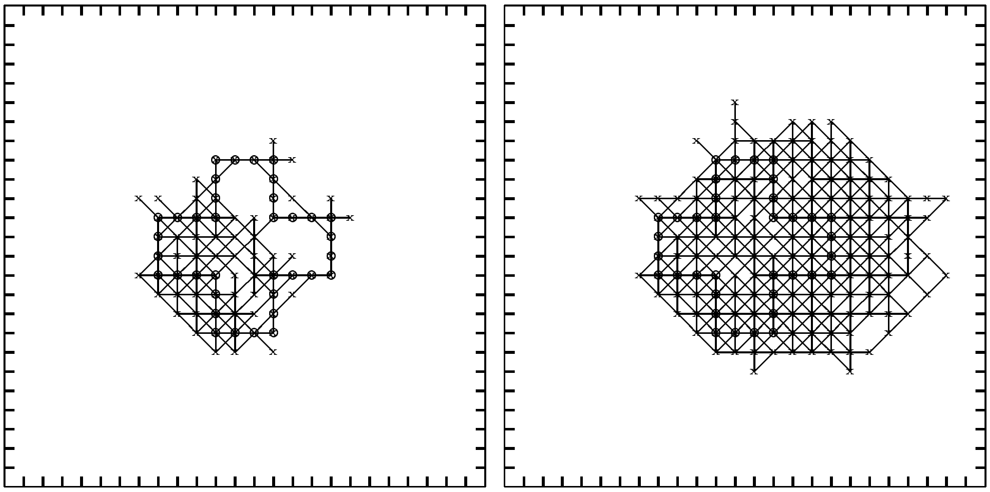


Figure 3.15: The best configuration found out for the Solitaire game, using 122 lines (on the right); and the configuration for this best play after 40 moves (on the left).

in order to achieve a good score while playing by hand.

3.5 Discussion

As discussed in section 3.1.2, the performance of SAGE relies heavily on the construction procedure. Indeed, the construction procedure determines the distribution of solutions when the search space is sampled. In practice, a few tries are often required to discover an appropriate procedure which results in a good performance. As a rule of thumb, the construction procedure should be designed so that early choices result in a reliable discrimination between poor domains of the search space and domains that are worth being explored. For instance, in the case of DFA learning, an incorrect merge performed early in the construction of a DFA results in the propagation of more constraints than an incorrect merge performed a few iterations later. Therefore, a mistake in early choices is likely to result in the construction of DFAs whose size is much larger than the target (and thus will have poor predictive ability).

The population-based model allows however some flexibility for the design of this procedure. Indeed, the ideal case would be that the score of each alternative be a reliable information how to focus the search. Then, the selection of the best alternative would be the appropriate way to continue the search. However, the availability of such reliable information is unlikely for two reasons. First, the score evaluation can be subject to a large variance due to the sampling technique itself. Second, the actual distribution of solutions might not exhibit this nice "greedy choice" property. That is, the expected value for a distribution is not representative of the best element for that distribution, with respect to the distributions associated with the other alternatives.

However, because of the parallel exploration of several alternatives, SAGE is more robust against such flaws. By adjusting the threshold that controls the increment of the commitment degree, several high quality alternatives can be maintained in the population until one of them consistently outperforms the others. An example which shows the necessity of adjusting correctly this threshold is shown in Figure 3.16. We conducted experiments to measure the probability of constructing a 10-input sorting network with optimum length (that is using 29 comparators). The population size was set to $4,096$ and the neighborhood radius for the competition phase was set to 1. Then, 20 runs were performed for each value of the threshold. Figure 3.16 shows that as the value of the threshold decreases, allowing the population to converge to a reduced number of alternatives, the success rate of the search increases. However, when forcing the population to converge even more by continuing to decrease the value of the threshold, the success rate starts decreasing. This means that the distribution of solutions doesn't exhibit the "greedy choice" property for that particular construction procedure. This example shows the necessity of achieving a balance between exploration and exploitation.

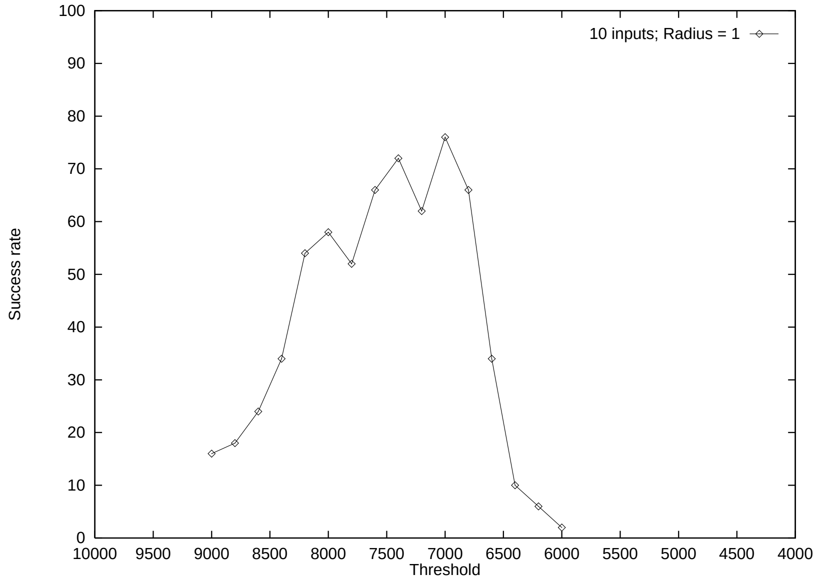


Figure 3.16: Example for the evolution of the success rate with respect to the value of the threshold that controls the increment of the commitment degree.

3.6 Concluding Remarks

This chapter presents SAGE, a new algorithm for search in trees and directed acyclic graphs which is based on a random sampling strategy to evaluate the score of internal nodes and the management of a population to allow the search to focus on most promising alternatives in an adaptive manner. This distributed stochastic algorithm admits a parallel implementation on both fine-grained and coarse grained distributed systems because of the loose central control. Indeed, the only information gathered at the level of the population is the measure of diversity which is used to determine when the commitment degree should be increased.

We have described the application of SAGE on three difficult problems. For the sorting network problem, this search algorithm has rediscovered the best upper bound for the shortest constructions (up to 16-input sorters) and improved one of these upper bounds by one comparator. For the Solitaire game, the performance is comparable to the one of a human player who has a good expertise of the game. We haven't been able to discover any problem-specific heuristics that would approach the results achieved by SAGE for that game. Finally, the performance of SAGE has been analyzed on the problem of DFA learning, inspired from the recent Abbadingo competition. Those experiments have shown that for average size target DFAs (on the order of 64 to 128 states) SAGE compares favorably to the well-known Trakhtenbrot-Barzdin algorithm and to a new evidence-driven heuristic. However, as the size of the target DFA increases, SAGE doesn't scale up and requires a prohibitive amount of computer resource. To search such a large state space, the introduction of problem-specific heuristics becomes necessary.

We have seen that the construction procedure plays an important role in the performance of search. This procedure must be designed carefully in order to exhibit some properties appropriate for SAGE. Some problem specific knowledge and heuristics can also be introduced when designing this procedure in order to reduce the size of the search tree and to drive the search in a particular direction.

The class of problems that can be successfully tackled by SAGE has been referred to in this dissertation as *sequential construction problems*. The construction procedure associated with those problems defines an *explicit partitioning* of the space of candidate solutions by constructing a hierarchy of subsets over this space. The examples of sequential construction problems that have been introduced in this chapter present the property that early decisions in the construction process contribute significantly to the expected quality of solutions constructed starting with those initial decisions. This property results in specific statistical regularities for the distribution of solutions in the search space that can be captured by the hierarchical partitioning associated with the construction procedure. SAGE implements a strategy that exploits this specific source of information in order to drive the search.

The motivation for the design of SAGE is that the performance of traditional search algorithms depends on the definition of heuristics that encode some specific regularities associated with the problem domain. As illustrated in this chapter, such problem-specific knowledge cannot always be formalized and is sometimes better stated in term of statistical properties. Like SAGE, EC techniques can also be seen as tools to perform statistical inference. However, sequential construction problems are not very appropriate for an encoding in the framework of EC techniques because they are highly constrained. As a result, the contribution of any component of a representation to the total quality (or fitness) of the solution is highly dependent on the other components of this representation. This feature, known as epistasis, results in a ill-structured search space which makes difficult for Evolutionary Algorithms to exploit the specific statistical properties associated with the domain of sequential construction problems.

One important contribution of this work is to provide some insights concerning the domain of stochastic search algorithms. The general idea for search algorithms is to exploit some information about the search space. This information can be provided either explicitly (hand-coded knowledge) or extracted by performing some form of statistical analysis. Then, this knowledge is exploited to decide how to focus the search. However, the actual distribution of solutions in the search space is a source of important information that has rarely been exploited in the context of tree exploration. We believe that SAGE is a first example of a tree search algorithm able to exploit this information efficiently.

Chapter 4

Coevolution and the Emergence of Adaptability

Chapter 2 describes the *indirect partitioning methodology* as a framework to capture some specific statistical properties associated with a category of ill-structured multi-objective problems. Indeed, when little information is available about the properties of a problem domain, the design of heuristics is not possible and no explicit structure can be identified *a-priori* in order to support a statistical inference process. However, if candidate solutions are evaluated with respect to some set of test cases, then an extra piece of information is available. Indeed, this feature allows the definition of structures over the state space which may then be exploited in order to capture some regularities and drive the search process.

As discussed in section 2.1.3, the indirect partitioning methodology defines subsets of candidate solutions by *intention*. In general, there is no direct procedure to construct the members of any specific subset in this decomposition. Therefore, some special mechanisms must be implemented in order to exploit properties that are captured by structures constructed from those subsets. In section 2.1.3, the problem of constructing an S-expression to tell apart two intertwined spirals was introduced in order to illustrate this issue: given a subset of points from the training set, there exists no general purpose computational procedure returning an S-expression that classifies accurately those test cases. Instead, some strategy must be defined to explore the space of S-expression until such a candidate is identified. Moreover, different types of regularities or statistical properties may be exhibited by the implicitly defined structure. Therefore, depending on the properties a search algorithm attempts to exploit, some specific strategies must also be defined.

The central strategy underlying the approach introduced in the next chapter of this dissertation concerns the search for domains of the state space over which continuous progress can be observed. *Coevolution*, a paradigm involving the embedding of evolving agents in an environment that responds to their progress, is proposed as the basic concept to implement this strategy. Indeed, continuous progress concerns a specific property exhibited by individuals, namely: their ability to adapt in order to solve increasingly difficult problems. Therefore, the principles of operation underlying coevolution make this paradigm a good candidate for capturing this dynamic property.

The next section presents a general description of the concept of coevolution. It also introduces our motivations for exploiting this particular paradigm. This chapter considers a methodology that describes coevolution from the perspective of search algorithms. Indeed, the fundamental mechanisms underlying coevolution are still poorly understood. Therefore, we believe that following a methodology that attempts to identify the underlying search strategies embedded in different coevolutionary models will contribute to a better understanding of this paradigm. Moreover, since coevolution covers a large variety of evolutionary models, the definition of a single formal framework that captures all the different features associated with coevolution may not be the most appropriate approach for the study of this paradigm. This observation motivates the experimental rather than theoretical methodology followed in this chapter. Then, a description of important landmarks in the research literature concerning the applications of coevolution concludes this chapter.

4.1 Coevolution as a Paradigm for Capturing Dynamic Properties of Evolving Agents

In coevolution, individuals are evaluated in an environment which responds to their progress (instead of a fixed environment or a fixed objective function). This definition encompasses a large variety of possible models depending on the interactions between agents and the environment in which they are evaluated (which could in fact be represented as another population of evolving agents). In the extreme, even fitness sharing (or resource sharing) models [91, 46] could be considered as models of coevolution since they satisfy that definition (agents are competing for resources and their fitness depends on the performance of other agents with respect to the different resources). In fact, it seems that an exact definition of coevolution is not easy to state and different research communities might not agree on the same definition. Even in natural evolution, coevolution is not that easy to identify [43].

For the following, the term *coevolutionary learning* will denote a learning procedure involving the coevolution of two populations: a population of *learners* and a population of *problems* such that continuous progress is achieved on the long term. Figure 4.1 illustrates the interactions between the two populations in the canonical model of coevolutionary learning. Arrows represent the contribution of the members of each population to the fitness of individuals in the other population.

The central contribution of the next chapter is to propose and implement a

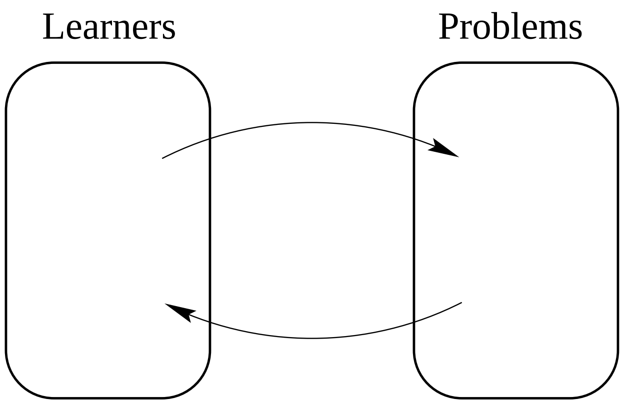


Figure 4.1: Coevolution between a population of learners and a population of problems.

strategy that searches for domains of a state space over which continuous progress can be observed. This means that the evaluation of individuals is a function of their *ability to progress* and is no longer determined by some well-defined static objective function. More precisely, from the perspective of this research work, the ability to progress is defined as the likelihood that successive transformations of a candidate solution result in solutions of increasing performance with respect to the task. In the case of multi-objective problems, which are the focus of this research work, this means that individuals cover accurately an increasing proportion of a set of test cases.

The ability to progress is a dynamic property whose exact evaluation by a computational procedure is usually too expensive in terms of computer cycles. Therefore, some alternative approaches should be investigated in order to estimate this property. As stated in the beginning of that section, coevolutionary models involve the evaluation of individuals in a dynamic environment. This means that individuals adapting faster to changes in the environment get an evolutionary advantage in the race against the other members of the population. Therefore, coevolution provides an appropriate framework to capture the notion of *adaptability*. The idea of coevolutionary race is a recurrent theme in the EC community and is often presented as a potential solution for the open-ended emergence of continuously improving solutions [27]. However, as illustrated in the following sections, adaptability doesn't necessarily means emergence of continuous progress.

In the next sections, the behavior of different coevolutionary models is studied experimentally in order to identify the important issues that are relevant to this search paradigm. This work is based on a case study: the discovery of cellular automata rules that implement the majority classification task.

4.2 Dynamics of Coevolution Between Two Populations: a Case-Study

4.2.1 Motivations

The goal of this section is to present to the reader a sample of the variety of behaviors that can be observed in a coevolutionary setup. This analysis is based on an experimental study illustrated with a combinatorial optimization problem: the discovery of cellular automata rules to implement the majority classification task.

Multiple reasons motivate the choice of that particular problem. First, while being stated as an optimization problem, this problem exhibits several features that are common to many learning tasks. For instance, the evaluation of candidate solutions is based on their performance against an environment which offers a potentially infinite number of training examples. Also, this problem can be stated as an inductive learning problem: the goal is to construct a model that covers with high accuracy a set of test cases and eventually generalizes to instances outside the training set by capturing some intrinsic property of the problem domain. For this reason, the concepts of search and learning are used in an inter-changeable manner in this experimental study. Finally, the encoding of this problem exhibits some interesting properties that allow the construction of simple diagrams to illustrate the different issues involved in coevolution.

The next section describes this problem along with a background about related research work. As discussed previously, there are several different alternatives to define the interactions between two coevolving populations. Depending on those interactions, the resulting dynamics can be extremely different. In a preliminary stage, this problem is addressed in the framework of a simple evolutionary model. The purpose of this first step is to provide a reference for the qualitative analysis of the coevolutionary models that follow.

4.2.2 Description of the problem

One-Dimensional Cellular Automata

A one-dimensional cellular automaton (CA) is a linear wrap-around lattice composed of N cells in which each cell can take one out of k possible states. A rule is defined for each cell in order to update its state. This rule determines the next state of a cell given its current state and the state of cells in a predefined neighborhood. For the model discussed in this dissertation, this neighborhood is composed of cells whose distance is at most r from the central cell. This operation is performed synchronously for all the cells in the CA. For the work presented in this dissertation, the state of cells is binary ($k = 2$), $N = 149$ and $r = 3$. This means that the size of the rule space is $2^{2^{2*r+1}} = 2^{128}$.

Cellular automata have been studied widely as they represent one of the simplest systems in which complex emergent behaviors can be observed. This model is very attractive as a means to study complex systems in nature. Indeed, the evolution of such systems is ruled by simple, locally-interacting components which result in the emergence of global, coordinated activity.

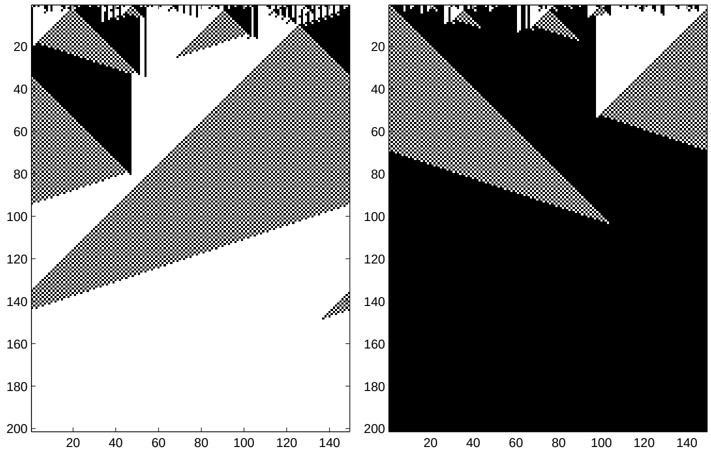


Figure 4.2: Two space-time diagrams for the GKL rule.

The Majority Classification Task

The task consists in discovering a rule for the one-dimensional CA which implements the majority function as accurately as possible. This is a density classification task, for which one wants the state of the cells of the CA to relax to all 0's or 1's depending on the density of the initial configuration (IC) of the CA, within a maximum of M time steps. Following Mitchell, Crutchfield and Hraber [59], ρ_c denotes the threshold for the classification task (here, $\rho_c = 1/2$), ρ denotes the density of 1's in a configuration and ρ_o denotes the density of 1's in the initial configuration. Figure 4.2 presents two examples of the space-time evolution of a CA for the Gacs-Kurdyumov-Levin (GKL) rule, with $\rho_0 < \rho_c$ in the left diagram and $\rho_0 > \rho_c$ in the right one. For each diagram, the initial configuration is at the top and the evolution in time of the state of the CA is represented downward.

The task $\rho_c = 1/2$ is known to be difficult. In particular, it has been proven that, given a radius r and a sufficiently large N, there exists no rule such that the CA relaxes to the correct state for all possible ICs [52]. Indeed, the density is a global

N	149	599	999
Das rule	0.823 +/- 0.001	0.778 +/- 0.001	0.764 +/- 0.001
ABK rule	0.824 +/- 0.001	0.764 +/- 0.001	0.730 +/- 0.001
GKL rule	0.815 +/- 0.001	0.773 +/- 0.001	0.759 +/- 0.001

Table 4.1: Performance of different published CA rules for the majority classification task.

property of the initial configuration while individual cells of the CA have access to local information only. Discovering a rule that will display the appropriate computation by the CA with the highest accuracy is a challenge, and the upper limit for this accuracy is still unknown. Table 4.1 describes the performance for that task for different published rules and different values of N. The GKL rule was designed in 1978 for a different goal than solving the $\rho_c = 1/2$ task [59]. However, for a while it provided the best known performance. Mitchell, Crutchfield and Hraber [59] and Das, Mitchell and Crutchfield [22] used Genetic Algorithms (GAs) to explore the space of rules. The main purpose of this work was to study the evolution of the different strategies that were discovered by the Genetic Algorithm to address the task. In particular, some epochs were observed such that a transition between two epochs would often correspond to a significant evolution in the degree of complexity of the underlying strategy implemented by the new rules. The concept of "particles" has also been introduced to describe at a higher level of abstraction the dynamics of the CA and, in some cases, the corresponding strategy implemented by the rule to address the task. The GKL and Das rules are human-written while the Andre-Bennett-Koza (ABK) rule has been discovered using the Genetic Programming paradigm [4]. While this last rule improved the case $N = 149$, it doesn't generalize as well as the GKL or the Das rule. More recently, Paredis [74] described a coevolutionary approach to search the space of rules and showed the difficulty of coevolving consistently two populations toward continuous improvement. A coevolutionary approach has also been studied by Sipper [100] for the exploration of rules for non-homogeneous CA. In that particular CA model, each cell has its own independent version of a rule. Capcarrere, Sipper and Tomassini [14] also reported that by changing the specification of the convergence pattern, from all $0's$ or all $1's$ to a pattern in which a block of at least two consecutive $1's$ exists if and only if $\rho_0 > 1/2$ and a block of at least two consecutive $0's$ exists if and only if $\rho_0 < 1/2$, then a two-state, $r = 1$ CA exists that can perfectly solve the density problem in $\lceil N/2 \rceil$ time steps.

For the $\rho_c = 1/2$ task, there is some strong evidence that the best rules are in the domain of the rule space with density close to 0.5. An intuitive argument to support this hypothesis is presented in [60]. It is also believed that the most difficult ICs are those with density close to 0.5 (since a few mutations are enough to switch from $\rho_0 < 1/2$ to $\rho_0 > 1/2$, and vice versa).

4.2.3 Learning in a Fixed Environment

The traditional methodology for evolutionary approaches to problem solving consists in designing a representation for solutions and a fitness function. The absolute performance of individuals is evaluated with respect to that fitness function and operators are used to explore the state space, exploiting the current best solutions to focus the search. Usually, a lot of effort goes into the design of the fitness function and/or the search operators in order to take advantage of problem specific knowledge. When such knowledge is unavailable about the structure or the regularities of the problem, the performance of a canonical evolutionary approach is usually very limited.

In order to have a reference to compare the dynamics and the performance of different models, consider the following setup to address the $\rho_c = 1/2$ task based on a direct implementation of an evolutionary algorithm. This implementation is similar to the one described in [59]. Each rule is coded on a binary string of length $2^{2*r+1} = 128$.

One-point crossover is used with a 2% bit mutation probability. The population size is $n_R = 200$ for rules and $n_{IC} = 200$ for ICs. The population of ICs is composed of binary strings of length $N = 149$. This population is fixed and it is initialized according to a uniform distribution over $\rho_0 \in [0.0, 1.0]$. The population of rules is also initialized according to a uniform distribution over $[0.0, 1.0]$ for the density. A natural definition for the fitness of rules is the number of ICs for which the CA relaxes to the correct state:

$$f(R_i) = \sum_{j=1..n_{IC}} covered(R_i, IC_j)$$

where:

$$covered(R_i, IC_j) = \begin{cases} 1 & \text{if the CA using rule } R_i \text{ and starting from initial} \\ & \quad \text{configuration } IC_j \text{ relaxes to the correct state,} \\ 0 & \text{otherwise} \end{cases}$$

At each generation, the top 95% reproduce to the next generation and the remaining 5% are the result of crossover between parents from the top 95% selected using a fitness proportionate rule. We chose a small generation gap (the percentage of new individuals in the next generation). This is not significant for the experiments in this section but it plays an important role in the next sections. Figure 4.3 describes the evolution over time of the distribution for rules density. A common behavior resulting from such an evolutionary setup is that the entire population focuses quickly to a small domain of the search space. Eventually, occasional progress may be observed over time. The variance for the final performance over several runs is usually important.

