The Shape of Evolutionary Search:

Discovering and Representing Search Space Structure

The work in this PhD thesis has been carried out at the Leiden Institute of Advanced Computer Science, Leiden University, under the auspices of the research school IPA (Institute for Programming research and Algorithmics). IPA Dissertation Series 2001-01.

The Shape of Evolutionary Search: Discovering and Representing Search Space Structure

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden
op gezag van de Rector Magnificus Dr. W.A. Wagenaar,
hoogleraar in de faculteit der Sociale Wetenschappen,
volgens besluit van het College voor Promoties
te verdedigen op donderdag 25 januari 2001
te klokke 14:15 uur

door

Márk Jelasity

geboren te Székesfehérvár, Hongarije
in 1973

Promotiecomissie:

promotor:	Prof. dr. J.N. Kok
	Prof. dr. A.E. Eiben (Vrije Universiteit Amsterdam)
Referent:	dr. ir. D. Thierens (Universiteit Utrecht)
Overige leden:	dr. J. Dombi (University of Szeged, Hungary)
	Prof. dr. G. Rozenberg
	Prof. dr. H.A.G. Wijshoff

Contents

Introduction

The central theme of the thesis is the exploration of the possibilities of characterizing and adaptively making use of the structure of the search spaces of global optimization problems. Instead of giving a pure technical introduction I will place the different parts of the dissertation in the wider context of my ideas about a present puzzle within machine learning and the field of metaheuristics.

The cheap and powerful computers changed the practice of engineering and science in many ways. We can do things that were absolutely impossible before electronic computers. In the course of the pre-computer age mathematicians could calculate only a limited number of steps of the available algorithms relatively slowly. Even in the golden age of artificial intelligence (around 1960) it was normal and accepted to publish surprisingly sophisticated problem solving algorithms that were tested only by "hand simulation", e.g. (Newell et al., 1960). Now it is possible to calculate very difficult functions and algorithms which opens a space never seen before. This effect is still getting stronger and stronger since the performance of scientific equipment grows according to an exponential curve and new networking technologies boost this performance even further. We have reached the level where it is possible e.g. to experiment with systems containing a huge number of components interacting with each other in an *arbitrary way* specified by the researcher. These include weather systems, sociological modeling and evolutionary dynamics. With this success of computer science new problems emerged as well. I am not referring to the well known open problems of e.g. complexity theory. The problems I mentioned are not so apparent and sometimes overlooked by researchers. These problems are connected to the *epistemology* of computer science. And not only computer science: all the fields that are connected to it or were created by it.

Technology is developing faster then theory, and this results in a very interesting situation. In physics the usual scenario is to develop theories first and then pick one based on experimental data. This is because doing mathematics for months is much cheaper then performing a single experiment in a cyclotron, or with the Hubble telescope. In computer science we can see a radically different picture. Doing experiments is much easier then developing theories, and besides of this, the scientists has to describe not only one world, but many, and the number of these "worlds" is increasing. According to this, when solving a problem, the practice is to experiment with lots of things, a toolbox that contains more and more algorithms produced by the researchers, until the solution is acceptable. Then, if they want to, the theorists can think about the reason why a particular solution did or did not work. And these theorists must face really deep problems.

To illustrate the above ideas, from now on I will focus on *search problems*, since the thesis

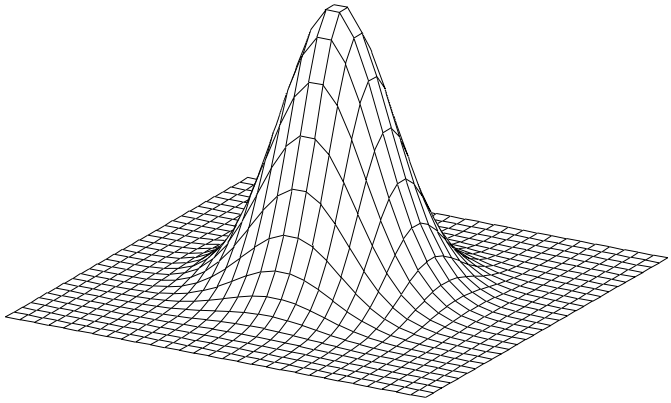


Figure 1: A simple surface for illustration.

belongs to this field. What is a search problem exactly? For us it will be sufficient to assume that in a search problem a *space* is given, and some kind of *evaluation function* over the space. The space is a set, e.g. a set of numbers or a set of timetables, or anything else. The problem is to find an element, that meets some criteria given with the help of the evaluation function. For instance we can search for elements with a high function value, or elements marked as "goal" by the evaluation function, etc. The exact nature of the entities is not important in this formalism. What is important is the *structure* of the space. The space is not only a set, it has structure, in fact, at least two structures. One is given by the evaluation function, and one is given by the actual algorithm applied for finding the solutions.