Several strategies have been proposed to improve the performance of evolutionary search. Usually, the central idea is to implement a mechanism maintaining diversity in the population in order to avoid premature convergence. Mahfoud [56] investigated thoroughly that domain of research and proposed several niching techniques based on that idea. Resource sharing, first introduced in [91], is a technique that has also been

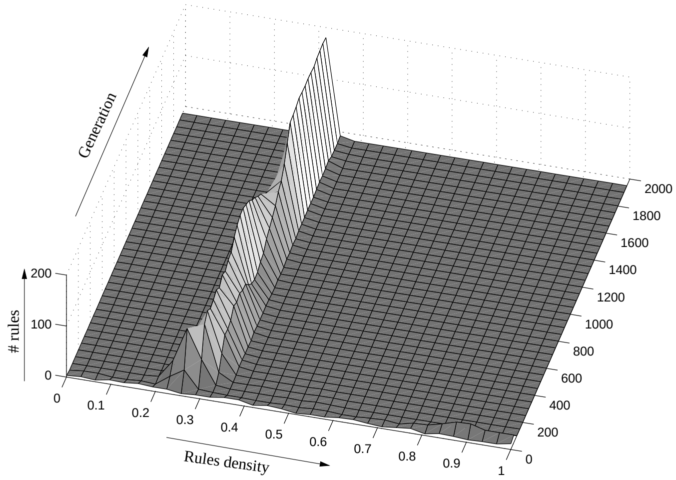


Figure 4.3: Evolution of CA rules in a fixed environment: the population converges quickly to a small domain of the search space. Occasional improvements may eventually be observed.

used successfully to maintain diversity [46]. Resource sharing can be used when individuals are evaluated against a number of training examples (or test cases) and a solution is sought which covers as many of those training examples as possible. Then, the basic idea consists in implementing a coverage-based heuristic by giving a higher payoff to test cases that few individuals can solve. One way to implement this technique for the evolution of CA rules is to define the fitness of rules as follows:

$$f(R_i) = \sum_{j=1..n_{IC}} weight\_IC_j \times covered(R_i, IC_j)$$

where:

$$weight\_IC_j = \frac{1}{\sum_{k=1..n_R} covered(R_k, IC_j)}$$

In this definition, the weight of an IC corresponds to the payoff it returns if a rule covers it. If few rules cover an IC, its weight will be much larger than if many rules cover that same IC. Figure 4.4 shows the evolution of the distribution for rules density over several generations. It can be observed that diversity is maintained for about 500 generations

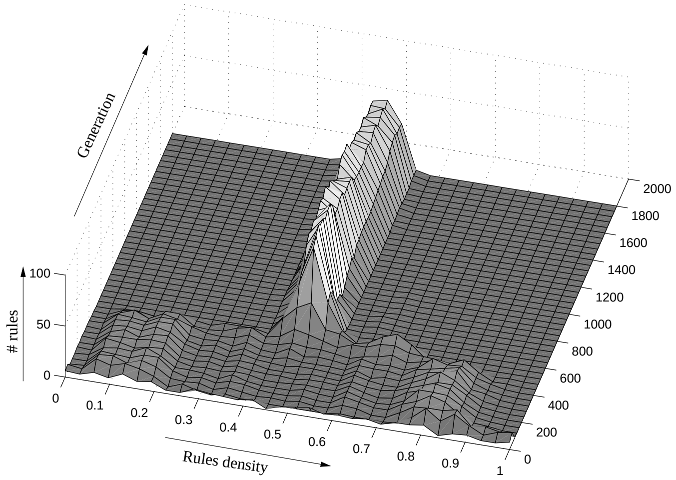


Figure 4.4: Evolution of CA rules in a fixed environment using resource sharing: multiple niches corresponding to the exploration of different alternatives are maintained in the population.

during which different alternatives are explored simultaneously by the population of rules. Then, a solution is discovered that moves the search in the area where density is close to 0.5. Afterward, several niches might still be explored, however the coding that has been exploited to construct the figures doesn't allow the representation of those different niches since all the rules have a similar density. Usually, this technique results in better performance on average. However, it takes also more time to converge. This is a trade-off between speed and quality of solutions.

4.2.4 Coevolutionary Search: Learning in an Adapting Environment

The idea of using coevolution in search was introduced by Hillis [38]. In his work, a population of parasites coevolves with a population of sorters. The goal of parasites is to exploit weaknesses of sorters by discovering input vectors that sorters cannot solve correctly while sorters evolve strategies to defeat parasites by sorting the corresponding input vectors. Eventually, as a result of this competition, some sorters are discovered that cannot be defeated (i.e., they are valid sorting networks). Several mechanisms were introduced by Hillis in his system (e.g. diploid genotype, elimination of redundant comparators, parasite/host relationship, geographically distributed population...), making difficult to identify the contribution of each of them to the performance of the search. Moreover, a strong bias was also introduced by initializing the population with the same prefix (first 32 comparators) as the best known construction. This reduces the size of the space of input vectors from $2^{16} = 65,536$ to 168 (out of which 17 are already sorted). Hillis' experiments resulted in a 16-input sorting network with 61 comparators, one more than the best known construction.

In the coevolutionary learning model under consideration in this analysis, there are several different possibilities to define the interactions between the two coevolving populations. Depending on those interactions, the underlying search heuristic implemented by the dynamics of coevolution is also going to be very different. Basically, two fundamental cases can be considered in such a framework, depending whether the two populations benefit from each other or whether they have different interests (i.e., if they are in conflict). Those two modes of interaction will be referred to as *cooperative* and *competitive* respectively.

In the following sections, those modes of interaction are illustrated experimentally in order to stress the different issues related to coevolutionary learning. The problem of the discovery of CA rules to implement the majority classification task is used again for that purpose.

Cooperation between Populations

In this mode of interaction, improvement on one side results in positive feedback on the other side. As a result, there is a reinforcement of the relationship between the two populations. From a search point of view, this can be seen as an *exploitative* strategy. Agents are not encouraged to explore new areas of the search space but only to perform local search in order to further improve the strength of the relationship. Following a natural extension of the evolutionary case to the cooperative model, the fitness of rules and ICs can be defined as follows for the $\rho_c = 1/2$ task:

$$f(R_i) = \sum_{j=1..n_{IC}} covered(R_i, IC_j)$$

$$f(IC_j) = \sum_{i=1..n_R} covered(R_i, IC_j)$$

For our experiments, the same setup as the one described in the previous section is used. The population size is $n_R = 200$ for rules and $n_{IC} = 200$ for ICs. The population of rules and ICs are initialized according to a uniform distribution over $[0.0, 1.0]$ for the density. At each generation, the top 95% of each population reproduces to the next generation and the remaining 5% is the result of crossover between parents from the top 95% selected using a fitness proportionate rule. This small generation gap has been chosen because of the relative definition of the fitness. Indeed, a large generation gap results in a dramatic change in the composition of the population. As a consequence, a lot of variations in individuals' fitness can occur because of the relative definition of the fitness. For instance, the best niche in one generation could be wiped out from the population in the following generation. This is an important issue in all implementations that use a definition for the fitness which depends on other members of the population. One technique that addresses this issue is *continuously updated sharing* [66] which updates the fitness of individuals during the reproduction process in order to maintain stable niches. The decision for this case study has been to select a small generation gap as a solution to this problem.

Figure 4.5 presents the evolution of the density of rules and ICs for one run using this cooperative model. Without any surprise, the population of rules and ICs quickly converge to a domain of the search space where ICs are easy for rules and rules consistently solve ICs. In that particular example, the population of rules converges to high density rules for which a CA usually relaxes to all 1's nearly independently of the initial configuration. This "niche" is then exploited by the other population which converges to high density ICs. This is a stable configuration for the two populations, resulting in little exploration of the search space.

Competition between Populations

In this mode of interaction, the two populations are in conflict. Improvement on one side results in negative feedback for the other population. For the $\rho_c = 1/2$ task, the fitness defined in the cooperative case can be modified as follows to implement the competitive model:

$$f(R_i) = \sum_{j=1..n_{IC}} covered(R_i, IC_j)$$
$$f(IC_j) = \sum_{i=1..n_R} \overline{covered(R_i, IC_j)}$$

where $\overline{covered(R_i, IC_j)}$ returns 1 if a CA using rule R_i and starting from IC_j fails to relax to the correct state. Otherwise, it returns 0. Here, the goal of the population of rules is to discover rules that defeat ICs in the other population by allowing the CA to relax to the correct state, while the goal of ICs is to defeat the population of rules by discovering initial configurations that are difficult to classify. For our experiments, the setup for reproduction/selection is the same as the previous example. Figure 4.6 describes an example of a run using this definition of the fitness. In a first stage, the two populations exhibit a cyclic behavior. Rules with low density have an advantage because

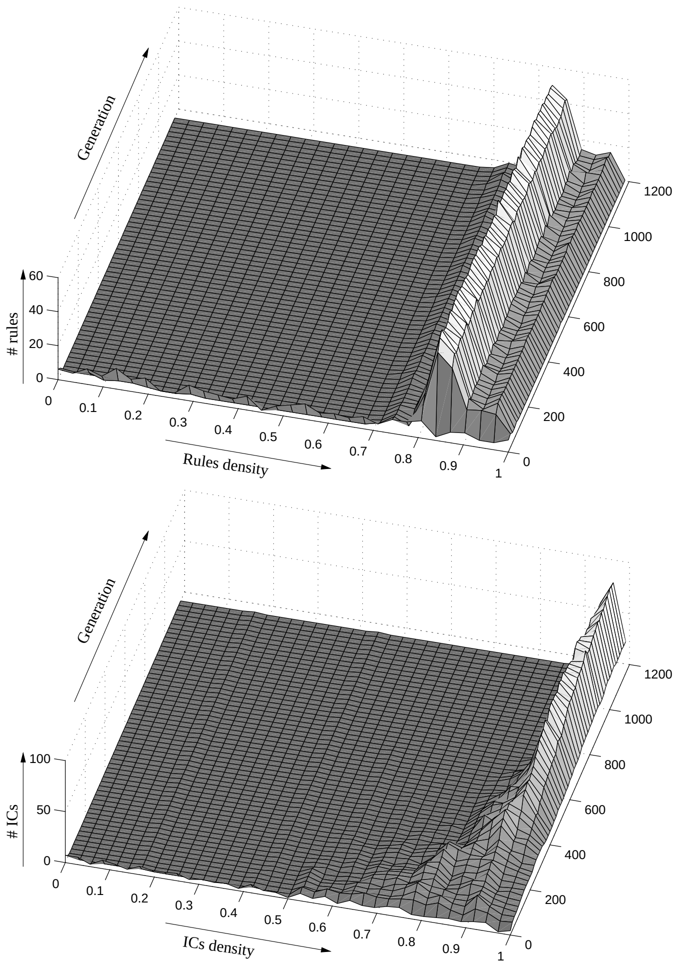


Figure 4.5: Coevolution of CA rules (top) and ICs (bottom) in a cooperative relationship: the strong incentive for each population to propose easy problems to the other results in little exploration of the search space.

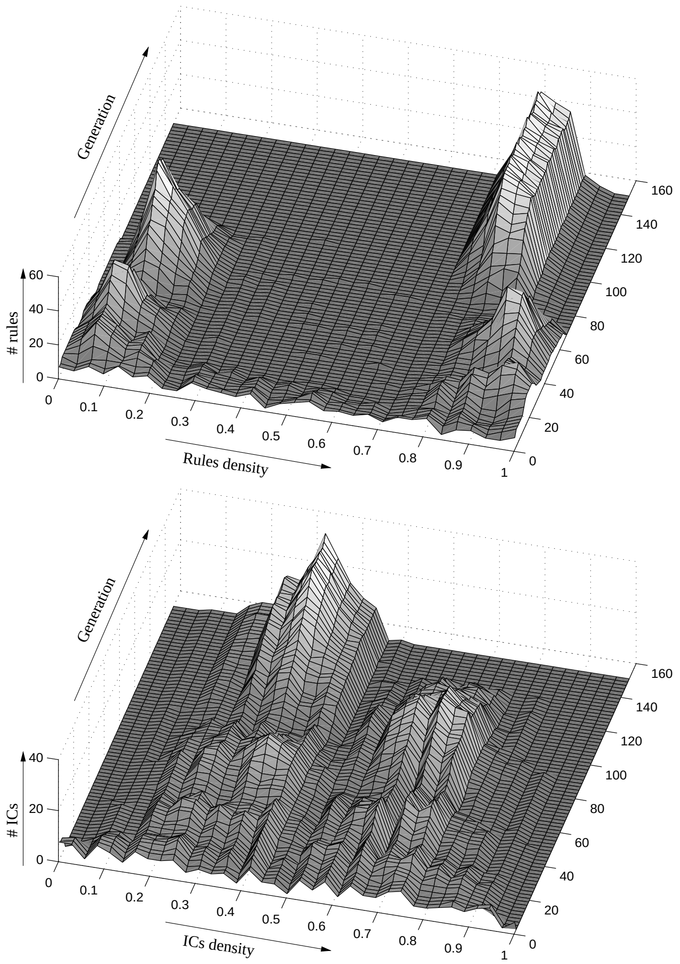


Figure 4.6: Coevolution of CA rules (top) and ICs (bottom) in a competitive relationship: the two populations follow conflicting goals, resulting in an unstable behavior.

there are more ICs with low density. In response to this, ICs with high density have an evolutionary advantage and their number increases in the population. In turn, rules with high density get an evolutionary advantage... This unstable behavior exhibited by the two populations is an instance of the *Red Queen* effect [18]: fitness landscapes are changing dynamically as agents from each population adapt in response to the evolution of members of the other population. The performance of individuals is evaluated in a changing environment, making continuous progress difficult. A typical consequence is that agents have to learn again what they already knew in the past. In the context of evolutionary search, this means that domains of the state space that have already been explored in previous generations are searched again. Then, a stable state is reached: in this case, the population of rules adapts faster than the population of ICs, resulting in a population focusing only on rules with high density and eliminating all instances of low density rules (a finite population is considered). Then, low density ICs exploit those rules and overcome the entire population. The population of rules can't respond since all low density rules have been wiped out. Therefore, the population of rules gets no feedback and no gradient is available to drive the search since their fitness is always 0. All ICs get maximum fitness of 1. The two populations will stay in this stable state unless a "lucky" rule is discovered that can defeat some ICs.

Resource Sharing and Mediocre Stable States

Resource sharing has been presented earlier as a strategy which allows the simultaneous exploration of different alternatives by maintaining several niches in the population. This section describes some experiments where resource sharing is introduced in the competitive model of interaction presented in the previous section. The fitness of rules and ICs is then defined as follows:

$$f(R_i) = \sum_{j=1..n_{IC}} weight\_IC_j \times covered(R_i, IC_j)$$

where:

$$weight\_IC_j = \frac{1}{\sum_{k=1..n_R} covered(R_k, IC_j)}$$

and

$$f(IC_j) = \sum_{i=1..n_R} weight\_R_i \times \overline{covered(R_i, IC_j)}$$

where:

$$weight\_R_i = \frac{1}{\sum_{k=1..n_{IC}} \overline{covered(R_i, IC_k)}}$$

This framework allows the presence of multiple niches in both populations, each niche corresponding to a particular competence relevant to defeat some members of the other population. Figure 4.7 describes one run for this definition of the interactions between the two populations. It can be seen that the unstable behavior which was observed in the previous section doesn't occur any more and that two species coexist in the population of rules: a species for low density rules and another one for high density rules. Those two species drive the evolution of ICs toward the domain of initial configurations that are the most difficult to classify (i.e., $\rho_0 = 1/2$). However, the two populations have entered a *mediocre stable state*. That is, a stable configuration is reached that involves multiple average performance niches which coexist in both populations. There are two possible reasons for the existence of a stable state:

- any slight alteration of an individual in one niche results in no improvement or a smaller performance.
- the gap an individual in one niche must bridge to solve problems already solved by other niches is too large. This comes from the representation scheme or the search operators: the sequence of transformations (with respect to the search operators)

	Initial Configurations						
	1	1	1	1	0	0	0
	1	1	1	1	0	0	0
	1	1	1	1	0	0	0
Rules	0	0	0	0	1	1	1
	0	0	0	0	1	1	1
	0	0	0	0	1	1	1
	0	0	0	0	1	1	1

Table 4.2: An interaction matrix representing a stable state.

that should be followed by an individual to improve its performance is too long or has a very small probability of occurring. As a result, individuals receive no feedback from problems they cannot solve (since they are consistently defeated by those problems) to drive search.

In this example, the matrix of interactions between the two populations which describes the outcome for each pair $(rule, IC)$ looks like the one in Table 4.2. ICs are concentrated around the $\rho_0 = 1/2$ threshold and they can be divided into two groups: those with density $\rho_0 < 1/2$ and those with density $\rho_0 > 1/2$. This distribution means that ICs can be exploited consistently by rules with low and high density which are both present in the second population (because a CA implementing a low density rule relaxes mostly to all 0's for any IC and mostly to all 1's when implementing a high density rule). However, this is a mediocre stable state in the sense that coexisting species are too specialized to defeat only a subset of the possible opponents and the likelihood for improving their performance is small.

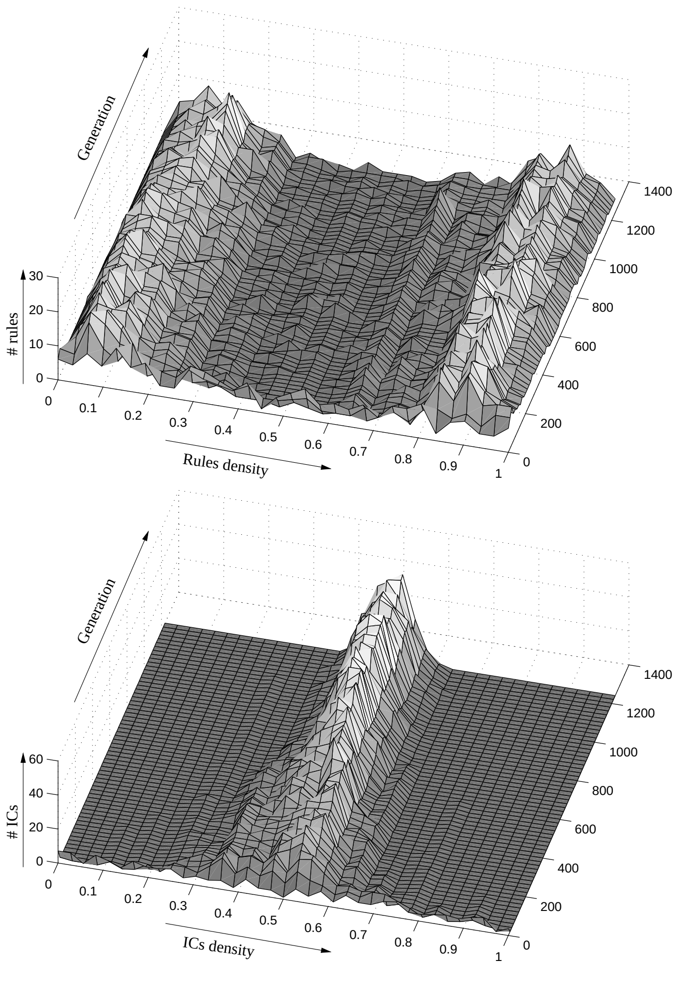


Figure 4.7: Coevolution of CA rules (top) and ICs (bottom) in a competitive relationship, using resource sharing in both populations: the two populations converge to a mediocre stable state involving a number of sub-optimal niches.

4.3 Background on Coevolutionary Search

With the emergence of the field of evolutionary computation, researchers have taken inspiration from nature for the design of new search strategies. Along those lines, coevolution has been the object of a lot of attention. Indeed, coevolution is inherent in Nature where any living system's reproductive success is based on how it performs with respect to others for exploiting finite resources, finding mates or socializing ... As a problem solving paradigm, coevolution has been proposed as elegant approach to problems that involve multiple criteria or can be decomposed into sub-tasks. Axelrod's work [9] on the Prisoners' Dilemma was certainly among the first applications of coevolution. His aim was the study of the emergence of cooperation in evolution when competition seems to be the driving force toward progress. The idea of using coevolution for problem solving was introduced by Hillis [38]. Since then, several models have been developed by researchers using a similar methodology. The next sections review some of this work. Following the two modes of interaction discussed previously, this review is divided into two parts. The first part addresses competition-based models while the other is dedicated to cooperation-based models.

4.3.1 Competition between Populations

Competitive coevolution involves at least two species or populations, each of them seeking to defeat members of the other one(s). This framework has been used to address a variety of problems, each approach proposing some specific techniques to address the problems inherent to this model, like the Red Queen effect. Two models of competitive coevolution have been widely exploited in the EC community: *predator/prey* and *parasite/host*. The difference between those two models is subtle and, so far, it has not been relevant from an artificial evolution point of view. According to Roughgarden [94], parasite/host is a form of symbiosis in the sense that there is a long-term physical relationship between the parasite and the host. On the contrary, in the predator/prey model, there is no such long-term physical relationship.

Hillis' work marked a significant breakthrough by showing that coevolution may improve search performance [38]. In his work, a population of sorters (the hosts) coevolve with a population of input vectors (the parasites). The goal of sorters is to construct sequences of comparator-swaps that sort the input vectors that are proposed by the parasites while parasites search for input vectors that are difficult to sort. In a sense, this can be seen as an implementation of a coverage-based heuristic: a construction is sought that sorts correctly every possible input vector, thus resulting in a sorting network. This heuristic adaptively focuses the search for solving problem instances (i.e., input vectors) that are the most difficult for the population of networks. Several mechanisms help to prevent indirectly the unstable behavior associated with the Red Queen effect. For instance, parasites represent a set of input vectors instead of a single input vector. This reduces the chance that a single parasite implements only difficult input vectors with respect to the current population of hosts. Also, the distributed population introduces some robustness in the search by maintaining different alternatives in the two populations. The use of diploid genomes, another feature implemented by Hillis, allows the introduction of a pressure toward shorter sorting networks because individuals that contain more redundancy (i.e., that have more homozygous pairs) are more robust to crossover. Because of the representation scheme, such individuals represent shorter networks.

Following Hillis' approach, Husbands used a similar architecture and implemented a geographically distributed model of coevolution to address a generalized version of the job-shop scheduling problem [40].

Paredis [73] used competitive coevolution between a population of solutions and a population of problems as a search strategy for applications in inductive learning [71] and constraint satisfaction problems [70]. In both applications, a set of test cases is identified and a concept or solution that covers accurately this test set is searched. To encourage continuous progress, Paredis implemented a steady-state model and introduced life-time fitness evaluation (LFTE), a mechanism that averages the performance of individuals over a sliding window covering several generations in order to evaluate individuals against a significant number of problems. This mechanism provides candidate solutions with a gradient involving the recent history, thereby helping search.