Fig. 1 helps to illustrate the idea. In this example the evaluation function is shown, where the space is the two dimensional plane of real vectors which form an Eucledian vector space. An important aspect of this structure is that it has a *distance* defined on it, namely the Eucledian distance. Most algorithms searching such vector spaces use this structure when moving around, e.g. when they want to generate a new candidate that is "close" to some other known vector, they mean this "closeness" in terms of the Eucledian distance. This notion of distance has the big advantage of being easily imaginable by humans. However, there is another structure, which is defined by the evaluation function. The elements having the same function value are equivalent according to this structure: the space has a special symmetry, where rings (in terms of the Eucledian space) with the highest element in their center play a distinguished role. We can think of it as another distance measure defined by the difference between the function values. Note that these two structures we assigned to the space are different in a topological sense.

So far, everything is clear, maybe even boring. The tasks that are tackled by computer scien-

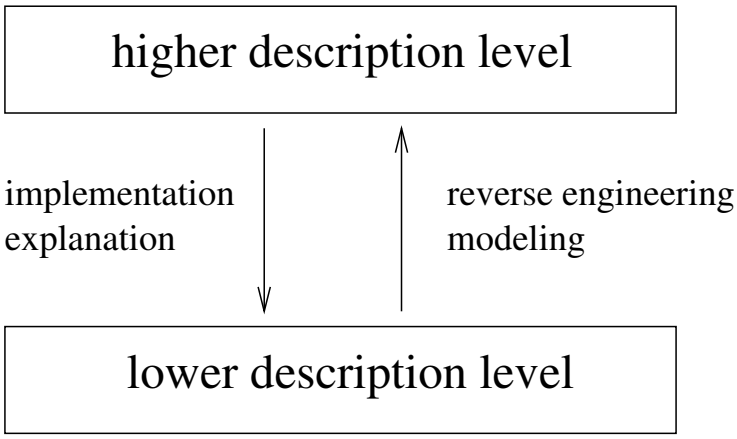


Figure 2: The lower level can have many models and a higher level can have many implementations.

tists day by day can have much more interesting properties, however. The root of the problems is that in computer science it is possible to handle abstract entities, spaces, which can have an arbitrary abstract structure, and which can be arbitrarily complex and large. The structure of the space is not known in general (and available knowledge may well be misleading). Neither the structure of the evaluation function, which is a black box in many cases, nor the structure defined by the search algorithm. Such problems form the main application domain of several search heuristics, such as evolutionary computation, simulated annealing, tabu search, ant colony optimization, etc. To be precise the structure is of course known in the sense that it is computable. For example the distance of every pair of points can be calculated w.r.t. to both kinds of structures mentioned earlier. What is missing is a *model* of this structure: the intuition and the insight.

Let us elaborate on this latest idea as this is in the very hart of the problem. The structure of our scientific knowledge about the world is not a single huge consistent theory. To tackle certain situations we need *description levels* which are built on top of each other. For instance we can talk about electricity — even in the context of causal relationships — without referring to its "implementation". In fact the term was invented and used successfully a long time before the invention of charged particles like electrons. The notion of description levels is generally used not only in science but also in engineering. For example a good object oriented design can be understood without a single implementation. Not to mention that the implementation does not even have to be object oriented. The possible relations between description levels are illustrated in Fig. 2. The reason of the existence of different description levels is that they are useful. In computational science higher level descriptions of the domains of search algorithms are usually missing.

An example could be finding a decision in a space of possible decisions, where the evaluation of a decision is done by a model, which simulates the effects of the decision in the real situation (which can be a power plant, for instance). The result of the simulation is a number scaling the quality of the decision. As earlier the space of possible decisions has at least two

kinds of structure. The first is defined by the search algorithm, which has to *represent* the decisions somehow, for example with an array of parameters, and then it has to explore the space generating new candidate solutions based on the existing ones. This mechanism implicitly defines a structure, since (if the algorithm is not pure random search) the newly generated solution will depend on the search history, so, loosely speaking, we have a sort of neighborhood relation. The effect of this structure is sometimes called the *bias* of the search algorithm (Mitchell, 1997). The other structure, as before, is defined by the evaluation function, which is a black box in this case.