Pursuer/evader games have also been used as a test problem for research in coevolution. In particular, Cliff and Miller [18, 19] illustrate the difficulty of designing experiments in which non trivial behaviors (i.e., evasion and preying strategies) do emerge. Detecting when such behaviors occur and analyzing the outcome of coevolutionary simulations in the context of the pursuer/evasion model is also a difficult task. In their work, several tools were developed to track progress and detect loss of traits resulting from the Red Queen effect. Sims' block-creatures [99] and Reynolds' experiments with the game of tag [86] are also two successful applications of competitive evolution. In all those works, the emergence of complex behaviors was observed. However, specific selection and evaluation strategies were chosen in order to maintain a variety of different strategies in the population and to allow significant progress for the performance of evolved agents over time.

Rosin's work on coevolutionary learning [93] addresses the different issues related to competitive evolution in the context of adversarial problems (e.g., game strategies). The goal of his work is to define a framework for coevolutionary search that results in continuous progress on the long term. In a theoretical analysis [92], Rosin and Belew described a coevolutionary environment and proved it allows the discovery of perfect game strategies, provided a *"strategy-learning algorithm"* (i.e., an algorithm able to learn strategies that defeat a given set of opponents) is available. Relying on the existence of such a learning algorithm, a *covering competitive algorithm* is designed which results in polynomial-time learnability in the logarithm of the size of the first-player and second-player strategy space, and the *specification number*. The specification number is defined as the size of the smallest *teaching set* for the first-player. A first-player teaching set is defined as a set of second-player strategies such that for any imperfect first-player strategy, there is a strategy in the teaching set that defeats it. The idea of this covering competitive algorithm is to call alternatively for the first-player and for the second-player the strategy-learning algorithm with the set of all previously seen opponent strategies. This algorithm returns a strategy that can defeat all those opponent strategies, therefore bootstrapping coevolutionary learning. However, those requirements are difficult to implement in practice. In particular, the existence of the strategy-learning algorithm is a strong assumption. In [91, 93], Rosin introduced several heuristics to overcome the flaws inherent in coevolutionary search and to implement some of the concepts introduced in his theoretical analysis. In the framework he proposed, the strategy-learning algorithm is introduced implicitly in the design of the credit assignment procedure. *Competitive fitness sharing* is introduced in order to maintain different strategies in the population game strategies. A *Hall of fame* keeps track of the best individuals from previous generations and encourages long-term progress by also evaluating new individuals against a sample of this hall of fame. A *phantom parasite* heuristic prevents a solution that is perfect with respect to the current population of adversaries to take over the entire population (which would result in a loss of diversity).

More recently, Pollack, Blair and Land [80] used a simple coevolutionary approach to the game of backgammon with significant success. One goal of this work was to prove that the noisy evaluation of players resulting from the dice roll plays an important role in the search of game strategies. In particular, a simple coevolutionary setup involving the current best strategy and a challenger constructed by mutating this top strategy was enough for the discovery of strong players.

4.3.2 Cooperation between Populations

Scalability is an important issue in Artificial Intelligence. However, the design of large scale complex structures that are able to perform a given task in an efficient and coordinated manner is difficult. The traditional methodology in engineering is to design a top-down decomposition, defining at each level the interactions and controls between the different modules. Usually, such a design includes a central control system that coordinates the activity of the different modules.

The coevolutionary paradigm has also been proposed as a potential solution to this problem. For instance, Potter [81] developed a cooperative model in which a number of populations explore different decompositions of the problem. The central idea of this approach is to have each population specializing on a particular functionality that is relevant to address the problem and, at the same time, the functionalies discovered by the different populations can be recombined into a composite solution that solves the task. In Potter's work, a predefined strategy determines how a composite solution is constructed from the modules implemented by those populations. Then, a credit assignment strategy focuses the search on individuals in each population that participate in better composite solutions. Eventually, if a good decomposition is discovered, the computational effort to construct a solution to the problem is greatly reduced. Indeed, an appropriate decomposition of the problem enables incremental improvement because orthogonal dimensions for search have been identified: each module can be searched independently for local improvement of the composite solution. On the contrary, the search for a monolithic solution that would address the entire problem can be much more expensive because of *epistasis*: without the notion of modularity, the different elements of the representation are strongly interdependent, making the search more difficult because of a large number of local optima. Potter, DeJong and Grefenstette exploited this approach to cooperative coevolution for function optimization [82] and for the design of control systems [83].

Moriarty also used a coevolutionary approach as a search paradigm to construct modular structures [65]. In the system he developed, named SANE, each "module" corresponds to one hidden unit of a neural network. Entities undergoing evolution encode the connections and the weights between an hidden unit and neurons in the input and the output layer. At a higher level of abstraction, a population of network specifications represents composite solutions. Each network specification identifies a set of hidden units that contribute to the composite solution. Search is performed in the population of hidden units for neurons that participate in high quality networks. In the other population, the space of network specifications is searched for good "teams" of hidden units that result in high performance composite solutions. SANE has been applied successfully to several sequential decision tasks like the pole balancing problem [62], game playing [61] or robot arm control [63].

Another approach to cooperative coevolution is the one exploited by Paredis [72]. In his work, a population of solutions and a population of permutations performed on the genotype of the first population coevolve. The motivation underlying this work is to limit the disruptive effect of search operators by discovering an appropriate representation (that is, a representation that is more robust and makes progress more likely). Paredis performed his experiments in a GA framework using one-point crossover. Solutions for which functionally related genes are grouped together on the genotype are more robust to this operator. The goal of the population of permutations is to discover such arrangements for genes. Success was reported with a setup involving multiple copies of the same problem. As a result of the cooperative interactions, a permutation would emerge that group together the bits corresponding to the same instance of a problem, allowing a more efficient exploration of the solution space for each problem with the mutation operator.

As discussed in section 4.2.4, the drawback of cooperative coevolution from a search point of view is that the positive reinforcement between the members of each population usually results in fast convergence to an average performance solution because there is little pressure toward exploration. Different mechanisms have been introduced to address this issue by maintaining diversity. For instance, Potter introduces some randomness in the selection of the components for the construction of composite solutions instead of selecting systematically the best component from each population. In the case of Moriarty's work, multiple composite solutions are explored in parallel by the population of network specifications instead of focusing on a single modular decomposition. In a way, this approach combines both bottom-up and top-down search. In the case of Paredis' work, the noisy evaluation of individuals' fitness also prevents premature convergence.

4.4 Discussion

This chapter describes our motivation for exploiting coevolution as a paradigm to capture a specific property of evolving agents defined as *adaptability*. Adaptability corresponds to a dynamic property exhibited by individuals. Therefore, evaluating those individuals in a changing environment, as it is the case in coevolution, may be a potential strategy to capture that property.

However, depending on the rules that control the interactions between the members of the coevolving populations, a lot of diversity may be observed for the resulting dynamics of the system. That is, the trajectory followed by each population to explore its respective search space may correspond to extremely different strategies. Those issues have been illustrated in the previous sections with the problem of discovering cellular automata rules that implement the majority classification task. The fundamental models of cooperation and competition have been analyzed in this framework in order to interpret the resulting dynamics of coevolution from an evolutionary search point of view and to identify the underlying search heuristics.

In summary, it appears that none of the canonical models described in this chapter can achieve alone the goal of coevolutionary learning, that is: continuous progress on the long term. The reason is that the underlying search heuristics implemented in those models are too simplistic and introduce a pressure only in the direction of exploration or exploitation. In the next chapter, the concept of the "Ideal" trainer is described. This paradigm exploits a coevolutionary framework in which the underlying heuristics introduce a pressure toward *adaptability* while driving search to allow continuous improvement. This result is achieved by maintaining an appropriate balance between exploration and exploitation among the two coevolving populations, and by introducing a meta-level strategy which controls the direction of evolution. This paradigm is illustrated with two applications presented in chapter 6.

Chapter 5

Coevolving the "Ideal" Trainer: a Paradigm for Achieving Continuous Progress

From the perspective of this research work, continuous progress corresponds to the identification of domains of the state space which correlate better with the operators embedded in a search algorithm. This approach is proposed as an alternative to other search paradigms when little knowledge is available about the regularities of a problem domain. However, as illustrated in chapter 4, implementing a coevolutionary framework that exhibits continuous progress is a non-trivial task. Indeed, the dynamics of coevolution is an emergent phenomenon which results from the definition of rules controlling the interactions between coevolving agents. Therefore, the designer has little control over the global behavior of the system. In extreme cases, a slight change in the definition of those rules may even result in dramatic variations for the dynamics of the coevolving

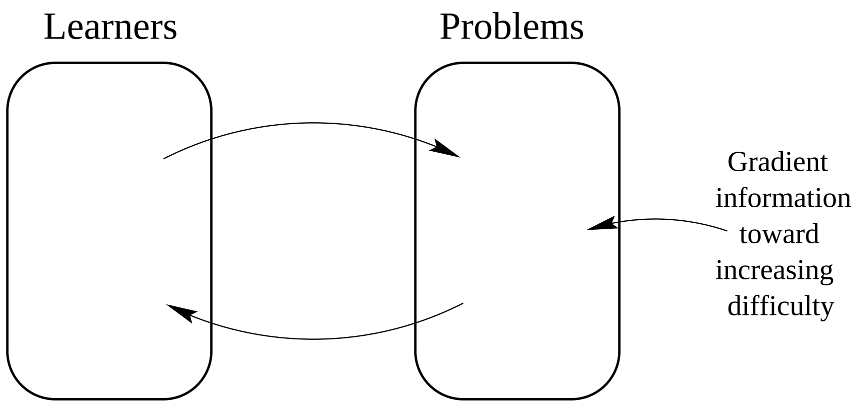


Figure 5.1: Introduction of gradient information in the "Ideal" trainer approach to coevolutionary learning.

populations.

One observation that may illustrate the complexity of achieving continuous progress is that, even in Nature, evolution doesn't seem to pursue any other goal than to favor the emergence of species that have a better ability to survive and reproduce in a particular evolving and/or adapting environment. While continuous progress in Nature could be defined as an improvement of living systems ability to survive when compared to anterior generations, the measure of survival ability is a function of the environment in which those organisms are evaluated. Since this environment is in perpetual change, the notion of continuous progress doesn't necessarily make sense in that context: continuous progress can be measured only against a fixed reference. On the contrary, problem solving is a goal directed process. That is, a task has been identified and a solution to that particular task is sought. Therefore, some mechanisms must be implemented in order to drive coevolutionary search toward solutions that can achieve the goal.

The next section reviews the most important obstacles that prevent coevolution from achieving continuous progress in the context of problem solving. In response to those issues, the central contribution of this chapter consists in the statement of two specific conditions whose implementation is necessary for allowing the emergence of continuously improving solutions: 1) the need to maintain useful feedback from the training environment, and 2) the need for a meta-level strategy to ensure progress in the long term. This second requirement is represented in Figure 5.1 with an external control which introduces a gradient for the discovery of problems of increasing difficulty. Finally, a new paradigm, referred to as the "Ideal" trainer, is proposed. This paradigm is based on a specific framework which allows the implementation of those two requirements. The next chapter illustrates this paradigm with two applications. The first one is a follow up of the majority classification task introduced in chapter 4. The second application exploits the "Ideal" trainer paradigm to propose a modular approach to inductive learning.

5.1 Coevolutionary Learning: Learned Lessons

5.1.1 Adaptability is a Relative Measure

Coevolution has been presented in chapter 4 as a paradigm to capture adaptability. That is, coevolution introduces a pressure such that individuals that adapt faster to the changing environment get an evolutionary advantage. From the perspective of search, this means that fewer transformations of those individuals by the search operators are necessary to improve their performance. Or, from a learning point of view, this means that agents have captured some intrinsic properties underlying the evolution of the training environment that allow them to react to those changes.

This description also means that adaptability is defined with respect to the way the training environment evolves. Therefore, the underlying search strategy implemented by the dynamics of coevolution depends on the changes that are observed in the training environment. However, the goal of coevolutionary learning is to observe continuous progress. This means that the goal is to capture the *right* type of adaptability: the ability to solve problems of increasing difficulty. Capturing adaptability for its own sake is not a sufficient condition to observe continuous progress because adaptability is relative to the dynamics of evolution of the training environment.

5.1.2 Continuous Progress is an Absolute Measure

On the other hand, continuous progress is a global property: it is defined with respect to some absolute measure of performance. Therefore, this information must be introduced one way or another in the coevolutionary framework if the goal of coevolutionary learning is to be achieved.

In general, such an absolute measure for the evaluation of individuals' performance is not necessarily available. In fact, in Nature, it is not even sure whether such an absolute measure exists (e.g., what absolute measure could compare the performance of a mammal to the performance of an ameba? Each of those two species is specialized to exploit some very specific resources.) However, for problem solving, an absolute measure of performance usually exists. In some cases, it might however be intractable to evaluate this measure because it would involve too much computer resources. For instance, the absolute performance of a game strategy would be evaluated by playing against a potentially infinite set of opponent strategies. In such instances, some other methods must be employed. For the applications presented in this dissertation, the absolute performance of individuals can either be evaluated exactly or estimated with reasonable accuracy.

5.2 Coevolutionary Learning: Conditions for Success

The purpose of the next sections is to identify two important requirements for achieving continuous progress in the framework of coevolutionary learning.

5.2.1 Need for Maintaining Useful Feedback

Adaptability is the primal component underlying the ability for an agent to exhibit continuous progress. However, capturing adaptability is possible only if the population of evolving agents is exposed to a gradient. That is, some information is available allowing differentiation between the members of the population.

In order to illustrate this notion, consider a simple multi-objective problem involving a set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of training examples. The goal is to isolate a candidate solution that covers all the test cases T_i's. Now, consider a coevolutionary framework involving a population of candidate solutions and a training environment composed of a subset of $\mathcal{T}$ (this subset is changing over time in response to the evolution of candidate solutions). Let $\Psi_{(t)}$ be the subset of candidate solutions that cover all the problems proposed by the training environment at time t ($\Psi_{(t)}$ will also be called the solution set at time t). Here, the attention is focused on the evolution of the successive solution sets $\Psi_{(t)}$'s. The rules that control the interactions between the population of candidate solutions and the training environment are not directly relevant to the discussion.

Figure 5.2 illustrates an instance where individuals in the solution set $\Psi_{(t)}$ are exposed all of a sudden to a training environment which is much more difficult (corresponding to a tiny solution set $\Psi_{(t+1)}$). In that case, the likelihood that one of those agents is already a member of $\Psi_{(t+1)}$ or that transformations operated to those agents result in the discovery of a solution in $\Psi_{(t+1)}$ is very little. No gradient information is available to the population of agents to drive the search since all the individuals have same performance.

Such dynamics is typical of mediocre stable states which were introduced in section 4.2.4. Mediocre stable states correspond to a similar situation involving multiple "niches" and for which no useful feedback is returned to the evolving agents. In that

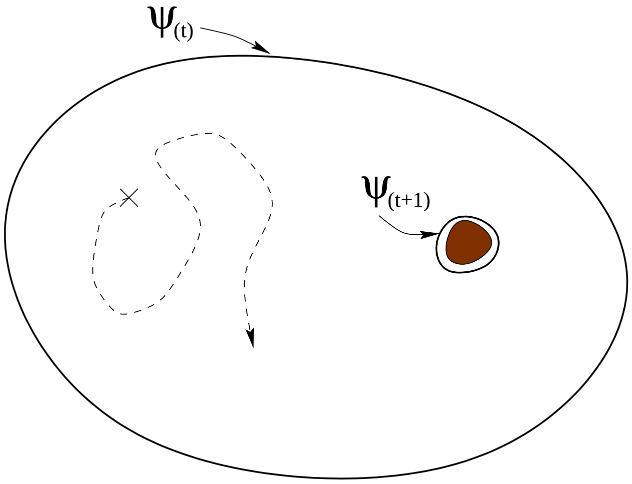


Figure 5.2: If the difficulty of problems proposed by the training environment increases too quickly, there is a small probability that an individual in the solution set $\Psi_{(t)}$ be mapped to the solution set $\Psi_{(t+1)}$ by the search operators: no gradient is available to drive search toward the target solution set (represented by the dark area).

case, all the agents in a niche cover the same subset of problems proposed by the training environment and fail to cover the others.

Therefore, the evolution of the training environment should always result in little variations for the successive solution sets $\Psi_{(t)}$'s in order to always expose the population of evolving agents to a gradient. This is a requirement for capturing adaptability.

5.2.2 Need for a Meta-Level Strategy

As discussed before, capturing adaptability for its own sake is not a sufficient condition to achieve continuous progress. This can be illustrated with the diagram in Figure 5.3. In that example, the solution sets evolve progressively and provide a gradient for search. This dynamics facilitates the emergence of adaptability. However, even if the target solutions (represented by the dark area in Figure 5.3) are members of the different solution sets, the evolution of the $\Psi_{(t)}$'s doesn't introduce any pressure to drive the search toward this target. This dynamics is typical of the Red Queen effect: while evolving agents adapt to the changing environment, there is no high-level strategy to drive search toward solutions with increasing performance with respect to the global task.

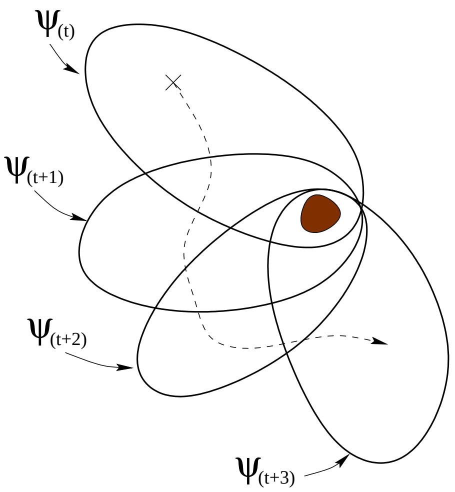


Figure 5.3: The pressure toward adaptability is not enough to drive search toward the target solution set (represented by the dark area): a high-level strategy is necessary to control the evolution of the solution sets $\Psi_{(t)}$, $\Psi_{(t+1)}$, $\Psi_{(t+2)}$ ...toward that target.

Indeed, continuous progress is defined with respect to some absolute measure. Therefore, if adaptability is to result in continuous progress (and the emergence of high quality solutions), the training environment must propose problems of increasing difficulty. This can be achieved only if this absolute notion of difficulty is introduced in the system in the form of a meta-level strategy. The purpose of the meta-level strategy is to prevent cyclic behaviors by providing a direction for the evolution of the training environment.

In summary, two requirements should be satisfied simultaneously in order to achieve continuous progress:

1. the training environment should facilitate adaptability by always exposing the population of evolving agents to a gradient, and

2. problems of increasing difficulty should be proposed by the training environment.

Figure 5.4 illustrates the dynamics for the evolution of the solution sets resulting from the implementation of those two conditions. The sets $\Psi_{(t)}$, $\Psi_{(t+1)}$, $\Psi_{(t+2)}$ ... converge progressively toward the target solutions. At each step in this evolution, the gap between two consecutive solution sets can be bridged by the evolving agents.

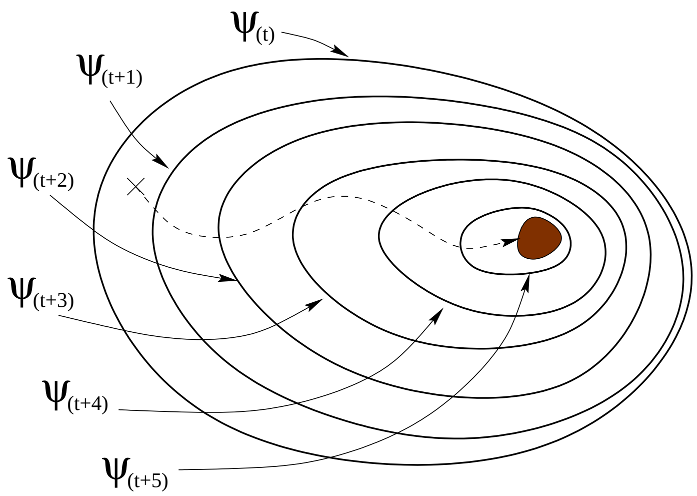


Figure 5.4: Continuous progress is possible by allowing the progressive evolution of the solution sets $\Psi_{(t)}$, $\Psi_{(t+1)}$, $\Psi_{(t+2)}$ ... toward the target solution set (represented by the dark area).

The next section introduces the concept of the "Ideal" trainer. This paradigm proposes a framework to implement those requirements. This framework defines the rules of interactions between the coevolving populations such that continuous progress emerge from the resulting dynamics.

5.3 Coevolving the "Ideal" Trainer: Presentation

From the analysis of the experiments presented in section 4.2 and from the discussion in the previous sections of this chapter, at least two reasons seem to prevent continuous progress in coevolutionary search. The first one, identified as the occurrence of mediocre stable states, is that the training environment provided by the population of problems returns little information to the population of learning agents because a stable configuration is reached in which the credit is distributed according to a fixed pattern. The second reason is that the dynamics of the search performed by the two coevolving populations doesn't drive individuals toward domains of the state space that contain most promising solutions because there is no "high-level" strategy to play that role. This is a consequence of the Red Queen effect which usually results in the cyclic behavior observed in the competitive model of coevolution.

The idea of the "Ideal" trainer is based on the introduction of explicit mechanisms to implement the two requirements identified in section 5.2. The purpose of those mechanisms is to control the evolution of the population of problems. In fact, the central strategy underlying the concept of the "Ideal" trainer can be described with the following statement:

> *"The best way for adaptive agents to learn is to be exposed to problems that are just a little more difficult than those they already know how to solve."*

This strategy covers exactly the two fundamental requirements discussed previously: the need to maintain useful feedback (by exposing agents to problems "a little more difficult" than those they know how to solve) and the need for a meta-level strategy (to propose problems of increasing difficulty). The implementation of this strategy requires the definition of the following terms:

- a *distance measure* to formalize the concept of "a little more difficult", and

- a *partial order* over the space of problems in order to control the evolution of the training environment toward problems of increasing difficulty.

As a result, this search procedure always maintains a gradient to drive the exploration of the space of solutions (i.e., the learning agents). That is, the training environment proposes a variety of problems covering a range of difficulty without exposing the learners to problems that are too difficult or too easy. Indeed, if problems are too difficult, none of the learning agents can solve them. On the contrary, if they are too easy, all the agents can solve them. In both cases, those problems are useless for learning since they provide little feedback. Also, a "high-level" strategy is implemented in order to allow continuous progress by proposing problems of increasing difficulty, thereby preventing some of the negative effects associated with the Red Queen. By maintaining this constant pressure toward slightly more difficult problems, a race is induced among learners such that learners that adapt better have an evolutionary advantage. The underlying heuristic implemented by this evolutionary race is that *adaptability* for solving increasingly difficult problems is the driving force.

From the perspective of learning, the ability of individuals to adapt faster to a changing environment means that those individuals have captured some of the intrinsic mechanisms that control the evolution of the training environment. In the context of the "Ideal" trainer, a meta-level strategy controls the evolution of the training environment toward problems of increasing difficulty. Therefore, the pressure toward adaptability implemented in this framework may eventually result in the emergence of individuals that are able to generalize their performance to unseen problems. That is, they are likely to have a better performance when faced with even more difficult problems.