The interesting part is that — unlike in the case of the simple example in Fig. 1 — it is impossible to find *words* (like *ring*) to describe the situation. The conceptual knowledge is missing, which is the epistemological problem: *we do not know — or in other words cannot talk about — the entities we work with*. In the history of science this situation is unique since the vocabularies of scientific theories are traditionally based on the vocabulary of everyday language. When dealing with such abstract spaces, we have no grounds to hope that the semantic hierarchy that describes it meaningfully can be grasped by human intelligence. This claim is not uncontroversial, a part of the thesis, especially Section 5.2, is devoted to arguing for it.

The question arises: is it at all worth the trouble to find such new terms, a vocabulary or even a semantic hierarchy for describing specific domains? We have to keep in mind that one of the main advantages of several metaheuristics is that they need very little domain knowledge and in a real world application there might not be enough time or money to develop such models. However if we ever want to *understand* these algorithms, their biases, the way they work, we have to address these questions. Furthermore, with the help of such a description much more effective search can be performed, since the two kinds of structures I mentioned above can be synchronized. For example the symmetry of the function in Fig. 1 can be exploited using special algorithms that work in a polar coordinate system. And it can be done only because we know the terms ring and symmetry and (which is not independent of these) polar coordinate system *and* we also know that in this case these are relevant. But we do not need to rely only on abstract examples. If we look around between the things created by evolution we can see a great amount of modeling. For example the frog's eye detects motion and size, which are key elements of the vocabulary for describing the frog's world (and the human world too). The success of the human race is related to the emergence of language, and one of the functions of language is to model the world we live in in a much more effective and flexible way than other animals do.

In the following the outline of the thesis is given. Though this introduction may suggest that this thesis is concerned mainly with philosophical matters, this is not at all true. The first half of the thesis contains the descriptions of two algorithms that were developed without any deep philosophical considerations. However, in historical perspective, the experiences I had during the process of this research suggested me that certain generally accepted approaches might be insufficient for the explanation of the performance of the algorithms on the problems I studied and this made me think about possible alternative solutions. The second half of the thesis contains the result of these efforts. The reactions to these attempts and my studies in linguistics and cognitive science helped me put this earlier work of mine in a wider perspective which is presented in this introduction.

The first chapter contains an introduction to the basic terms used throughout this disserta-

tion. The material of the thesis forms four chapters. Chapter 2 (Jelasity and Dombi, 1998) and 3 (Jelasity et al., 2001) describe global optimizers that also explore the structure of the local optima of an optimization problem. The first version is called GAS. This algorithm is able to make use of structural information adaptively using a niching technique based on a subpopulation approach simulating the formation of species. This means that the search space is divided to clusters (which we call species), and these cluster-structure is constantly modified according to the results of global search restricted to these clusters. The cluster structure reflects the distribution of local optima. The bias of the algorithm can be controlled explicitly, as the structure of the space (the distance function of the space) can be implemented arbitrarily, and the automatic parameter setting of some important parameters is possible based on this distance function and other user-given parameters. The output of the algorithm contains the structural information as well as the final solution. The second version is called UEGO. The difference from GAS is that the optimizer that works inside of a species can be any algorithm. This turns the approach into a general hybrid paradigm that can be applied along with any optimizer if the structure of the local optima is important or if stable solutions must be found. Another difference is that the structure of the algorithm was simplified to make parallel implementation easier. For results of the parallel implementation please consult (Ortigosa et al., 2001).

The sections of Chapter 4 are related to the way of description of the working of evolutionary algorithms. Section 4.1 (Jelasity and Dombi, 1996) shows that using a restricted vocabulary like it is done in some traditional theoretical approaches is insufficient, and suggests a way of expanding the framework. This work is closely related to the epistemological problems of computer science and shows that theoretical investigations based on some *a priory* fixed set of features of the space are doomed to failure. Section 4.2 (Jelasity, 1997) gives an alternative model with an emphasis on unknown and possibly important structural information. This work builds on the previous one. The difference is that all aspects of the old way of looking at the problem are rejected and it is suggested that the search should be described as a path through some feature space where these features are relevant to the search space at hand. Section 4.3 (Jelasity et al., 1999) offers a way of visualizing the structure of spaces and also provides a measure of problem difficulty. In this case the visualization is done based on the evaluation function *and* the algorithm at hand so it is a tool to look at the structure defined by the evaluation function from the viewpoint of the search algorithm.