When applying the "Ideal" trainer concept to a specific task, multiple difficulties must be overcome in order to implement accurately the different concepts introduced in this section. So far, our methodology has consisted in constructing an explicit structure over the space of problems by defining a partial order with respect to the relative difficulty of problems among each other. In our current work, the concept of "relative difficulty" has been defined by exploiting some *a priori* knowledge about the task. The definition of this topology over the space of problems makes possible the implementation of the two goals required in our coevolutionary learning approach. Indeed, since learners are evaluated against a known range of difficulty for problems, it is possible to monitor their progress and to expose them to problems that are just "a little more difficult". In this framework, learners are always exposed to a gradient for search and it is possible to control the evolution of the training environment toward problems of increasing difficulty in order to ensure continuous progress.

In the future, our goal is to eliminate some of those explicit components by introducing some heuristics that automatically identify problems that are appropriate for the current set of learners. The work of Rosin [93] already describes some methods to address this issue.

5.4 Related Work

The idea of introducing a pressure toward adaptability as the central heuristic for search is not new. Schmidhuber [96, 97] proposed the Incremental Self-Improvement system in which the ability to exhibit continuous progress is the measure that is optimized. In that work, the fitness is an explicit measure of progress over time: a history of the transformations that have been performed on the current solution is stored and is used to monitor the evolution of the performance of that solution. When progress stops, the search procedure backtracks (i.e., undoes the transformations that have been operated) and explores new transformations. However, the basic process to perform search in that work differs in many ways from the approach described in this dissertation.

In particular, Schmidhuber's work takes root in traditional tree search algorithms and may not exploit some of the advantages associated with evolutionary-inspired search algorithms like robustness and an efficient implementation on distributed architectures.

The concept of an ideal trainer has also been introduced by Epstein [25] in the context of game learning. However, this work addresses the issue of designing the "Ideal" training procedure which would result in high quality players rather than coevolving the training environment in response to the progress of learners. Epstein exhibited some evidence that a combination of training against a perfect player and self-playing is a better learning procedure than pure self-training or playing only against a perfect player. The idea underlying this result is that the perfect player gives information about the perfect play skeleton in the search tree while self-playing allows the exploration of variations of that perfect framework in order to increase robustness of the strategy.

5.5 Concluding Remarks

In this chapter, the concept of the "Ideal" trainer has been introduced in the context of coevolutionary learning. The underlying idea implemented in this strategy is to always propose some problems that challenge the population of learners without exposing them to problems that are too difficult. In the literature, different approaches have been proposed to address the issues associated with the Red Queen effect [73, 93]. However, to our knowledge, explicit methods to force continuous progress and to prevent mediocre stable states in the context of evolutionary search have never been tested. In the framework described in this chapter, some explicit mechanisms have to be introduced in order to implement this strategy. One goal for the future is to replace those explicit mechanisms with some general purpose heuristics that would construct, for instance, a partial order over problems as they are generated or that would adaptively maintain the balance between cooperation and competition without requiring the definition of a distance measure.

As a consequence of the strategy implemented in the “Ideal” trainer coevolutionary framework, an evolutionary race is maintained among the learners such that learners adapting faster to the new challenges get an evolutionary advantage. As a result of this pressure toward adaptability, learners that have a higher generalization ability are likely to emerge from this race. From a machine learning point of view, the issue of generalization is particularly important. Traditional techniques for achieving good generalization consist in introducing some explicit mechanisms in order to drive the search toward the discovery of compact models. For instance, this approach has been used to construct neural networks with fewer hidden nodes (e.g., for applications in time-series prediction [110]) or, in genetic programming, to construct S-expressions with a minimum number of primitives [42, 116]. This idea of model compactness in learning, also known as Occam’s razor, has always been seen as a requirement for generalization and is at the origin of techniques implementing the Minimum Description Length (MDL) principle [87, 88] or the Minimum Message Length (MML) principle [107, 108]. However, there is no such explicit pressure in the “Ideal” trainer framework. Instead, the pressure toward better generalization occurs as a side-effect of the evolutionary race. This feature is particularly interesting because the definition of mechanisms that implement the MDL or the MML principle is dependent on the representation language.

The next chapter presents two applications of the “Ideal” trainer concept. The first one is a follow up of the experiments with the cellular automata problem introduced in chapter 4. This application resulted in the discovery of new CA rules which improve significantly over previously best known rules for the majority classification task.

The second application exploits the “Ideal” trainer concept to implement a bottom-up approach to inductive learning. A system, named Modular Inductive Learning (MIL), has been developed which involves a population of local models (the learners) that coevolve with a population of domains defined over the input space (the problems). By maintaining an appropriate balance between accuracy of local models and size of domains, the dynamics of the evolutionary race implemented in MIL allows the emergence of classification theories with good generalization ability.

Chapter 6

Applications of the "Ideal" Trainer Paradigm

6.1 Application 1: Discovery of CA Rules for the Majority Classification Task

6.1.1 Presentation

This section describes the application of the "Ideal" trainer paradigm described in section 5.3 to the majority classification problem for CAs (i.e. the $\rho_c = 1/2$ task). This application is a natural follow up of the experiments presented in section 4.2. Indeed, this experimental analysis has provided us with insights about the different issues involved in coevolutionary learning. Therefore, studying the dynamics of search performed in the framework of the "Ideal" trainer for that same problem will provide us with useful information to perform a comparative analysis. Moreover, this problem offers an interesting challenge allowing us to compare our approach to the results that have been achieved by

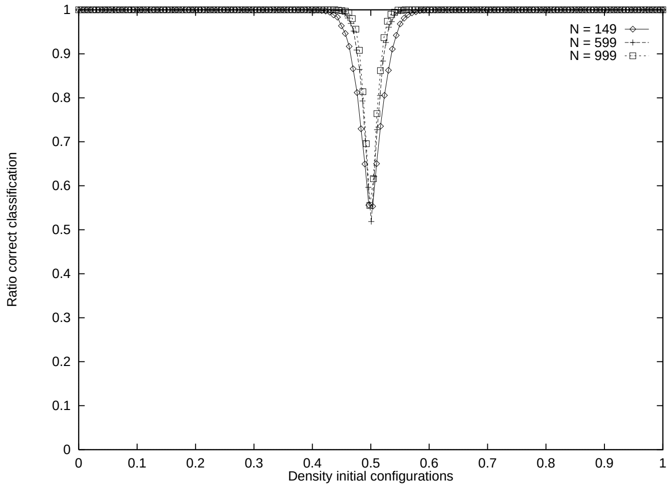


Figure 6.1: Distribution of performance for the GKL rule for $\rho_0 \in [0.0, 1.0]$.

other researchers.

6.1.2 Experimental Setup

For the majority classification task, it is believed that Initial Configurations (ICs) become more and more difficult to classify correctly as their density gets closer to the ρ_c threshold. This hypothesis is supported by the distribution of the performance for the GKL rule for $\rho_0 \in [0.0, 1.0]$ presented in figure 6.1. Therefore, our idea is to construct a framework that adapts the distribution of the density for the population of ICs as CA-rules improve at performing the task. The following definition for the fitness of rules and ICs has been used to achieve this goal.

$$f(R_i) = \sum_{j=1}^{n_{IC}} weight\_IC_j \times covered(R_i, IC_j)$$

where:

$$weight\_IC_j = \frac{1}{\sum_{k=1}^{n_R} covered(R_k, IC_j)}$$

and

$$f(IC_j) = \sum_{i=1}^{n_R} weight\_R'_i \times E(R_i, \rho(IC_j)) \times \overline{covered(R_i, IC_j)}$$

where:

$$weight\_R'_i = \frac{1}{\sum_{k=1}^{n_{IC}} E(R_i, \rho(IC_k)) \times \overline{covered(R_i, IC_k)}}$$

where $covered(R_i, IC_j)$ returns 1 if a CA using rule R_i and starting from initial configuration IC_j relaxes to the correct state. Otherwise, it returns 0. $\overline{covered(R_i, IC_j)}$ returns the complement of $covered(R_i, IC_j)$.

This definition implements a competitive relationship with resource sharing. However, a new component, namely $E(R_i, \rho(IC_j))$, has been added in the definition of the ICs' fitness. Indeed, in order to apply the concept of the "Ideal" trainer to the majority classification task, the space of ICs has been decomposed into equivalence classes, each equivalence class corresponding to all the ICs having the same density. The underlying idea behind the definition of those equivalence classes is that randomly chosen ICs with the same density are considered as having similar difficulty. This setup also allows the construction of a new set of ICs at each generation according to a given distribution for the density. Indeed, the strategy of generating a new set of ICs at each evolutionary step appears to result in the discovery of rules that are more reliable because they are evaluated against a variety of ICs. For this reason, each individual in the population of ICs represents a density instead of a fixed instance of an initial configuration.

The purpose of the new component $E(R_i, \rho(IC_j))$ is to penalize ICs with density $\rho(IC_j)$ if little information is collected with respect to the rule R_i. Indeed, we consider that if a rule R_i has a 50% classification accuracy over ICs with density $\rho(IC_j)$ then this is equivalent to random guessing and this R_i shouldn't contribute to the fitness of IC_j. On the contrary, if the performance of R_i is significantly better or worse than the 50% threshold for a given density of ICs this means that R_i captured some relevant properties to deal with those ICs. Once again, the idea is that the training environment should be composed of ICs that provide useful information and, therefore, it should avoid proposing problems (i.e., ICs) that are too difficult or too easy. To achieve this goal, the purpose of the component $E(R_i, \rho(IC_j))$ is to maintain a balance between the search for new ICs that are more difficult while still being covered by some rules. In fact, this component extends the model of competitive coevolution by introducing a form of cooperation.

In order to achieve continuous progress, our implementation exploits an intrinsic property of the $\rho_c = 1/2$ task. Indeed, CA-rules that cover ICs with density $\rho_0 < 1/2$ (resp. $\rho_0 > 1/2$) with high performance are usually successful to also cover ICs with density $\rho_0' < \rho_0$ (resp. $\rho_0' > \rho_0$). Therefore, as ICs become more difficult, their density approaches $\rho_0 = 1/2$ but rules don't have to be tested against easier ICs. Following this idea, we defined $E(R_i, \rho(IC_j))$ as the complement of the entropy of the outcome between rule R_i and ICs with density $\rho(IC_j)$:

$$E(R_i, \rho(IC_j)) = \lg(2) + p\lg(p) + q\lg(q)$$

where: p is the probability that an IC with density $\rho(IC_j)$ defeats the rule R_i and $q = 1 - p$. $E()$ implements the distance measure discussed in section 5.3. Its purpose is to maintain the balance between the search for more difficult ICs and ICs that can still be solved by some rules. In practice, the entropy is evaluated by performing some statistics over the population of ICs.

6.1.3 Experimental Results

Experiments were performed with different sizes for the population of rules and ICs. The best rule whose performance is reported in Table 6.2 resulted from the experiments that used the largest population size. In those experiments, 6 runs were performed for $5,000$ generations, using a size of $1,000$ for both populations. Each rule is coded on a

Coevolution rule	00010100	01011111	01000000	00000000
	00010111	11111100	00000010	00010111
	00010100	01011111	00000011	00001111
	00010111	11111111	11111111	11010111
Das rule	00000111	00000000	00000111	11111111
	00001111	00000000	00001111	11111111
	00001111	00000000	00000111	11111111
	00001111	00110001	00001111	11111111
ABK rule	00000101	00000000	01010101	00000101
	00000101	00000000	01010101	00000101
	01010101	11111111	01010101	11111111
	01010101	11111111	01010101	11111111
GKL rule	00000000	01011111	00000000	01011111
	00000000	01011111	00000000	01011111
	00000000	01011111	11111111	01011111
	00000000	01011111	11111111	01011111

Table 6.1: Description of the current best rule and published rules for the $\rho_c = 1/2$ task.

binary string of length $2^{2*r+1} = 128$. One-point crossover is used with a 2% bit mutation probability. The population of rules is initialized according to a uniform distribution over $[0.0, 1.0]$ for the density. Each individual in the population of ICs represents a density $\rho_0 \in [0.0, 1.0]$. This population is also initialized according to a uniform distribution over $\rho_0 \in [0.0, 1.0]$. At each generation, each member generates a new instance for an initial configuration with respect to the density it represents. All rules are evaluated against this new set of ICs. The generation gap is 5% for the population of ICs (i.e., the top 95% ICs reproduce to the next generation). There is no crossover nor mutation. The new 5% ICs are the result of a random sampling over $\rho_0 \in [0.0, 1.0]$ according to a uniform probability distribution. The generation gap is 80% for the population of rules. New rules are created by crossover and mutation. Parents are randomly selected from the top 20%. All runs consistently evolved some rules that score above 82%. Table 6.1 describes lookup tables for the current best CA rule and other rules discussed in the literature.

N	149	599	999
Coevolution	0.863 +/- 0.001	0.822 +/- 0.001	0.804 +/- 0.001
Das rule	0.823 +/- 0.001	0.778 +/- 0.001	0.764 +/- 0.001
ABK rule	0.824 +/- 0.001	0.764 +/- 0.001	0.730 +/- 0.001
GKL rule	0.815 +/- 0.001	0.773 +/- 0.001	0.759 +/- 0.001

Table 6.2: Performance of different published CA rules and the new best rule for the $\rho_c = 1/2$ task.

The leftmost bit corresponds to the result of the rule on input 0000000, the second bit corresponds to input 0000001, ... and the rightmost bit corresponds to input 1111111.

Figure 6.2 describes the evolution of the density of rules and ICs for one run. As rules improve, their density gets closer to 1/2 and the density of ICs is distributed on two peaks on each side of $\rho_c = 1/2$. In that particular run, it is only after $1,300$ generations that a significant improvement is observed for rules and that, in response, the population of ICs adapts dramatically in order to propose more challenging initial configurations. This shows that our strategy to coevolve the training environment and the learners has been successfully implemented in the definition of the fitness functions.

Figure 6.3 presents three examples of the space-time evolution of a CA for this new best rule. For the left diagram, where $\rho_0 < \rho_c$ and the middle one, where $\rho_0 > \rho_c$, the CA relaxes to the correct configuration. The third diagram shows an instance for which the CA doesn't relax to any of the two desired convergence patterns. Indeed, contrary to the GKL rule, the CA doesn't always relax to one of the two fixed configurations for this new rule.

As a comparison, [4] used a population of size $51,200$ and runs were performed for 50 generations. In their work, the training environment was composed of a fixed training set of 1000 ICs constructed from a uniform sampling from the space of all ICs (thus, the distribution for the density of ICs in the training set is binomial, centered

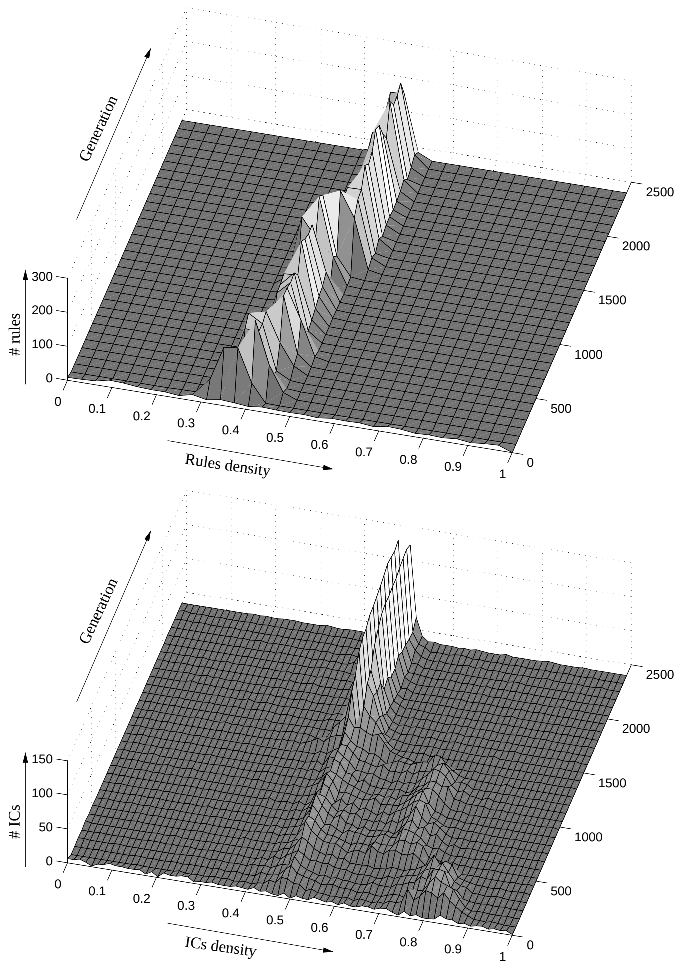


Figure 6.2: Coevolutionary learning between CA rules (top) and ICs (bottom): the difficulty of problems proposed by the population of ICs adapts in response to the progress of the population of rules in order to maintain a challenging environment and to allow continuous progress.

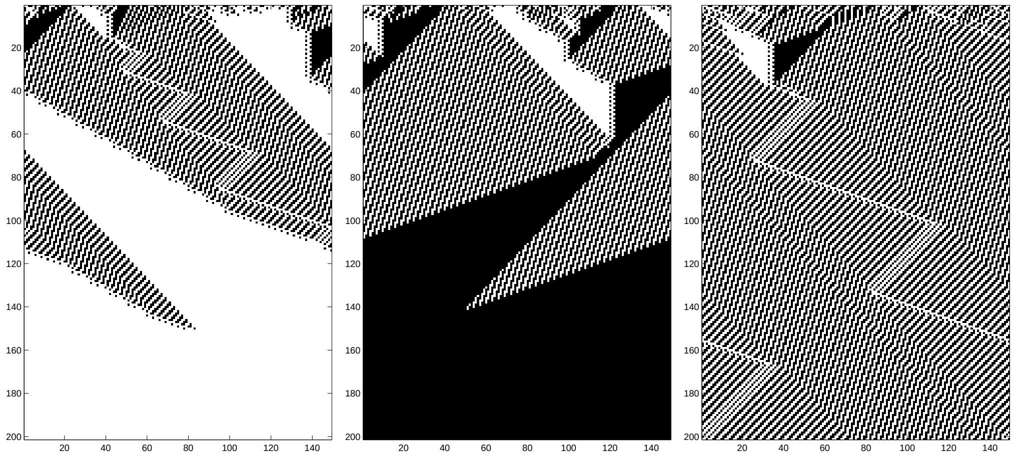


Figure 6.3: Three space-time diagrams describing the evolution of CA states: in the first two, the CA relaxes to the correct uniform pattern while in the third one it doesn't converge at all to a fixed point.

on 1/2). In experiments described in [22, 59], the learning environment is composed of a set of 100 ICs sampled at each generation according to a uniform distribution over $\rho_0 \in [0.0, 1.0]$. Those authors acknowledged that this distribution for the sampling of the space of ICs, while helpful to bootstrap the search, might no longer provide useful information once some average performance rules have been discovered.

6.1.4 Performance Comparison: Fixed vs. Adapting Search Environment

The purpose of this section is to illustrate experimentally that the search performed in a framework where the environment responds appropriately to the progress of individuals does result in an improved performance. This analysis is composed of a set of three experiments. In all experiments, a population size of 200 is used for rules. For each set of experiments, the population of ICs is generated as follows:

- **Experiment 1**: rules are evaluated in an environment composed of 200 ICs drawn at each generation from an *unbiased* distribution. This means, that the distribution of the density of ICs is binomial centered on 0.5.

- **Experiment 2**: rules are evaluated in an environment composed of 200 ICs drawn at each generation from a *biased* distribution, such that the distribution for the density of ICs is uniform.

- **Experiment 3**: rules are coevolving with a population of ICs composed of 200 individuals. The "Ideal" trainer framework is implemented to control the evolution of the population of ICs.

For all those experiments, the definition for the fitness of rules is identical:

$$f(R_i) = \sum_{j=1}^{n_{IC}} weight\_IC_j \times covered(R_i, IC_j)$$

where:

$$weight\_IC_j = \frac{1}{\sum_{k=1}^{n_R} covered(R_k, IC_j)}$$

This definition implements resource sharing in order to allow multiple species in the population of rules. In the coevolutionary experiments, the definition for the fitness of ICs is the same as the one introduced in section 6.1.2:

$$f(IC_j) = \sum_{i=1}^{n_R} weight\_R'_i \times E(R_i, \rho(IC_j)) \times \overline{covered(R_i, IC_j)}$$

where:

$$weight\_R'_i = \frac{1}{\sum_{k=1}^{n_{IC}} E(R_i, \rho(IC_k)) \times \overline{\overline{covered(R_i, IC_k)}}}$$

and:

$$E(R_i, \rho(IC_j)) = \log(2) + p\log(p) + (1-p)\log(1-p)$$

The representation for rules and ICs is also identical to the one in previous experiments. In the coevolutionary experiments, each individual in the population of ICs represents

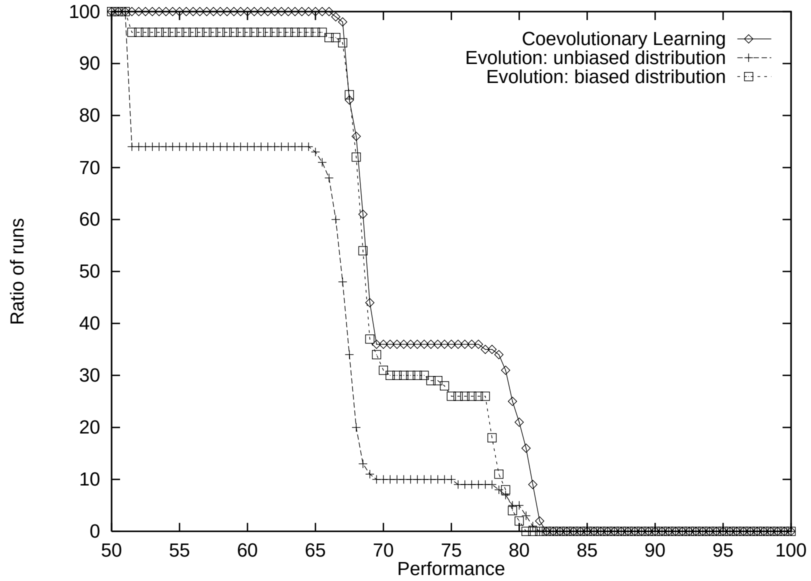


Figure 6.4: Distribution of the ratio of runs achieving a specific performance after 500 generations.

a density $\rho_0 \in [0.0, 1.0]$. At each generation, each member generates a new instance for an IC with respect to the density it represents. For each set of experiments, 100 runs were performed. For the three sets of experiments, the generation gap is 80% for the population of rules (i.e., the top 20% reproduces in the next generation). For the third set of experiments, the generation gap is 3% for the population of ICs.

At each generation, the best rule with respect to the number of ICs it covers is evaluated against $5,000$ ICs drawn randomly according to the unbiased distribution. Then, after 500 and $1,000$ generations respectively, the average of the performance for the best 10 top rules is computed. An average over a window of size 10 is considered in order to smooth the noisy evaluation for the performance of rules. Figures 6.4 and 6.5 describe the evolution of the ratio of runs for which this average is above a given performance (given on the x axis).

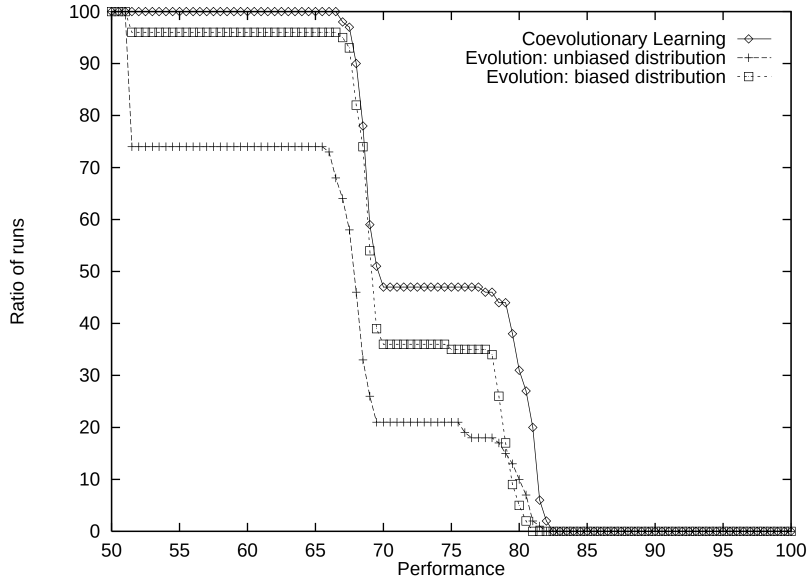


Figure 6.5: Distribution of the ratio of runs achieving a specific performance after 1000 generations.

The following observations can be made from those figures. First, the "Ideal" trainer framework always results in a higher ratio of success. In particular, there is a 100% success rate for generating a rule of performance above 67% (about 96% of success with the biased distribution and about 74% of success with the unbiased distribution). Moreover, coevolution also results in the discovery of rules of higher performance (a few rules with performance above 82% were discovered).

Second, the experiments with the biased distribution resulted in a higher success rate than the experiments with the unbiased distribution. Indeed, in the case of an unbiased distribution, rules are exposed to ICs with density close to 0.5, which are the most difficult. As a result, there is little information about the gradient and a large number of runs don't even discover rules that do significantly better than random guess. However, for those few runs were good rules have been discovered, the unbiased setup proposes a better environment to continue improvement than the biased setup. The reason is that the biased environment doesn't offer a challenging environment anymore to the population of rules because most of the ICs are covered by those rules. As a result, there is almost no gradient to drive the search. On the contrary, the unbiased environment still presents a gradient for search and results in the discovery of rules that have better performance than the best rules discovered in the experiments with the biased distribution, both after 500 and $1,000$ generations.

6.1.5 Analysis of Experiments

The goal of this section is to provide some insights about the role of coevolution in the discovery of the new improved rule whose performance is presented in Table 6.2. This analysis is performed with the data gathered from the 6 runs discussed in section 6.1.2. Those runs exploited a population size of $1,000$ for rules and ICs and were performed for $5,000$ generations.

Dynamics of Coevolutionary Search

In the work of Mitchell, Crutchfield and Hraber [59], "epochs of innovation" were observed in the evolution of the GA. Each of those epochs corresponds to the discovery of more elaborate strategies to perform the task. The discovery of such new strategies usually results in a significant increase in the performance of rules. For the majority classification task, two such strategies have been identified:

- the *block expanding* strategy for which the size of uniform domains (i.e., blocks of all 0's or all 1's) progressively increases. The underlying idea is that a large block of cells with identical state is more likely to occur if there is a majority of cells in that particular state in the initial configuration. Rules implementing that strategy have usually a performance in the range 0.65 to 0.70.

- the *particle-based* strategy for which the resulting computation performed by the cellular automata is more elaborate. At a higher level of abstraction, this computation can be described in terms of interacting "particles". The underlying principle of particle-based computation is that information about local properties of the state of the cellular automata is computed and propagated by "particles". As described by Crutchfield, Mitchell and Hraber [59], the analysis of the space-time diagram for the dynamics of the cellular automata reveals some domains that correspond to the recurrence of identical patterns, and particles corresponding to the boundary between such domains. Then, the interactions between those particles result in the processing of this information about local properties and eventually decide of the final outcome. That is, whether the cellular automata relaxes to all 0's or all 1's. The performance of rules that implement accurately such a strategy is usually above 0.75.

The issue addressed in this section is to determine whether similar epochs of improvement also occur in the coevolutionary runs. The answer to this question would provide some information about the dynamics of the search performed by the "Ideal" trainer framework and whether it presents some significant differences compared to experiments performed in [59].

Figure 6.6 plots the evolution of the performance of the top individual for the first 200 generations. Each individual is evaluated against $5,000$ ICs drawn according to the unbiased distribution. It appears that epochs corresponding to the $0.65 - 0.70$ performance do occur. A transition from performance between 0.65 and 0.70 to rules with performance above 0.75 also occur in all those runs. Then, some slow improvement over several hundreds of generations is observed.

In all of those 6 runs, some rules were discovered with performance between 82% and 84%. However, for the run that resulted in the best rule (run #4), another

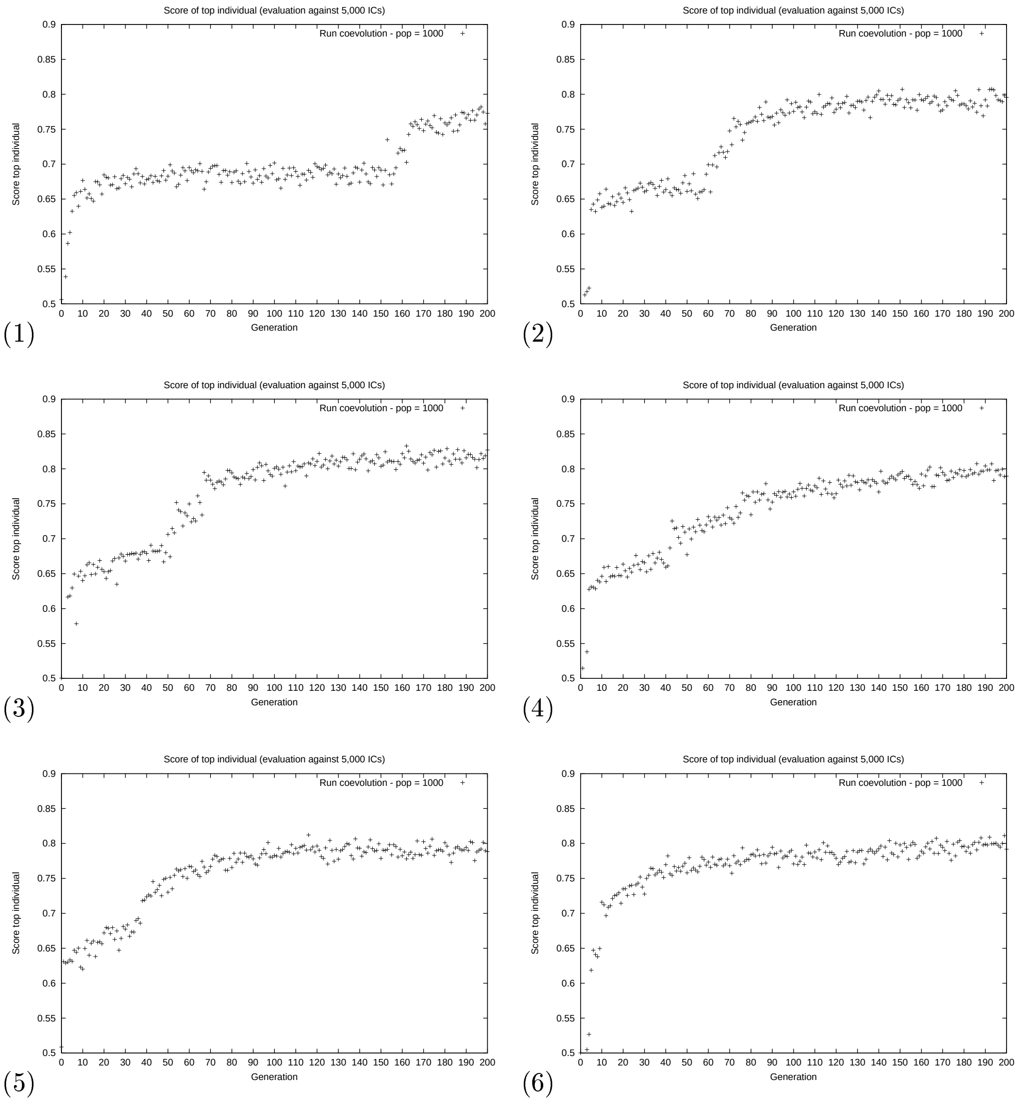


Figure 6.6: Evolution of the performance of the top individual in the first 200 generations for each run.

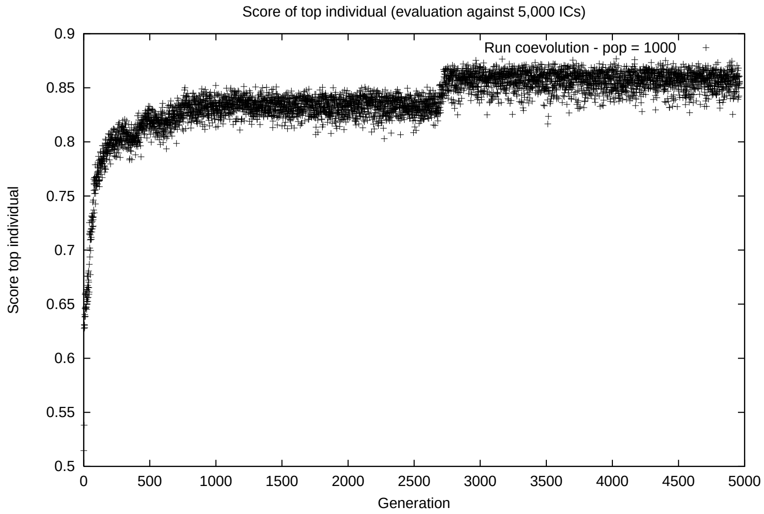


Figure 6.7: Evolution of performance of top individual for the 4^{th} run.

important transition occurs around generation $2,500$, as shown in Figure 6.7. In order to determine whether this transition also results in a significant modification of the lookup table for the corresponding rules, the evolution of the hamming distance between the top rule of each generation and the overall best rule is plotted in Figure 6.8. It appears that this transition corresponds to some changes in the lookup table involving between 10 and 15 entries (out of 128).

Another approach to analyze the contribution of coevolution in the search process is to study the evolution of the distribution for the density of ICs. Figure 6.9 plots this distribution for run #4 at generations: 20, 50, 100, 250, 1000 and 2500. At generation 20, the ICs are almost uniformly distributed between the densities 0.20 and 0.80. Low and high densities are no longer represented at that time since some "block-expanding" rules have already been discovered (as shown in Figure 6.6-(4)). Then, this distribution progressively converges to a two-peak distribution centered on the 0.50 density. However, it appears that this distribution doesn't change too much after 1000

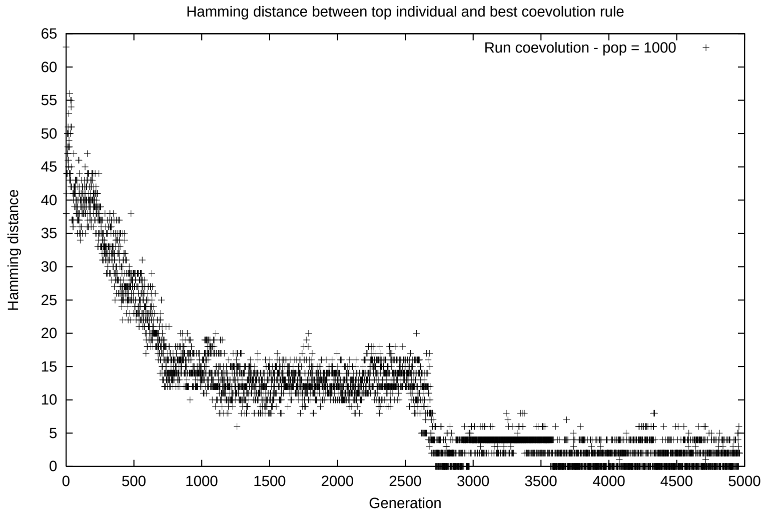


Figure 6.8: Hamming distance between top individual and best rule for the 4^{th} run.

generations. The problems posed by the population of ICs no longer offer a dynamic training environment to the population of rules because the boundary for the maximum difficulty of problems has been reached (since problems with density immediately above and below 0.50 are not represented in the population of ICs as a result of implementation choices). Coevolution doesn't seem to contribute significantly after generation $1,000$. However, it is likely that its role in the early generations might have driven the search in some areas of the rule space where progress is still possible. Unfortunately, this last point seems difficult to prove since it would rely on some properties about the structure of the rule space that are still unknown.

Indeed, the underlying pressure toward adaptability implemented by the coevolutionary framework works only if some appropriate properties are exhibited by the search space. In the case of the majority classification task, it has been shown experimentally that incremental improvement is favored by coevolution. That is, by starting with simple problems and then gradually increasing the difficulty of problems as rules

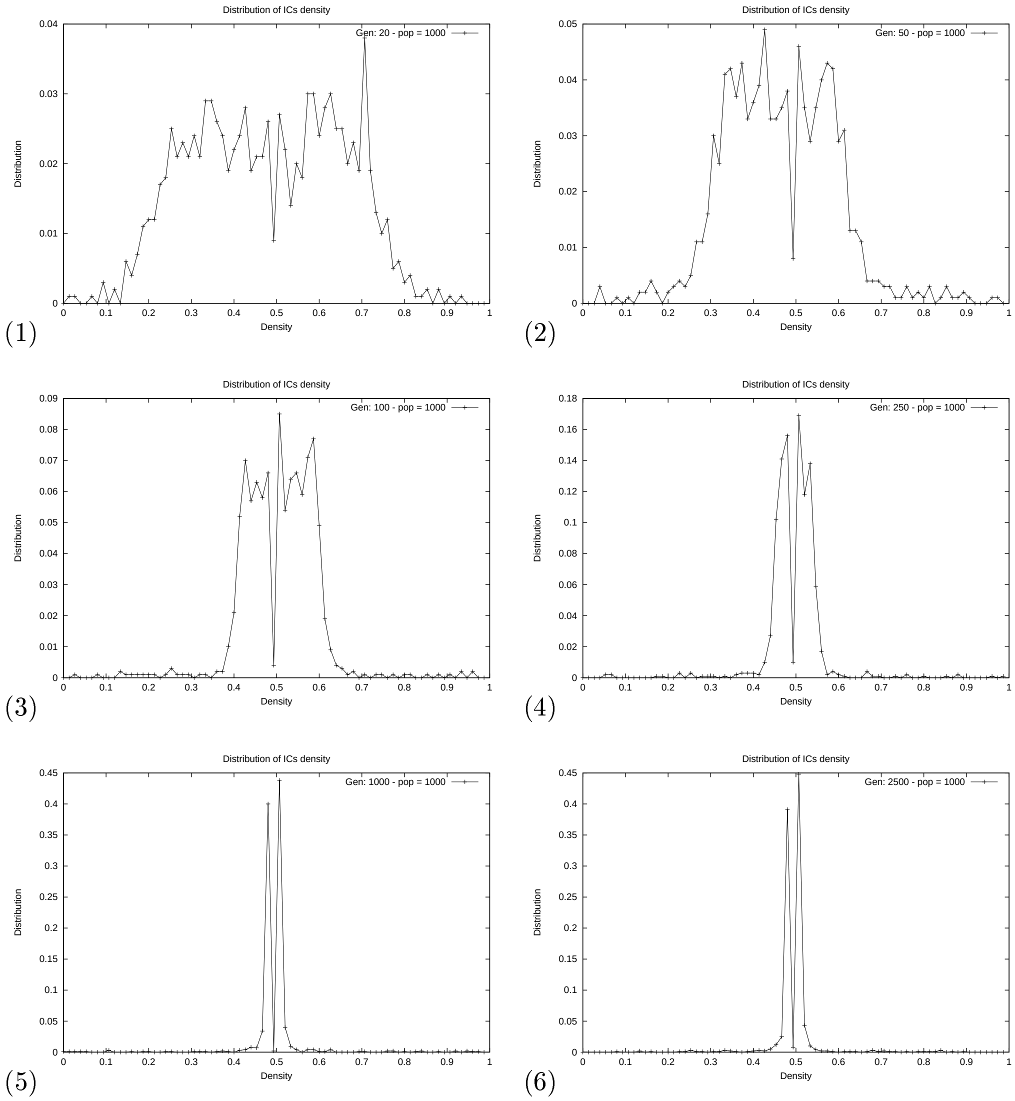


Figure 6.9: Distribution of ICs' density for generations: 20, 50, 100, 250, 1000 and 2500.

become better, it is possible to build up from the current best rules and discover even better ones. This means that some implicit properties about the problem are exploited.

Since the properties of the search space are determined by the choice for the representation to describe rules, it would be interesting to identify whether some relevant features that would favor adaptability are implemented in this representation language.

At this stage of the research concerning cellular automata and the implementation of the majority classification task with CAs, it is only possible to conjecture. One theory is that “particles” are used as building blocks in the representation of rules and that some particular combinations of particles allow the emergence of rules with better adaptability. Indeed, since particles correspond to boundaries between different domains, the relevant entries associated with a given particle can be identified in the lookup table of a rule. However, the process of identifying the entries corresponding to a given particle in lookup tables is tedious since no tools seem to exist to perform that task automatically (partly because the identification of domains is also a difficult task). Moreover, it is common that different particles share multiple entries in the rule description.

Analysis of New Best Rule (Coevolution Rule)

It appears that the number patterns involved for the definition of the different domains (and therefore the number of particles) is much larger in the case of the new best rule compared to the GKL rule (for instance). One of the most important feature in the case of the GKL rule is that all-white and all-black domains emerge very early by wiping out the small isolated islands of 1’s and 0’s. Such early “loss of information” doesn’t seem to occur in the case of the coevolution rule. In order to compare qualitatively different rules, the entries in the lookup tables associated with those rules are grouped into classes. Each class corresponds to the entries associated with input patterns having

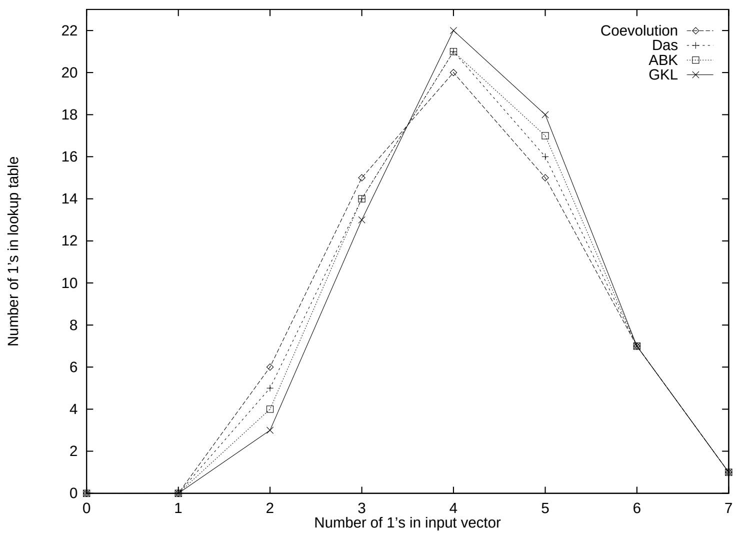


Figure 6.10: Distribution for the count of 1's in the lookup tables associated with rules for different densities of the input pattern.

the same density. The number of elements in each class is: $\{1, 7, 21, 35, 35, 21, 7, 1\}$ which correspond respectively to input patterns having a number of 1's in the range 0 to 7. Then, for each class, the count of entries that return a 1 in the lookup table is evaluated. The distribution for this count is plotted in Figure 6.10 for the four rules: coevolution, Das, ABK and GKL. From this figure, it appears that, as the rules are getting better (in terms of performance for the $\rho = 1/2$ task), the count of 1's for low density input pattern is getting higher. This means that the loss of information occurring in early time steps due to isolated islands of 0's and 1's being wiped out is delayed. In fact, there might be a trade-off between avoiding losing too much information and the emergence of particles. Indeed, the creation of particles requires the formation of stable patterns in space and time which, in turn, requires some loss of information about local properties of the initial configuration.

The data for the plots in Figure 6.10 is reported in Table 6.3 which presents also the distribution for the count of 0's. This table shows an interesting property for this

Density	Coevolution		GKL		Das		ABK	
input pattern	# 1's	# 0's	# 1's	# 0's	# 1's	# 0's	# 1's	# 0's
0	0	1	0	1	0	1	0	1
1	0	7	0	7	0	7	0	7
2	6	15	3	18	5	16	4	17
3	15	20	13	22	14	21	14	21
4	20	15	22	13	21	14	21	14
5	15	6	18	3	16	5	17	4
6	7	0	7	0	7	0	7	0
7	1	0	1	0	1	0	1	0

Table 6.3: Distribution for the count of 1's and 0's in the lookup table with respect to the density of the input pattern

distribution. Indeed, for those four rules, there is a symmetry for the distribution of the count of 1's and 0's (reading one distribution top-down and the other bottom-up). That same property has been tested and is also satisfied for a few more high performance rules (i.e., performance around 85% for $N = 149$) that have been discovered using coevolution. The reasons why such a property emerged haven't been understood yet.

6.1.6 Concluding Remarks

The application of the "Ideal" trainer paradigm to the majority classification task implemented by CAs resulted in the discovery of a new rules which improves very significantly over the performance of previously known rules (by at least four points for $N = 149, 599$ and 999).

The dynamics of search implemented by the "Ideal" trainer paradigm have been illustrated with diagrams (Figure 6.2) describing the evolution of rules and ICs (or, more precisely, the evolution for the distribution of rules density and ICs density). The analysis of those diagrams confirms that the conditions for achieving continuous progress are indeed implemented in the coevolutionary framework. That is, the population of ICs exposes the population of rules to problems that are not too difficult with respect to their actual performance and that the population of rules is challenged with problems of increasing difficulty over time.

The performance of the "Ideal" trainer paradigm has been also compared with two evolutionary learning procedures that propose a static training environment. That is, the distribution for the density of ICs doesn't adapt in response to the progress of the population of rules. Experiments with an unbiased (binomial) distribution and a biased ("uniform") distribution have been performed. In both cases, and with similar computational resource involved, the "Ideal" trainer setup consistently performed significantly better.

6.2 Application 2: a Modular Approach to Inductive Learning

6.2.1 Modular Approaches to Inductive Learning: Presentation

In the field of inductive learning, among approaches for the construction of a classification theory or a continuous model of an observed system (e.g., time series prediction) are the following: (1) to construct a single model that is defined over the entire input space, or (2) to construct many local models from the query vectors or from the superposition of local approximators. Locally weighted learning techniques [8] or radial basis function networks (RBF) [13] are examples of the second approach.