Chapter 5 introduces a philosophical problem of the boundaries of human design where many of the interesting questions mentioned in this introduction are covered. In Section 5.2 (Jelasity, 1999) it is suggested that the way of looking at evolution in the field of evolutionary computation is very similar to the so called *adaptational stance*. It is also shown that — beside the similarities — there are significant differences as well, and these are connected to the epistemological crisis of computational science. The point is that since biologists have a natural vocabulary for talking about features of things (wing, eye, hart, etc.) in computer science the situation is radically different. To imagine this try to talk about an elephant referring only to its DNA sequence; well, in evolutionary computation people have no other information. I also propose that the problem is deep and we have no grounds to think that it will ever be solved by humans. Human intelligence was evolved in a specific environment for specific tasks. Though the emergence of language made it possible to reach high levels of abstraction, to really describe

such spaces and to make use of their structure the only way might be to do it automatically using artificial intelligence and machine learning where concepts that are completely un-natural for humans can be handled and evolved. Section 5.1 (Jelasity, 2000) offers an approach that makes this automatic process possible. A framework is presented in which binary encodings of a problem domain can be learned using ideas from machine learning. The actual learning procedure can be arbitrary. Using such an approach, it may be possible in the future to solve the reverse engineering problem in a similar way evolution solved it for us evolving "devices" such as the eye and human language. The later — besides its other functions — allows us to describe the human domain, i.e. the aspects of the world that are important for us. This makes is possible to plan our actions and to predict future events efficiently. In abstract domains a similar approach of evolving such representations that allow efficient search, information compression and even communication is becoming computationally feasible.

Part I

Evolutionary Computation Basics

Chapter 1

Evolutionary Computation Basics

The aim of this chapter is to give a short introduction to the field of evolutionary algorithms (EA). This introduction is not intended to be complete; its purpose is only to make the thesis self contained. Several good textbooks are available that describe the field in depth. Michalewicz's book is a classic and has several editions (Michalewicz, 1996). The "Handbook of Evolutionary Computation" contains very useful short and clear papers on the important aspects of the field (Bäck et al., 1997), and it is kept up to date constantly. Other useful references are (Mitchell, 1996; Bäck, 1996; Schwefel, 1995).

1.1 Evolutionary Algorithms and Heuristic Search

It is not an easy task to define evolutionary search since the boundaries of this area are fuzzy. Several terminologies exist that describe essentially the same set of methods, though the unification of these terminologies has already begun. These terminologies include not only the different flavors of EA but also completely separate fields like tabu search, simulated annealing, scatter search, path relinking, etc. The notations used are closest to the conventions of the field known as genetic algorithms (GA) and in Section 1.1.6 links will be given to some other interesting "languages".

1.1.1 Short History

In (Fogel, 1998) the reader can find examples of the earliest attempts to make use of the evolutionary analogy for solving computational problems. Some of these works were written soon after the structure of the DNA was discovered and the Darwinian principles gained a firm scientific foundation. Evolutionary computation was born together with artificial intelligence, machine learning, generative grammar, the theory of automata and cognitive science in the post-war era of the exceptional scientific excitement and optimism.

One of the independently emerging fields is *evolutionary programming* (EP). This method was introduced by Fogel, Owens and Walsh (Fogel et al., 1966) originally for developing finite state automata for solving specific problems. Another field was founded in Germany (Rechenberg, 1973). This algorithm is traditionally called *evolution strategies* (ES) and was used for

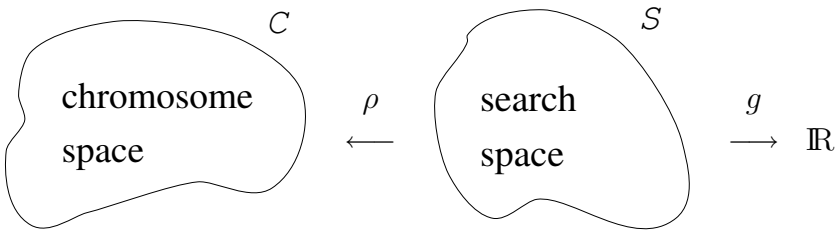


Figure 1.1: The structure of the optimization problem.

real parameter optimization (see (Bäck et al., 1991; Bäck, 1996) for a modern introduction). The third important field is genetic algorithms (GA) starting with John Holland's widely cited book (Holland, 1975). One of the main features of this approach is that it was designed to search binary spaces of the form $\{0,1\}^n$. The last field, *genetic programming* (GP) is relatively new (Koza, 1992; Koza, 1994). This family of algorithms is used for evolving computer programs for given tasks.