Modular Inductive Learning (MIL) is a bottom-up approach to inductive learning which attempts to find a balance between those two approaches. The inductive learning strategy implemented in MIL consists in exploring at the same time a space of local models and a space of decompositions of the input space. Two goals underlie the

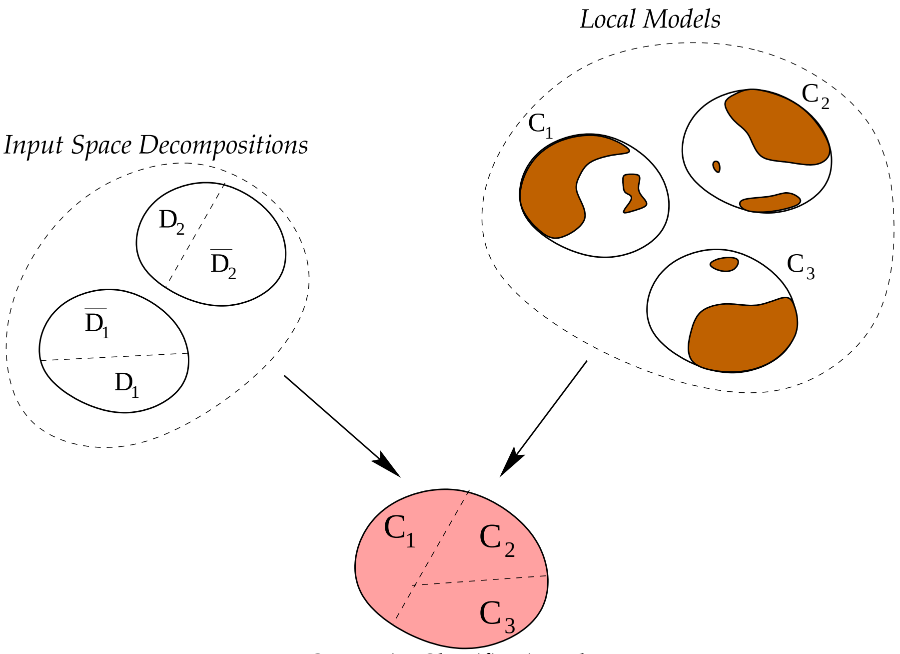


Figure 6.11: The space of decompositions and the space of local models are explored simultaneously. If some decompositions characterizing the domain of specialization of some local models (represented in dark) are discovered, then a composite classification theory may be constructed. In that example, the composite solution is defined as follows: **if** $(x, y) \in \mathrm{D}_2$ **then** C_1 **else if** $(x, y) \in \mathrm{D}_1$ **then** C_3 **else** C_2

strategy implemented in MIL for the exploration of those two spaces. The first one is to discover some local models that propose an accurate approximation of the training data over some domain of the input space, along with decompositions of the input space that capture the domains of "specialization" of those local models. The central idea motivating that strategy is that a composite classification theory with good accuracy over the entire input space may then be constructed. This strategy is illustrated with the diagram in Figure 6.11.

The second goal consists in discovering a composite solutions using as few components as possible while still exhibiting a high accuracy with respect to the training data. The motivation for that goal is to induce classification theories that also generalize to input data outside the training set. This is a direct application of Occam's razor.

The challenge of Modular Inductive Learning is to implement the right balance between accuracy and the complexity of the final classification theory (that is, the number of components in its description).

6.2.2 Related Work

Similar approaches to modular inductive learning have been proposed by researchers. For instance, this strategy has been investigated for concept learning in the field of Inductive Logic Programming [24, 30, 31, 37]. In those works, the idea is to use niching to construct multi-modal concepts, i.e. concepts represented in Disjunctive Normal Form (DNF). In such a multi-modal concept, a niche corresponds to each disjunct. A bias is introduced in the fitness function for the evolution of more general concepts resulting in a composite solution with fewer components. Darwen [21] also used that principle for the construction of modular solutions to the game of prisoner's dilemma. In his work, niching is used for the discovery of a variety of different strategies. Then, a gating algorithm allows the construction of a composite solution from those elementary strategies. The function of the gating algorithm is to determine which elementary strategy would be the most appropriate given the recent history of interactions with the opponent. Along those lines, Potter [81] proposed a model of cooperative coevolution in which each population is dedicated to the exploration of the search space associated with a component of a problem solution. In that system, the interactions between populations is defined explicitly by the designer. Potter also developed a specific scheme to adapt the number of collaborating populations in response to the decomposability and the dimensionality of the problem. As discussed in Section 4.3.2, this model provides a paradigm for extending EC methods by introducing explicitly the notion of modularity as a strategy to address problems of increasing complexity.

However, in these related works, the decomposition strategy is very simplistic. It is usually hard coded or it involves a simple learning procedure. The emphasis on finding accurate local models introduces a bias toward a particular type of composite models which may not result in classification theories that generalize well.
On the contrary, the goal of MIL is to introduce more flexibility by exploring simultaneously the space of decompositions and the space of local models. The central idea underlying this strategy is that the dynamics of search in those two spaces may result in the discovery of classification theories with better generalization ability.

6.2.3 Issues Concerning the Automatic Decomposition of Problems

There has been some attempts in the past to address the problem of modular inductive learning using the same strategy as MIL but with limited success (e.g., Rosca's Evolutionary Divide-and-Conquer (EDC) framework [89, chapter 7]). Indeed, the correct implementation of this strategy is very complex because it involves a search along different interdependent dimensions, namely the space of decompositions and the space of local models.

When constructing composite solutions, the problem of determining the contribution of each component to the total performance of the system is known as the *credit assignment* problem. Being able to identify important components in a composite system is a valuable source of information for controlling the search for improvements of that system. Therefore, the issue of credit assignment must be addressed in order to The process of assigning credit can be performed according to different strategies, depending on the properties of the problem and the available knowledge concerning its decomposability. At one extreme, explicit strategies consider each component individually and evaluate their usefulness to address the global problem. This approach is possible when a problem can be clearly decomposed into independent subproblems. At the other extreme, implicit strategies don't compute directly any performance value for components. Instead, those strategies favor the emergence of high quality components by exploiting some indirect mechanisms. Moriarty's SANE system [65] or Angeline and Pollack's work on the emergence of modularity [6] are two examples that illustrate such implicit strategies.

Moreover, for the same reasons which prove that there is no universal search algorithm doing uniformly better than random search over all possible state space ("No Free Lunch" theorem [114]), there exists no "universal" strategy for modular inductive learning in terms of automatically constructing an optimum decomposition and determining its components across all possible problems. Therefore, there is necessarily an explicit bias in any system implementing the idea of modular inductive learning and, as a result, the performance of such a system will depend on the properties of the classification (or modeling) task under consideration. This explicit bias depends on the strategy which controls search in the space of decompositions and the space of local models along with the representation language that defines those state spaces. One of our motivations in the design of the architecture of the MIL system is to propose a framework in which the different mechanisms addressing those issues are clearly identified.

6.2.4 Applying the "Ideal" Trainer Paradigm

The motivation for exploiting the "Ideal" trainer paradigm for addressing the problem of modular inductive learning originated from the requirement for inducing theories with high generalization ability. Indeed, by maintaining a pressure toward adaptability, the "Ideal" trainer paradigm favors the emergence of agents that capture some intrinsic

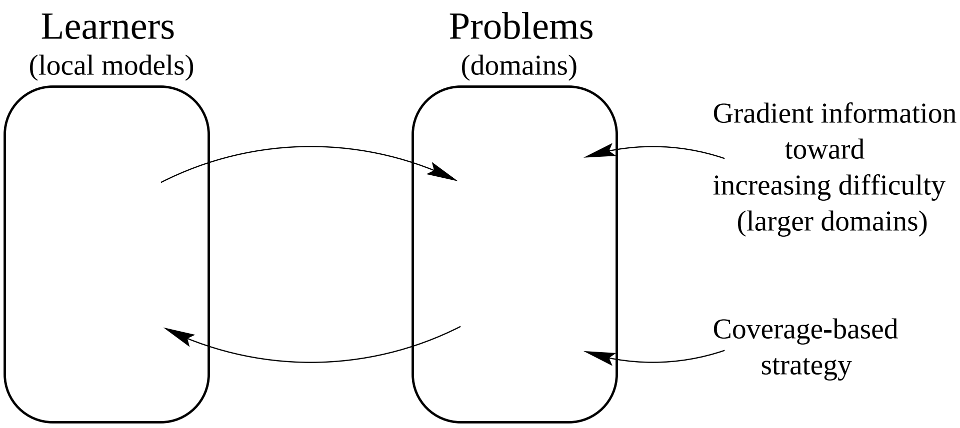


Figure 6.12: Extension of the "Ideal" trainer concept for Modular Inductive Learning.

properties of the changing environment. This side effect of adaptability is fundamental to the notion of generalization because it allows the construction of compact descriptions. The central idea of the MIL system is to exploit that feature as the fundamental strategy to control search in the space of decompositions and the space of local models. A typical scenario for the dynamics of search performed by MIL could be described as follows. In a first stage, a classification theory composed of several local models and covering the entire training set is constructed. Usually, this initial solution has poor generalization ability. Then, pressure is introduced in order to focus the search toward the exploration of local models that fit larger domains of the input space. This pressure toward larger domains induces an evolutionary race between local models resulting in the emergence of local models that generalize better outside those domains. By making sure that the entire input space is always covered by some local models, this strategy may result in the discovery of compact composite solutions(i.e., theories involving fewer components) with good accuracy.

In summary, the MIL system is a natural extension of the "Ideal" trainer concept. Learners correspond to local models (or concepts) while problems correspond to domains.

Then, as represented in Figure 6.12, a coverage-based heuristic is introduced in the population of domains. This heuristic allows the creation of niches which specialize on different regions of the input space. By maintaining this coverage of the input space, it is always possible to construct composite classification theories composed of high quality local model.

The next sections describe the technical details of our particular implementation for MIL. First, the global architecture of the system is presented. Evolutionary search is performed at different levels of abstraction. At the lowest level, the space of decompositions (domains) and the space of local models are explored. Then, a third population explores the space of pairs *(Domain, Solution)* for good matches. Finally, a fourth population explores the space of composite solutions constructed from pairs in the former population. Section 6.2.6 describes the rules of evolution that control search in each population and the interactions between members of different populations.

6.2.5 Architecture of the MIL System

Our approach to Modular Inductive Learning combines several mechanisms:

- A coverage-based heuristic implemented using competition for resources. The goal of this heuristic is to maintain niches such that all training examples are covered. As a result, composite solutions that cover the entire input space with good accuracy may be constructed. Here, each training data is considered as a resource which is shared among pairs *(Domain, Solution)*. A resource is "exploited" by a pair if it is selected by the domain. The efficiency of the pair to exploit a resource is determined by a measure of *cooperation performance* between the domain and the solution.

- Adaptation at the level of pairs *(Domain, Solution)* in order to isolate solutions that have better accuracy over a given domain.

- Pressure toward larger domains, providing that some solutions still have a reasonable accuracy over that new domain. The definition of the measure of cooperation performance introduces a bias which controls the balance: domain size vs. accuracy. As discussed in section 6.2.3, some explicit bias must be introduced in order to define a preference toward a particular type of decomposition.

- Parallel search allowing the simultaneous exploration of different alternatives for the construction of composite solutions. Here again, a balance between the exploitation of current composite solutions and the exploration for new decompositions must be maintained.

Those heuristics work at different levels of granularity: composite solutions, pairs (Domain, Solution), domains and solutions. To embed all those mechanisms in a single system, the architecture described in Figure 6.13 has been implemented. The elementary entities are domains and solutions. A population of evolving structures correspond to each of them. The population of pairs is composed of pairs of pointers to elements in the population of domains and solutions. Each composite solution is composed of a list of pointers to pairs. The list provides an order for the selection of the component generating the output value in response to input vectors. The last element of the list corresponds to the default component for generating the output when none of the previous domains in the list matches the input vector.

This architecture and Hierarchical SANE [64] have many features in common. SANE (Symbiotic, Adaptive Neuro-Evolution) [62] is an evolutionary inspired system for the design of neural networks applied to problems in reinforcement learning. In its first implementation, SANE was composed of a unique population of evolving agents, each agent representing one hidden unit of a neural network. The fitness of agents is

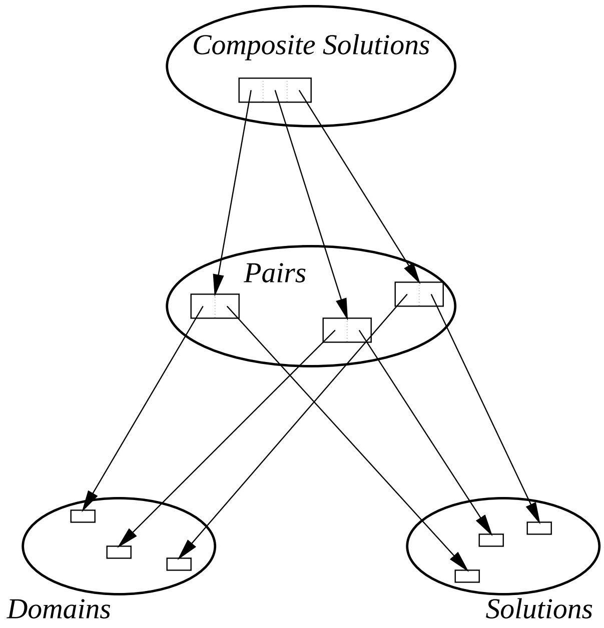


Figure 6.13: Architecture of the Modular Inductive Learning system

based on the evaluation of neural networks constructed from randomly selected groups of hidden units. While being successful, a major drawback of this architecture concerns its inability to keep track of groups of hidden units that resulted in high quality neural networks. Hierarchical SANE addressed this issue by introducing a population of network specifications (or blueprints), each specification being composed of a list of pointers to the population describing hidden units. This architecture allows a better control of the balance between exploitation and exploration. This balance is achieved by keeping track of successful groups of hidden units in order to perform local search (exploitation) and by sampling the space of groups of hidden units to explore new solutions. The first SANE algorithm exploited only a sampling strategy to construct solutions and distribute credit, but it didn't keep track explicitly of successful groups of hidden units.

6.2.6 Rules of Evolution

This section describes the rules of evolution for the different populations that compose the MIL system. A fitness is designed to evaluate individuals in each population. However, the different populations do not evolve independently. First, the MIL system implements a "bottom-up" approach by searching for pairs *(Domain, Solution)* that provide a high accuracy model over large domains of the input space, and composite solutions constructed from those pairs. Second, MIL also implements a "top-down" approach by focusing search on pairs that participate in the best composite solutions and the domains and solutions that compose the best pairs.

The following sections describe in detail the rules of evolution for each population.

Composite Solutions

- *Representation*: Each composite solution is represented by a list of pointers to pairs.

- *Search operators*: A greedy algorithm is used for constructing composite solutions from a given set of pairs. The output of this algorithm is an ordered list of pairs. At each step of the construction, priority is given to the pair covering the largest number of training examples not covered yet by the other pairs selected so far. Local search is performed by adding a group of randomly selected pairs to those in the current description of a composite solution and, then, by running the greedy construction algorithm. A recombination operator is defined using a similar technique. In that case, the initial set of pairs is composed of the genetic material (i.e., the set of pairs) of the parents and the offspring is the output of the greedy construction procedure.

- *Fitness evaluation*: In the current implementation, the fitness of composite solutions is a measure of their absolute performance over the entire problem. It is defined as the number of training cases correctly covered in the case of a classification problem or the sum of squared error over the training set for the induction of continuous models.

- *Selection strategy*: The top 10% individuals with respect to the fitness are kept for the next generation.

Pairs $(Domain, Solution)$

- *Representation*: Each pair is represented by a pointer to a domain and a pointer to a solution.

- *Search operators*: Adaptation is implemented at that level. Given a pair *(Domain, Solution)*, the population of solutions is sampled and if a solution is found that has better accuracy over the domain than the current one, it replaces that solution. This adaptation strategy is performed at each generation and for every member of the population of pairs. No recombination operator has been defined. Instead, new offsprings are created by mutating an existing pair (i.e., by selecting a new domain or a new solution) selected according to a fitness proportionate rule.

- *Fitness evaluation*: The coverage-based heuristic is implemented at that level. Resource partitioning is exploited in order to implement this heuristic: each training example is seen as a resource which is shared among the pairs that cover it. The contribution of a training example to the fitness of a pair depends on the performance of the pair relatively to others on that particular domain of the input space. This strategy is implemented by defining a weight for each training data:

$$w_k = \frac{1}{\sum_{Pair_i:(D_i,S_i)} covered(k, D_i) \times F_{coop}(D_i, S_i)}$$

where: $covered(k, D_i)$ returns 1 if the domain D_i selects the k^{th} test case and returns 0 otherwise. A small value for w_k means that the k^{th} test case is covered accurately by a large number of individuals. Therefore, it is considered "easy" and a small credit is given to individuals that cover it. Conversely, a large value for w_k means that few individuals cover the k^{th} test case, resulting in a high payoff. $F_{coop}(D_i, S_i)$ represents the performance of the cooperation between a domain and a solution. That is, it is a function of S_i's accuracy over the domain D_i. The cooperation performance takes a value in the range: 0 (no cooperation, i.e. poor accuracy of S_i over D_i accuracy) to 1 (maximum cooperation, i.e. perfect match). Depending on the inductive learning problem under consideration, the definition of $F_{coop}(D_i, S_i)$ can take different forms. For applications to classification problems, the following definition for the measure of cooperation performance has been considered:

$$F_{coop}(D_i, S_i) = \exp(-\frac{\%\ mismatches\ S_i\ over\ D_i}{\tau})$$

For the induction of continuous models, this measure has been defined as follows:

$$F_{coop}(D_i, S_i) = \Gamma_\alpha(average\ squared\ error\ of\ S_i\ over\ D_i)$$

where:

$$\Gamma_\alpha(x) = 1 - \frac{1}{1 + exp(-\frac{x-\alpha}{\tau})}$$

Only one parameter, namely τ, is introduced in the definition of the measure of cooperation performance for classification problems. However, in the case of continuous models, a second parameter (named α) is introduced for the definition of this measure. Indeed, for classification problems, a boolean information is associated with each training example (the test case is classified correctly or not). However, for continuous models, the piece of information associated with each training example is a function of the error between the value returned by the model and the actual value. The underlying idea in the definition of $\Gamma_\alpha(x)$ is to perform a discretization operation in order to reduce the dependence of the measure of cooperation performance $F_{coop}(D_i, S_i)$ on the continuous value for the accuracy of S_i over D_i: if $x < \alpha$, $\Gamma_\alpha(x) \approx 1$ while $\Gamma_\alpha(x) \approx 0$ if $x > \alpha$. The purpose of the parameter α is to define a threshold to perform this discretization. During evolutionary search, the value of α is adapted to match the average squared error of the current best composite solution.

As discussed previously, some subjectivity is necessary in order to define a preference toward a particular strategy for controlling the automated decomposition of problems (and the induction of modular classification theories). In the case of MIL, this subjectivity is introduced in the definition of the measure of cooperation performance: $F_{coop}(D_i, S_i)$. Indeed, $F_{coop}(.)$ defines explicitly a total order over pairs (D_i, S_i). Therefore, it embeds the control of the balance between domain size and accuracy over domains.

Finally, the fitness of a pair is defined as:

$$fitness(D_i, S_i) = \sum_{\substack{t_k:\ test\ cases \\ covered\ by\ D_i}} w_k \times F_{coop}(D_i, S_i)$$

As a consequence of this definition, the contribution of a test case to the fitness of a pair is smaller if a large number of pairs cover that test case with good accuracy. The additive definition for $fitness(D_i, S_i)$ is also at the origin of the pressure toward larger domains. It gives an advantage to pairs that cover a larger number of training examples, providing the penalty introduced by the value of $F_{coop}(D_i, S_i)$ (due to a smaller accuracy of S_i over D_i) is compensated by the larger size of the domain.

- *Selection strategy*: The top 50% individuals with respect to the fitness are kept in the next generation along with the pairs that are used by the top 10% composite solutions (i.e., the composite solutions that reproduce).

Domains (respectively Solutions)

- *Representation*: The architecture of this system is not dependent on a particular representation to describe domains and solutions. For instance, S-expressions or neural networks can be used.

- *Search operators*: The definition of the search operators depends on the choice of the representation for domains (respectively solutions). Those operators construct offsprings from individuals selected according to a fitness proportionate rule.

- *Fitness evaluation*: The fitness of a domain (respectively solution) is the sum of the fitness of the pairs that have a pointer to that domain (respectively solution). Here, the idea is simply to give more attention in the search process to domains and solutions that are used by a large number of pairs and, indirectly, composite solutions.

- *Selection strategy*: The top 50% individuals according to the fitness are kept in the next generation along with the domains (respectively solutions) that are used by the pairs that reproduce.

Elitist Strategy

In order to maintain in the different populations all the elements which are used to define the best composite solutions or the best pairs with respect to their respective definition of the fitness, an elitist strategy is implemented. This strategy ensures that all the pairs that are used by the top 10% composite solutions are present in the next generation along with the domains and solutions that are used in the description of those pairs. In the same way, this strategy ensures that all the domains and solutions that are used by the top 50% pairs are also present in the next generation. This strategy avoids a collapse in the performance of composite solutions which would be observed if some of their components (pairs, domains or solutions) were to disappear. The elitist strategy helps to maintain the trade-off between the exploitation of the current best composite classification theories and the exploration of new decompositions.

6.2.7 Experiments in Classification: the Intertwined Spirals Problem

Presentation

The intertwined spirals problem is an inductive learning problem which consists in classifying points into two classes according to two intertwined spirals. Figure 6.14 shows an example of a perfect solution for that problem along with the two spirals used for training. In Juillé and Pollack [46], this problem has also been addressed using coevolution as a mean to introduce a coverage-based heuristic in the search. In that work, a competitive fitness was defined which resulted in the implementation of niching in the space of phenotypes (and, as a consequence of the mapping genotype-to-phenotype, niching also occurred in the space of genotypes). This niching technique allows the simultaneous exploration of different classification theories until one alternative eventually takes over the entire population because it covers all the test cases. This approach resulted in a significant improvement in the quality and performance of classification theories compared to an absolute definition of the fitness.

The induction strategy implemented in the MIL system uses a different approach.

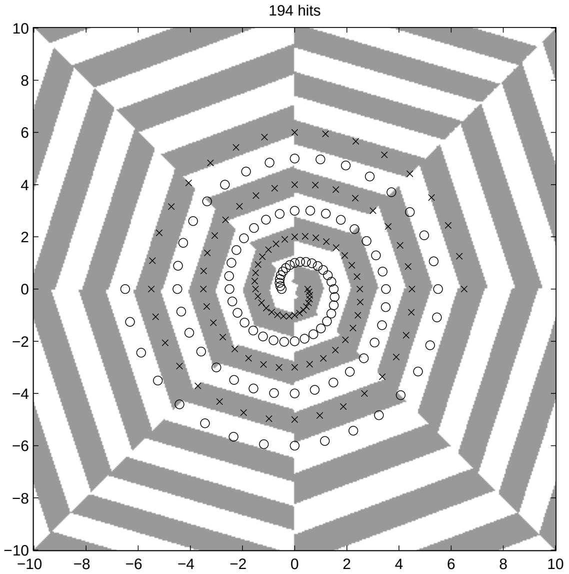


Figure 6.14: A perfect classification for the intertwined spirals exploiting a decomposition of the input space into four domains.

Indeed, in this former work, each evolving agent represents a complete solution and the mechanism for modularization is embedded in the representation language itself. The four-argument "`IFLTE`" instruction which implements the $if\ (\ldots \leq \ldots)\ then \ldots else \ldots$ statement is part of the primitive set and allows the description of modular concepts. On the contrary, in modular inductive learning, the modularization mechanism is implemented by the greedy procedure that constructs composite classification theories from pairs (Domain, Solution) and by the coverage-based strategy which controls the exploration of the space of pairs (and therefore modularity). Modularization is performed by decomposing the input space into domains which then become sub-problems. Therefore, in MIL, the credit assignment strategy that controls modularization is explicit whereas this strategy is implicit in [46].

Experimental Setup

We used the Genetic Programming [50] paradigm to address this problem. The set of terminals is composed of $\{X, Y, \Re\}$. They correspond respectively to vector coordinates in the input space and the ephemeral random constant. The sets of primitives to describe domains and solutions are different. This choice has been made arbitrarily with the idea that the goal of domains is to decompose the input space rather than providing a description for each class. Therefore, the language used to describe domains has a smaller expressive power than the one that describes solutions. For domains, the primitive set is composed of $\{+, -, \times, \%\}$, while for solutions the primitive set is composed of $\{+, -, \times, \%, \sin, \cos\}$. The population size is $1,000$ for domains and solutions, $2,000$ for pairs and 200 for composite solutions. The adaptation stage for pairs exploits a sample of solutions of size 50. For all the experiments described in this section, data is averaged over 100 runs.

Experimental Results

As shown in Figure 6.15, such a strategy allows the construction of a classification theory early in the run. Experiments were performed for different values of the parameter τ. This parameter controls the balance between the size of domains and the accuracy over domains. As described in Figure 6.15, if the pressure toward large domains is too large (*i.e.*, for large values of τ), a perfect classification theory cannot be constructed. In that case, the pressure toward accuracy is too weak and domains are allowed to expand quickly. Then, it becomes too difficult to discover a local model with high accuracy over those large domains. Figure 6.16 describes the evolution of the average number of components for the best composite solution at each generation. This figure shows that, initially a large number of components is required to cover accurately the entire input space. However, as the search goes on, accurate solutions over larger domains are

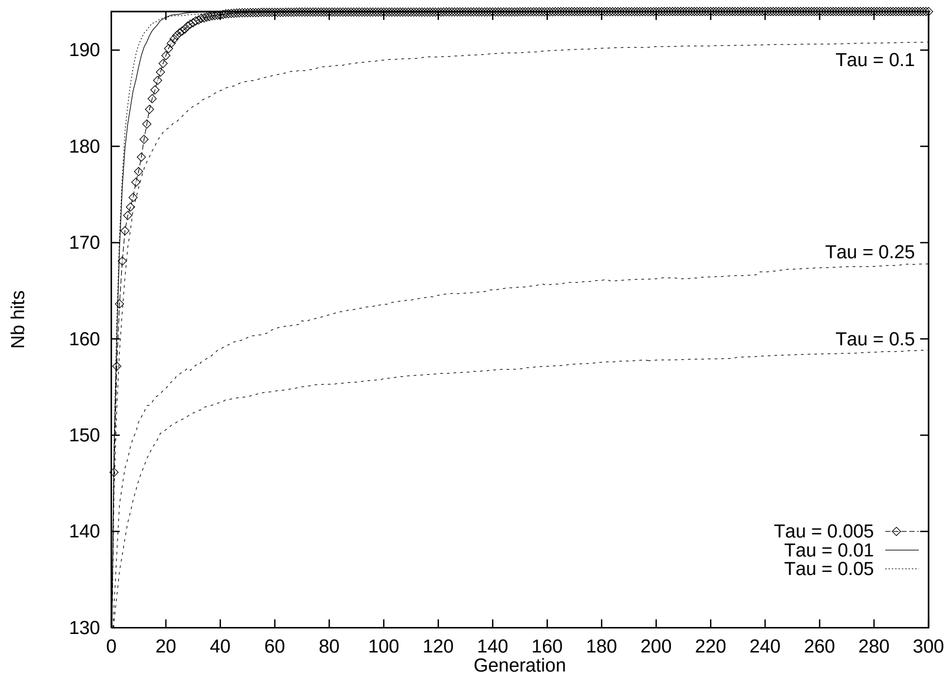


Figure 6.15: Evolution of average score for best composite solutions.

discovered and the number of components required to construct a classification theory decreases.

Another way to describe the dynamics of the search is to plot the evolution of the pair (number of hits, number of components in composite solution) with the generation number as the parametric variable. This trajectory is represented in Figure 6.17 for the same values of the parameter τ as in Figures 6.15 and 6.16, for the first 300 generations. Following Occam's razor, the goal of modular inductive learning is to construct a composite solution with few components which has high accuracy over the training data. Therefore, the goal is to reach the bottom-right corner of the diagram in Figure 6.17 or, more precisely, the point with coordinates $(194, 1)$, which corresponds to a monolithic perfect classification theory. Thus, Figure 6.17 highlights the fact that the trajectory for the pair (accuracy, model size) during evolutionary search plays an important role

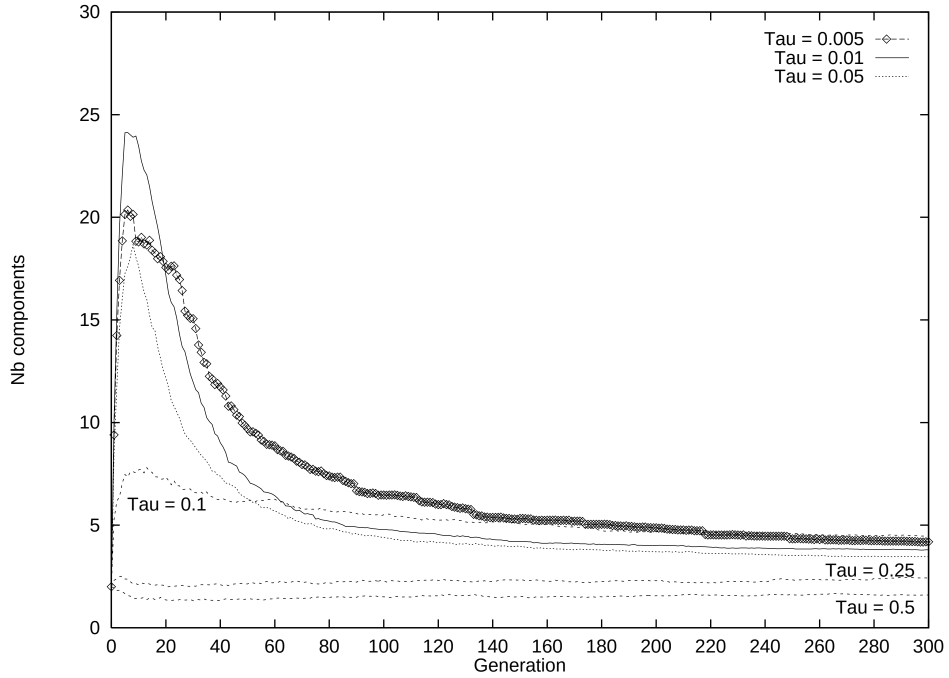


Figure 6.16: Evolution of average number of components for best composite solutions.

to determine whether this target area can be reached. In particular, in the case of this experimental setup, it appears that it is necessary to go through a stage where a large number of components are involved in the best composite solutions before constructing compact solutions with high accuracy.

From those experiments, the following observations can be made with respect to the effect of the parameter τ on the dynamics of search. First, if τ is large, there is more pressure toward larger domains. This results in classification theories with poor accuracy over the entire input space because the population of local models is exposed to arbitrary domains of large size for which accurate models are difficult to infer. On the contrary, if τ is small, there is more pressure toward accuracy. In that case, domains size can increase only if there exists some local models with good accuracy over those larger domains. This strategy usually results in composite solutions with a large number of components

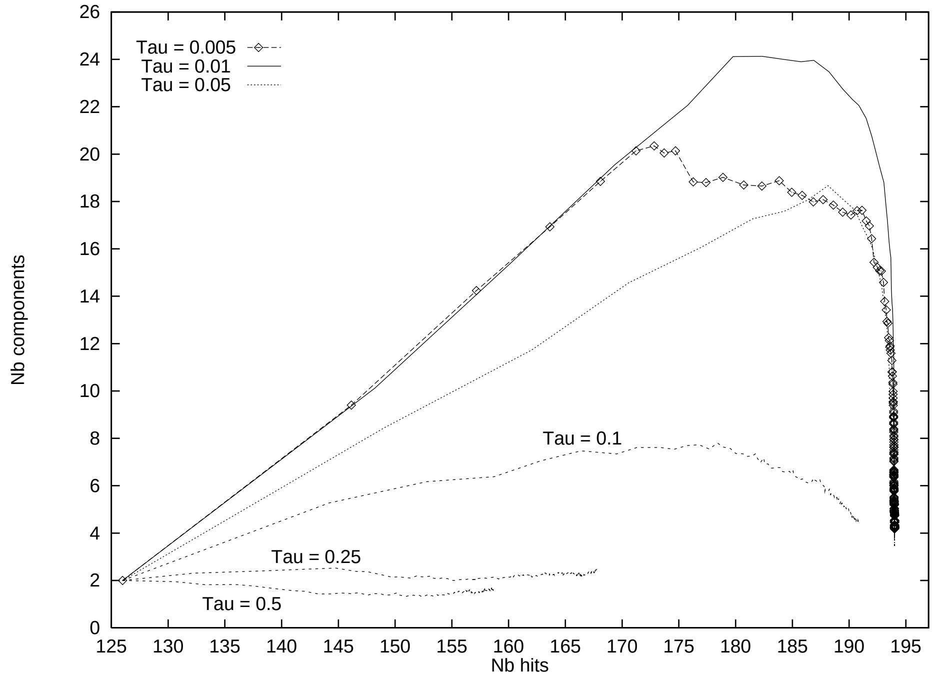


Figure 6.17: Trajectory of the pair (accuracy, model size) during evolutionary search.

because local models tend to specialize on small domains, thereby preventing them from generalizing on larger domains. For small values of τ, it also takes more time to construct a classification theory that covers the entire input space because more components are necessary.

6.2.8 Experiments in Time Series Prediction: the Wolfe Sunspots Database

Presentation

The *Wolfe sunspot data* is a record of the average number of sunspots observed each month since the year 1700. The curve of the average number of sunspots per year for the time period 1700 to 1995 is shown in Figure 6.18. The goal is to construct a model for this data set such that some prediction can be made for the number of sunspots for

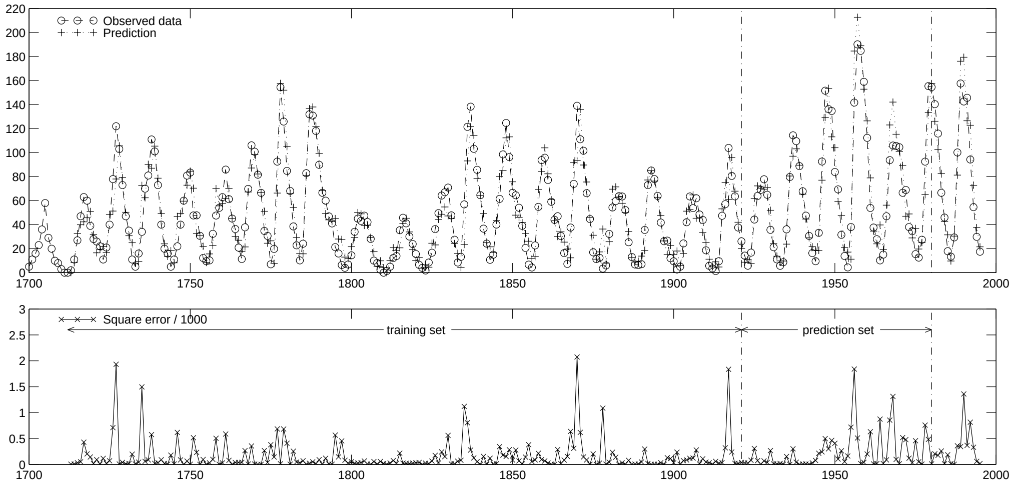


Figure 6.18: Above: the Wolfe sunspots data: average number of sunspots per year and prediction for run number four. Below: squared error of the prediction for that same run.

years beyond the observation period.

For that particular problem, over-fitting is an important issue, especially because the training data is noisy and sparse. Two basic approaches have been explored in the literature for addressing this issue. The first one is cross-validation. This technique divides the set into a training set and a validation set. Learning is performed from the training set and the validation set is used to detect over-fitting. The second approach uses a measure of complexity of the model. This measure is usually an estimate of the size of the model, which can be given by the number of free parameters in the case of neural networks [110] or the size of an S-expression [44].

Following Weigend, Huberman and Rumelhart [110], a normalized squared error is considered for the experiments presented in this section to evaluate the performance of a model on the training set and its prediction ability for input vectors outside the training set:

$$E_S = \frac{1}{\sigma^2} \cdot \frac{1}{|S|} \cdot \sum_{i \in S} (x_i - \hat{x}_i)^2$$

The purpose of the division by the variance ($\sigma^2 = 1535$) is to remove the dependence on the dynamic range of the data. Then, the average is taken to make the measure independent of the size of the data set S.

The nature of time series prediction problems is very different from classification problems illustrated in section 6.2.7 with the intertwined spirals. Indeed, the evaluation of the model against an input vector doesn't return a boolean answer but a value which is a function of the difference between the model's prediction and the actual value. Therefore, some specific mechanisms must be introduced for evaluating the performance of pairs (domain, local model). In the current implementation of the MIL system, this issue has been addressed by introducing the threshold function $\Gamma_\alpha(x)$ in the definition of the fitness of pairs. This function has been described in detail in section 6.2.6.

For the induction of continuous models, coevolutionary learning provides an environment in which the evolutionary race between local models forces the emergence of prediction models that have a better ability to generalize. The appropriate balance accuracy vs. number of components in composite solutions is determined by controlling the dynamics of coevolution between local models and domains. If local models over-fit the data, then composite solutions are composed of a large number of components and, therefore, are poor predictor. On the contrary, if the pressure toward larger domain is too strong, a single-component composite solution is discovered early in the evolutionary search but it has poor accuracy and, therefore, poor generalization ability. The idea is to capture the region for the value of the parameters of the MIL system for which the dynamics of the search is at the edge between those two behaviors. Our conjecture is that if parameters are in this region, solutions will be discovered that generalize better for input vectors outside the training set.

A specific search strategy has been designed in order to implement this idea. This strategy is composed of two stages. In a first stage, coevolution is set up such that priority is given to accuracy. The purpose of this stage is to discover a variety of local models that are defined over different sub-domains of the input space. At each generation, the value of the parameter α is set to the average squared error for the best composite solution discovered so far. As a result of this strategy, there is always a pressure toward the discovery of local models that have better accuracy. In the second stage, the evolutionary race is implemented among the local models. This is done by relaxing the pressure toward accuracy, allowing the emergence of larger domains. In practice, this is done by keeping α constant in the definition of $F_{coop}(D_i, S_i)$ for the remaining of the run. As a result, only local models with better generalization ability will survive. The run stops when a single component (monolithic) model is discovered. If no such model emerge in the evolutionary race after a fixed number of generation and only multi-component composite solutions are constructed then it is considered that the search failed to discover some relevant features to induce a monolithic model. This means that the local models are too specialized over subsets of the training data and those runs are discarded.

A criterion is defined to determine when the system switches from the first stage to the second. This criterion is given by the number of components in the current best composite solution. When this number is larger than the number of training data divided by a fixed constant β ($\beta = 5$ in our experiments), the second stage begins. It the second stage is not entered after a fixed number of generations, this means that the population of domains failed to capture the regions of the input space for which the models discovered by the population of solutions have a high accuracy. Therefore, those runs are also discarded.

Experimental Setup

Those experiments also use Genetic Programming to search for prediction models. The set of terminals is composed of $\{x_1, \ldots, x_{12}, \Re\}$. $\{x_1, \ldots, x_{12}\}$ constitutes the lag space and corresponds to the 12 past input values from the series. $\Re$ is the ephemeral random constant. The sets of primitives for domains and solutions are the same as for the intertwined spirals problem. There is no particular reason for the choice of those terminals and a different choice might result in a better performance. The population size is $1,000$ for domains and solutions, $2,000$ for pairs and 200 for composite solutions. The adaptation stage for pairs is based on a sample of solutions of size 50.

Experimental Results

We experimentally determined a range of values for which the dynamics of the coevolutionary search is close to the threshold for the emergence of single component composite solutions. The results for ten runs are presented in Table 6.4. This table describes at which generation the system switches from stage one to the second stage and at which generation a single-component solution is discovered. For two runs, the system never entered the second stage (runs 5 and 9). For two other runs, no monolithic model has been discovered (runs 1 and 10). For the other runs, monolithic models have been discovered whose predictive performance is comparable to the best known (see Table 6.5) in three cases (runs 2, 3 and 4).

Figure 6.19 describes the evolution of the number of components in the current best composite solution and the normalized squared error over the training set and the two prediction sets (between years 1921-1955 and years 1956-1979) for one of those runs (namely, run 4). Three phases can be identified. The first one corresponds to the initial stage during which local models are constructed. During that stage, priority is given to accuracy and the value of $E_{training}$ decreases. As a result, the number of

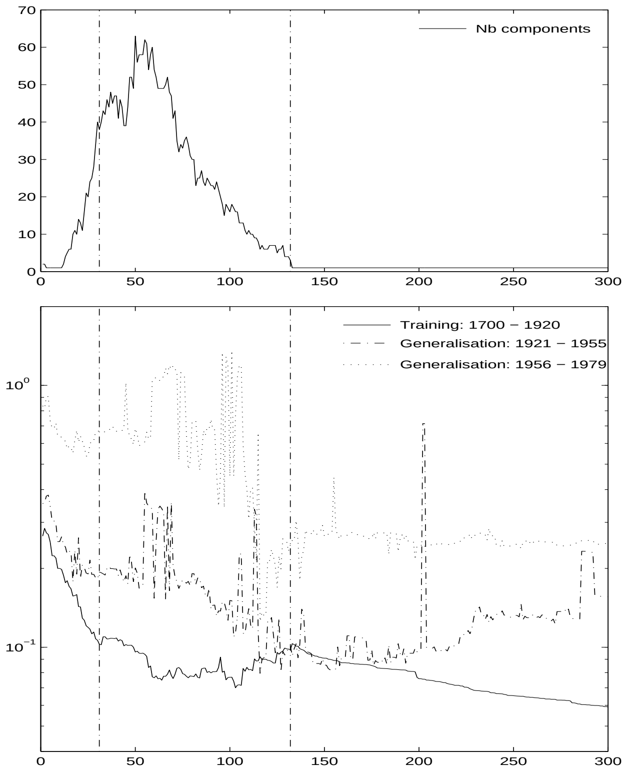


Figure 6.19: Results for run number four: Evolution of the number of components in the best composite solution and of the normalized squared error over the training and prediction sets.

Run	Stage 2 (generation #)	Single component (generation #)	Training error 1700-1920	Generalization $E_{predict(1921-1955)}$	Generalization $E_{predict(1956-1979)}$
1	32	3 components	0.067292	0.139104	0.335573
2	28	261	0.085308	0.120646	0.209681
3	25	135	0.102575	0.138104	0.156038
4	32	131	0.103196	0.092878	0.255542
5	—	—	—	—	—
6	40	276	0.085170	0.128788	0.543431
7	45	304	0.096550	0.153255	0.336291
8	20	69	0.127450	0.128760	0.292228
9	—	—	—	—	—
10	32	12 components	0.061501	0.206863	0.424998

Table 6.4: Description of experimental results for 10 runs with the Wolfe Sunspots database.

Model	Training 1700-1920	Generalization $E_{predict(1921-1955)}$	Generalization $E_{predict(1956-1979)}$
Tong and Lim [104]	0.097	0.097	0.28
Weigend et al. [110]	0.082	0.086	0.35

Table 6.5: Related experimental results in the literature.

components in composite solutions increases. Then, starting with generation 32, the evolutionary race takes place among the different local models. The pressure toward larger domains results in the emergence of local models which generalize better and, therefore, composite solutions with a decreasing number of components. At the same time, adaptation continues to improve the performance of local models. In particular, at the begin of the second stage, adaptation still seems to be the main force in action and the number of components in the composite solutions continues to increase. Once some local models are discovered that can generalize over larger domains, the number of components decreases. Eventually, a single-component solution is discovered. In that run, this happens at generation 131. After that generation, adaptation continues and the value of $E_{training}$ decreases. Over-fitting is observed and the prediction error is increasing (especially for $E_{predict(1921-1955)}$ in that run). In this experiment, big variations can be observed for the normalized squared error over the prediction set (for instance around generation 200 for $E_{predict(1921-1955)}$). This feature is a drawback of the representation scheme and the search operators implemented in Genetic Programming. Indeed, in GP a single mutation can modify considerably the behavior of a model. Therefore, S-expressions may not be the best representation for that problem, even if it results in good performance for a significant number of experiments. However, the architecture of the MIL system does not depend on a particular representation language or on some specific search operators. The implementation described in this chapter can easily be extended to embed some other representation languages like neural networks.

6.2.9 Concluding Remarks

This section describes the MIL system which proposes a framework exploiting the "Ideal" trainer paradigm in order to address problems in inductive learning. The important issues for the automatic decomposition of problems and the discovery of the corresponding components have been discussed. In particular, it has been concluded that some explicit bias has to be introduced in order to control the balance between accuracy of local models and size of domains. Depending on that bias, the resulting dynamics for search may vary significantly and, as demonstrated with the intertwined spiral problem, the quality of the final classification theory may also vary significantly.

The MIL system has been illustrated with two different applications: a classification problem and a time-series prediction problem. We believe that the central idea which consists in implementing an evolutionary race between local models is fundamental for the discovery of solutions with high generalization ability. The two applications presented in this section illustrate this strategy and demonstrate the relevance of coevolutionary learning for addressing the issue of generalization. In particular, they demonstrate that the control of the dynamics of coevolution is critical for the discovery of high quality models. Those two applications also validate the central strategy underlying the notion of the "Ideal" trainer. That is, by proposing problems of increasing difficulty to a population of learning agents, agents are likely to capture some intrinsic properties of the problem domain, allowing them to generalize to unseen problems.

Chapter 7

Conclusion

7.1 Summary

The motivation for the research work presented in this dissertation takes ground from a fundamental result in the field of search algorithms which states that there is no such thing as a "universal" search procedure. That is, any search algorithm embeds some mechanisms that make it more appropriate to capture some specific regularities or structural properties. Therefore, when addressing a new problem, one should always make sure the underlying regularities of the problem domain correlate with those the search algorithm can capture.

For many ill-structured problems, the identification and formalization of the intrinsic structural properties of the domain in the framework of heuristic search paradigms is not always possible. In fact, experience seems to show some evidence that evolutionary computation techniques may be an appropriate framework to address such problems. Even if the underlying search strategies embedded in evolutionary algorithms are not completely understood, their flexibility and robustness usually result in reasonable performance.

In this dissertation, EC techniques are described from the perspective of statistical inference procedures. It is observed that the principle of operation of EAs is based on a knowledge-gathering process resulting from the sampling of the search space. However, the structure that supports this process is an integral part of the EA and can not be dissociated from it. As a result, EAs don't always provide an appropriate framework to exploit certain kind of regularities.

The motivation of the research work presented in this dissertation is to extend the EC paradigm by proposing new methodologies that are based on the same principles of operation but expand their domain of applications. More precisely, the idea of this work is to make more explicit the structure that underlies the statistical inference process. Doing so results in more flexibility for the definition of this structure, providing a more appropriate framework to capture the regularities of a problem domain.

Two specific methodologies have been identified for the definition of such structures: the *explicit partitioning* methodology and the *indirect partitioning* methodology. Then, new strategies have been introduced, based on the exploitation of specific statistical properties that can be captured in the framework of those methodologies. The technical contribution of this dissertation is based on the description of algorithms that implement those strategies.

In the first algorithm, named SAGE, the structure that supports the statistical inference process is based on the explicit partitioning methodology. SAGE is innovative in the sense that sampling techniques haven't been proposed before as a strategy to search trees and directed acyclic graphs. This search algorithm has been implemented on a variety of distributed architectures, SIMD and MIMD, and achieved impressive results on difficult problems. In particular, SAGE has been able to discover some new constructions for sorting networks with fewer comparators than the previously known best constructions. Also, SAGE successfully tackled some difficult problems in DFA induction, ending up as a co-winner in the Abbadingo competition [54].

The indirect partitioning methodology has been exploited as a framework to implement a strategy which is based on the search for domains of the state space over which continuous progress can be observed with respect to a predetermined search procedure. Coevolution has been proposed as a paradigm to implement this strategy. Indeed, since coevolution is based on the evaluation of evolving agents in a dynamic environment, it provides an appropriate framework to capture such ability for continuous improvement. In the EC community, coevolution has also been proposed as a potential solution for achieving open-ended evolution, that is the emergence of agents of increasing complexity. The term *coevolutionary learning* denotes the application of this notion to the field of machine learning.

However, multiple impediments may prevent coevolution from achieving continuous progress. Those impediments have been illustrated in chapter 4 and discussed in section 5.1. As a result of this preliminary work, two fundamental conditions have been identified in order to observe continuous progress: the need to maintain useful feedback from the training environment and the need for a meta-level strategy to ensure progress in the long term. The "Ideal" trainer concept has been introduced as a paradigm that implements those two requirements. The central idea of this paradigm is to maintain an evolutionary race among the evolving agents and to force the evolution of the training environment toward problems of increasing difficulty.

The performance of this paradigm has been illustrated with the discovery of new cellular automata rules to implement the majority classification task that significantly outperform the previous best known rules for that task. Then, the concept of the "Ideal" trainer has been extended to the design of a system named Modular Inductive Learning (MIL). MIL exploits the idea of an evolutionary race among a population of local models to accurately cover large domains of the input space. This system has been illustrated with a classification problem and a time-series prediction problem.

7.2 Contributions and Discussion

Beyond the significant experimental results that have been achieved by applying the different techniques introduced in this dissertation, this research work also contributes to the understanding of fundamental issues in evolutionary computation.

First, by describing EC techniques as tools for performing statistical inference, some fundamental limits have been identified with respect to the class of problems that may be addressed successfully by these techniques. In particular, *sequential construction* problems introduced in section 2.1.2 exhibit some properties, like *epistasis*, that make them extremely difficult for an evolutionary algorithm like GAs.

Second, an extensive analysis of the coevolutionary paradigm supported by multiple experiments have made clear that some specific mechanisms must be introduced in order to allow the open-ended emergence of agents of increasing performance. In this research work, those mechanisms have been made explicit in the form of a meta-level strategy that controls the direction of evolution for the training environment. However, we believe that the "absolute" reference provided by the meta-level strategy can be approximated with some simple strategies that would be embedded in evolving agents themselves. This dissertation also demonstrates the importance of the dynamics of co-evolution for achieving continuous progress. In particular, maintaining an appropriate balance between problems difficulty and agents performance is critical. If problems difficulty don't increase fast enough, evolving agents tend to specialize and may not be able to improve when they are exposed to more challenging problems. On the other hand, if problems difficulty increase too quickly, agents get little feedback and are unlikely to progress.

Third, in chapter 5, *adaptability* has been presented as a fundamental condition for continuous progress. In the EC community, adaptability is also referred to as *evolvability*. Evolvability has been the object of a lot of attention recently [3, 106]. Indeed, evolutionary algorithms appear to be quite limited to address issues like scalability. In particular, defining a structure supporting the statistical inference process is not a sufficient condition for scalability. This has been illustrated in chapter 3 with SAGE where it was concluded that scalability could be achieved only by introducing more information about the problem domain as the search space grows.

From a machine learning point of view, adaptability means that some intrinsic properties about the training environment have been identified, allowing an agent to generalize its performance to unseen problems. Therefore, exposing learning agents to problem instances of increasing difficulty may allow them to capture relevant information for addressing the issue of scalability. The research work presented in this dissertation supports that idea and proposes the "Ideal" trainer as a paradigm to implement successfully this strategy.

7.3 Future Research

The research work presented in this dissertation is the source of several open questions and challenges for the future. In particular, some important issues that could be addressed are the following:

- In the conclusion of chapter 3, the introduction of problem-specific heuristics is proposed as a strategy for SAGE to address the issue of scalability. This idea originated from the experiments in DFA induction for which the search space grows exponentially as the size of target DFAs increases. However, introducing heuristics also reshapes the probability distribution of solutions in the search space. As a result, the sampling-based strategy may no longer capture some relevant information to drive search. The analysis of the interactions and the trade-off between knowledge-based search (i.e., heuristic) and statistical inference in the context of SAGE is a natural extension of this research work.

- As discussed in the previous section, the implementation of the "Ideal" trainer concept presented in this dissertation relies on some explicit strategies in order to maintain pressure toward adaptability and to provide a direction for the evolution of the training environment. This last strategy is based on the explicit definition of a partial order over the space of problems. However, this methodology assumes that the space of problems has been identified and that any element of arbitrary "difficulty" can be accessed. For many problems, this methodology cannot be applied directly. For instance, consider a model of coevolution between agents that implement a game strategy (to play chess or Go for instance). In that case, it is not possible to construct a game strategy of arbitrary difficulty. Otherwise, this would mean that the problem of discovering the best game strategy has been solved. Instead, some mechanisms must be implemented in order to approximate the dynamics associated with the "Ideal" trainer concept. Rosin [93] proposed several heuristics that extend the coevolutionary paradigm in order to address this type of issue. Such heuristics are based on the construction of structures, like the "hall of fame", that encode a sample of important solutions discovered in previous generations in order to provide a gradient for search. Designing such heuristics would make a very significant addition to the implementation described in this dissertation by providing more flexibility for addressing new classes of problems.

- The MIL system has been introduced as an application of the "Ideal" trainer concept for addressing problems in inductive learning (see section 6.2). A natural extension of this system concerns the embedding of multiple representation languages for the description of domains and solutions. Indeed, in such a "cultural" system [1], the evolutionary race would also involve a competition between the different representation schemes. Any representation language is more appropriate to describe concisely some specific structures or regularities. Therefore, this cultural environment would favor the emergence of those languages that capture more efficiently and reliably the regularities of the current problem. Such a system may eventually result in the construction of composite solutions that combine multiple representation schemes in their description, thereby taking advantage of the properties associated with each language.

- The representation problem is a recurrent issue in the field of artificial intelligence. In the field of evolutionary computation, a lot of effort is also dedicated to the design of an appropriate representation and of relevant search operators when addressing a particular problem by exploiting some specific knowledge about the problem domain. Indeed, the performance of any search procedure depends on the existence of a correlation between the properties of the landscape associated with the state space and the type of information that can be captured and exploited by the algorithm. This is a direct interpretation of the No Free Lunch theorem [114]. Therefore, as a follow up of the work on coevolutionary learning and the "Ideal" trainer concept discussed in this thesis, an important domain of research concerns the exploration of mechanisms that should be implemented in the representation language (and the search operators) in order to improve adaptability of evolving agents. Wagner and Altenberg [3, 106] addressed this issue and presented *modularity* as one important feature discovered by natural evolution in order to boost the evolvability of living organisms. Some other mechanisms like morphogenesis, learning or scalability may also be some relevant candidates that should be introduced in the representation language in order to help evolvability.

- Different methodologies have been explored for the design of systems exhibiting complex behaviors. Those methodologies are usually based on a decomposition into multiple interacting low-level behaviors. The purpose of this decomposition is to reduce the complexity of designing a monolithic solution to the task by looking for some nearly independent features. However, the definition of those low-level behaviors and the construction of an architecture that controls their interactions is a task of increasing difficulty as the system becomes more complex. The subsumption architecture proposed by Brooks [12] is one methodology which addresses this issue. However, there are some limits for the complexity of systems based on that architecture: as the number of layers in the architecture increases, it becomes more and more difficult to specify their interactions. Following a similar strategy, Albus [2] proposed a unified framework, the Real-time Control System (RCS), that ties together many components of the mind that have been identified — like perception, behavior generation, knowledge representation, planning, learning, ... — into an artificial intelligent agent. Albus' idea is that the goal of designing a topology which defines the interactions between the different component of a complex system can be achieve more efficiently by following some systematic design principles. However, such knowledge-based methodologies present a bottleneck for the design of highly complex systems because they still rely on the identification of low-level components and, especially, on the definition of a strategy to control how those components interact. A lot of applications that involve the processing of a large amount of information at different degrees of granularity and different levels of abstraction like vision systems, robot controllers or planning in complex environments (e.g. game strategies) don't seem amenable to implement such methodologies. For the design of such systems, search-intensive methodologies taking advantage of recent improvements in massively parallel technologies may offer an attractive alternative and/or addition to engineering-based approaches. In this dissertation, the problem of generating cellular automata rules to implement the majority classification task has been successfully addressed. This classification task requires the computation of a global property of the system based on the processing of local information. While being very simple, this example still identifies some important issues and it demonstrates that the methodologies exploited to address this problem, in particular the "Ideal" trainer concept, may be extended to tackle the task of designing large complex systems. Therefore, the study of hybrid techniques combining knowledge-based and search-intensive approaches is certainly a promising methodology for the design of intelligent systems.

The debate whether complex cognitive tasks can be implemented in a digital machine is still open [17]. Among those who think this is possible, there is still a lot of disagreement about the amount of computational resource that would be required to achieve an "artificial intelligence" that would exhibit a behavior similar to natural systems. However, it is likely that the exploration of new methodologies for the design of large complex systems constitutes an important part of the answer to the challenge of AI.

Bibliography

[1] Myriam Z. Abramson and Lawrence Hunter. Classification using cultural co-evolution and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 249–254. MIT Press, 1996.

[2] James S. Albus. The engineering of mind. In Pattie Maes, Maja j. Mataric, Jean-Arcady Meyer, Jordan B. Pollack, and Stewart W. Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 23–32. MIT Press, 1996.

[3] Lee Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994.

[4] David Andre, Forrest H. Bennett III, and John R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 16–18, 1996. Nara, Japan.

[5] Peter J. Angeline. Two self-adaptive crossover operations for genetic programming. In Peter J. Angeline and Kenneth E. Kinnear, Jr., editors, *Advances in Genetic*

Programming II, chapter 5, pages 89–109. MIT Press, 1995.

[6] Peter J. Angeline and Jordan B. Pollack. Coevolving high-level representations. In C. Langton, editor, *Artificial Life III*, pages 55–71. Addison-Wesley: Reading MA, 1994.

[7] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, September 1983.

[8] Christopher G. Atkeson, Andrew W. Moore, and Stephan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[9] Robert Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, 1989.

[10] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–24, 1993.

[11] Eric B. Baum. On optimal game tree propagation for imperfect players. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[12] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[13] D. S. Broomhead and D. Lowe. Multivariate functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[14] M. S. Capcarrere, M. Sipper, and M. Tomassini. Two-state, r=1 cellular automaton that classifies density. *Physical Review Letters*, 77(24):4969–4971, December 1996.

[15] Gail Carpenter, Stephen Grossberg, Natalya Markuzon, John Reynolds, and David Rosen. Fuzzy artmap: A neural network architecture for incremental supervised

learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3:698–713, 1992.

[16] Pang C. Chen. Heuristic sampling: a method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, April 1992.

[17] Paul M. Churchland. *Matter and consciousness : a contemporary introduction to the philosophy of mind*. MIT Press, 1988.

[18] Dave Cliff and Geoffrey F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *The Third European Conference on Artificial Life*, pages 200–218. Springer-Verlag, 1995. LNCS 929.

[19] Dave Cliff and Geoffrey F. Miller. Co-evolution of pursuit and evasion ii: Simulation methods and results. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 506–515, Cambridge, Massachusetts, 1996. MIT Press.

[20] Joseph C. Culberson. On the futility of blind search. Technical report, University of Alberta, July 1996. TR 96-18.

[21] Paul James Darwen. *Coevolutionary Learning by Automatic Modularisation with Speciation*. PhD thesis, University of New South Wales, Australia, 1996.

[22] Rajarshi Das, Melanie Mitchell, and James P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel Problem Solving from Nature - PPSN III, LNCS 866*, pages 344–353. Springer-Verlag, 1994.

[23] S. Das and M. C. Mozer. A unified gradient-descent/clustering architecture for finite state machine induction. In *Neural Information Processing Systems*, volume 6, pages 19–26, 1994.

[24] Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

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

[26] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kauffman, 1990.

[27] Sevan G. Ficici and Jordan B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In Christoph Adami, Richard K. Belew, Hiroaki Kitano, and Charles E. Taylor, editors, *Proceedings of the Sixth International Conference on Artificial Life*, Cambridge, Massachusetts, 1998. MIT Press.

[28] Lawrence J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[29] M. L. Forcada and R. C. Carrasco. Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural Computation*, 7(5):923–930, 1995.

[30] Attilio Giordana and Filippo Neri. Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416, 1995.

[31] Attilio Giordana, Filippo Neri, Lorenza Saitta, and Marco Botta. Integrating multiple learning strategies in first order logics. *Machine Learning*, 27:209–240, June 1997.

[32] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[33] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[34] David E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[35] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 431–437. AAAI Press / MIT Press, 1998.

[36] Milton W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, c.1969.

[37] David Perry Greene and Stephen F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993.

[38] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Chris Langton et al., editors, *Artificial Life II*. Addison Wesley, 1992.

[39] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[40] Phil Husbands. Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In T. Fogarty, editor, *Proceedings of Evolutionary computing, AISB Workshop Selected Papers*, pages 150–165. Springer-Verlag, 1994. LNCS 865.

[41] Hitoshi Iba and Hugo de Garis. Extending genetic programming with recombinative guidance. In Peter J. Angeline and Kenneth E. Kinnear, Jr., editors, *Advances in Genetic Programming II*, chapter 4, pages 69–88. MIT Press, 1995.

[42] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 12, pages 265–284. MIT Press, 1994.

[43] Daniel H. Janzen. When is it coevolution? *Evolution*, 34(3):611–612, 1980.

[44] Harri Jäske. Prediction of sunspots by gp. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*, pages 79–87, 1996.

[45] D. Johnson. The np-completeness column: an on-going guide. *J. Algorithms*, 3:298, September 1982.

[46] Hugues Juillé and Jordan B. Pollack. Co-evolving intertwined spirals. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 461–468. MIT Press, 1996.

[47] Hugues Juillé and Jordan B. Pollack. Improved algorithm for sorting network design validation. Submitted to Mathematical Systems Theory, 1998.

[48] Donald E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison Wesley, 1973.

[49] Donald E. Knuth. Estimating the efficiency of backtracking programs. *Math. Comp.*, 29:121–136, 1975.

[50] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[51] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[52] Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

[53] Kevin J. Lang. Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52, 1992.

[54] Kevin J. Lang and Barak A. Pearlmutter. Abbadingo one: Dfa learning competition. http://abba–dingo.cs.unm.edu, 1997.

[55] Kevin J. Lang and Michael J. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Summer Schools*. Morgan Kaufmann, 1988.

[56] Samir W. Mahfoud. Niching methods for genetic algorithms. Technical report, University of Illinois at Urbana-Champaign, May 1995. IlliGAL Report No. 95001.

[57] Steven Minton, Mark D. Johnston, Philips Andrew B, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17–24, 1990.

[58] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[59] Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[60] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[61] David E. Moriarty and Risto Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3):195–209, 1995.

[62] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[63] David E. Moriarty and Risto Miikkulainen. Evolving obstacle avoidance behavior in a robot arm. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 468–475, Cambridge, Massachusetts, 1996. MIT Press.

[64] David E. Moriarty and Risto Miikkulainen. Hierarchical evolution of neural networks. Technical Report AI96-242, University of Texas, Austin, Texas, 1996.

[65] David Eric Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks.* PhD thesis, University of Texas at Austin, USA, 1997.

[66] Christopher K. Oei, David E. Goldberg, and Shau-Jin Chang. Tournament selection, niching, and the preservation of diversity. Technical Report IlliGAL No. 91011, University of Illinois at Urbana-Champaign, 1991.

[67] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3.* Morgan Kaufmann, 1995.

[68] Andrew J. Palay. *Searching with Probabilities.* Pitman, 1985.

[69] Ian Parberry. A computer-assisted optimal depth lower bound for nine-input sorting networks. *Mathematical Systems Theory*, 24:101–116, 1991.

[70] Jan Paredis. Co-evolutionary constraint satisfaction. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel Problem Solving from Nature - PPSN III, LNCS 866*, pages 46–55. Springer-Verlag, 1994.

[71] Jan Paredis. Steps towards co-evolutionary classification neural networks. In Brooks and Maes, editors, *Artificial Life IV*, pages 102–108. MIT Press, 1994.

[72] Jan Paredis. The symbiotic evolution of solutions and their representations. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 359–365. Morgan Kaufmann, 1995.

[73] Jan Paredis. Coevolutionary computation. *Artificial Life*, 1996. To appear.

[74] Jan Paredis. Coevolving cellular automata: Be aware of the red queen! In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 393–400. Morgan Kaufmann, 1997.

[75] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[76] Charles C. Peck and Atam P. Dhawan. Genetic algorithms as global random search methods: An alternative perspective. *Evolutionary Computation*, 3(1):39–80, 1995.

[77] Charles C. Peck III. *Analysis of Genetic Algorithms from a Global Random Search Method Perspective with Techniques for Algorithmic Improvement*. PhD thesis, University of Cincinnati, 1993.

[78] Riccardo Poli, William B. Langdon, and Una-May O'Reilly. Analysis of schema variance and short term extinction likelihoods. In John R. Koza, Wolfgang Banzhaf,

Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick L. Riolo, editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 284–292. Morgan Kaufmann, 1998.

[79] Jordan B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[80] Jordan B. Pollack, Alan Blair, and Mark Land. Coevolution of a backgammon player. In Chris Langton, editor, *Proceedings of Artificial Life V*. MIT Press, 1996.

[81] Mitchell A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, Fairfax, Virginia, 1997.

[82] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, editors, *Parallel Problem Solving from Nature - PPSN III, LNCS 866*, pages 249–257. Springer-Verlag, 1994.

[83] Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366–372, San Mateo, California, 1995. Morgan Kauffmann.

[84] Armand E. Prieditis. Machine discovery of effective admissible heuristics. *Machine Learning*, 12:117–141, 1993.

[85] G. Rawlins, editor. *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

[86] Craig W. Reynolds. Competition, coevolution, and the game of tag. In Brooks and Maes, editors, *Artificial Life IV*. MIT Press, 1994.

[87] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:445–471, 1978.

[88] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society*, 49(3):223–239, 1987.

[89] Justinian Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms.* PhD thesis, University of Rochester, Rochester, New York, USA, 1997.

[90] R. S. Rosenberg. *Simulation of genetic populations with biochemical properties.* PhD thesis, University of Michigan, 1967.

[91] Christopher D. Rosin and Richard K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, California, 1995. Morgan Kauffmann.

[92] Christopher D. Rosin and Richard K. Belew. A competitive approach to game learning. In *Proceedings of the Ninth Annual ACM Conference on Computational Learning Theory*, 1996.

[93] Christopher Darrell Rosin. *Coevolutionary Search Among Adversaries.* PhD thesis, University of California, San Diego, 1997.

[94] Jonathan Roughgarden. The theory of coevolution. In Douglas J. Futuyma and Montgomery Slatkin, editors, *Coevolution.* Sinauer Associates Inc, 1983.

[95] J. D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *The Second International Conference on Genetic Algorithms*, pages 36–40, 1987.

[96] Jürgen Schmidhuber. On learning how to learn learning strategies. Technical report, Fakultät für Informatik, Technische Universität München, November 1994. FKI-198-94.

[97] Jürgen Schmidhuber. Discovering solutions with low kolmogorov complexity and high generalization capability. In A. Prieditis amd S. Russell, editor, *Machine Learning: Proceedings of the twelfth International Conference*, pages 188–196. Morgan Kaufmann, 1995.

[98] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley, Chichester, UK, 1981.

[99] Karl Sims. Evolving 3d morphology and behavior by competition. In Brooks and Maes, editors, *Artificial Life IV*, pages 28–39. MIT Press, 1994.

[100] Moshe Sipper. Coevolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1994.

[101] Rok Sosic and Jun Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 22:22–24, 1991.

[102] Harold S. Stone and Janice M. Stone. Efficient search techniques - an empirical study of the n-queens problem. *IBM Journal Research Development*, 31:464–474, 1987.

[103] Gil Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[104] Howell Tong and K. S. Lim. Threshold autoregressive, limit cycles and cyclical data. *Journal of Royal Statistical Society. Series B*, 42:245, 1980.

[105] B. A. Trakhtenbrot and Ya M. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company, 1973.

[106] Günter P. Wagner and Lee Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, June 1996.

[107] C. Wallace and D. Boulton. An information measure for classification. *Computer Journal*, 11:185–194, 1968.

[108] C. Wallace and P. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society*, 49(3):240–265, 1987.

[109] R. L. Watrous and G. M. Kuhn. Induction of finite state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414, 1992.

[110] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193–209, 1990.

[111] L. D. Whitley, editor. *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, 1993.

[112] L. D. Whitley and M. D. Vose, editors. *Foundations of Genetic Algorithms 3*. Morgan Kaufmann, 1995.

[113] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, 1984. Second edition.

[114] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical report, Santa Fe Institute, July 1995. SFI-TR-95-02-010.

[115] Z. Zeng, R. M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1994.

[116] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[117] Anatoly A. Zhigljavsky. *Theory of Global Random Search*. Kluwer academic, 1991. volume 65 of Mathematics and Its Applications (Soviet Series).