1.1.2 Basic Structure

The problem to be solved by the EA is formulated as shown in Fig. 1.1. We are given a search space S and an objective function g defined on it. The problem is to find the global maximum (or minimum) of g in S. Finding the optimal solution is *not* guaranteed; the EA performs heuristic search. To apply the EA we have to create the *coding function* or *representation* ρ that partially maps S to the finite *chromosome space* C. ρ has to be invertible to make it possible for the EA to calculate the objective function value of any element of C. On C *operators* of type $C^n \to C^m$ have to be defined. These operators will be used to generate new solutions from known ones.

Let us mention that defining the operators on S instead of C would have been sufficient so we could have ignored C altogether making the discussion simpler. However, many times it is more convenient and natural to change the representation than to change the operators from a practical point of view. Another reason for using C is that it is a convention in the field. In many subfields of EA the operators are fixed and to adapt the algorithm to a problem one has to find a good encoding.

Another special property of evolutionary optimization is that a *population* of solutions is maintained instead of a single solution as usual in most of the exact methods. A population is a multiset (or bag) over C (i.e. it may contain more than one instances of a given element). However, it is possible that the population contains only one element. The skeleton of the algorithm is shown in Fig. 1.2. In the simplest case the initial population P_0 may be generated by a uniform random sampling of C. A priori knowledge can be incorporated as well choosing "good" individuals. The exit criterion in line 3 may be based e.g. on a previously fixed number of objective function evaluations, the speed of convergence, the combination of the two, or any other convenient method. It should be noted here that the size of the starting population $|P_0|$ is a parameter of the EA, while $|P'_t|$, $|P''_t|$ and $|P_{t+1}|$ are determined by the implementations of

```
1 create the initial population P_0
2 t <- 0
3 while not Exit(P_t)
4       P'_t  := Selection( P_t )
5       P''_t := Recombine( P'_t )
6       P_{t+1} := New-pop( P_t, P''_t )
7       t <- t+1
```

Figure 1.2: The skeleton of the EA.

`Selection` and `New-pop`. The remaining parts are discussed in separate sections.

1.1.3 Representation

As already mentioned in Section 1.1.2, a representation function must be given if the EA is the method selected to solve the problem at hand. This is a rather problematic and sensitive problem. One of the main goals of the thesis is to emphasize the crucial role of the structure of the space in which the selected representation is a major component. Here, three ad hoc examples of fairly different types are given to let the reader form an impression of the process of choosing a coding function.

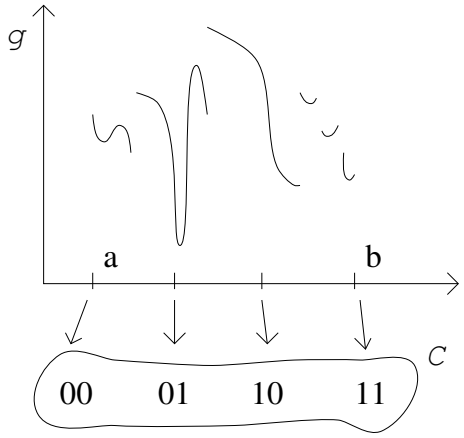
Real Functions

For real function optimization one natural choice is to use S directly. This approach is taken by ES operating directly on real numbers. Other approaches are possible, however, though some results indicate (Wright, 1991) that e.g. the working of the GA is very similar to ES in the case of the problem class discussed in this section. Let us take a look at the GA approach.

Let $S = [a, b] \subset \mathbb{R}$. To code this interval, it has to be discretized first (i.e. a finite number of points has to be selected from it). Let this discretization be equidistant and contain 2^n point for some $n \in \mathbb{N}$. Let us choose $C = \{0, 1\}^n$. Than, ρ can be defined simply by assigning a permutation of C to the increasing sequence of the selected points (see Fig. 1.3). Two special permutations are used generally; they are shown in Table 1.1 for $n = 4$. An empirical comparison of Gray and binary representations is given in (Caruana and Schaffer, 1988). It is generally believed that the advantage of Gray coding is that it preserves the adjacency relation of the coded points with respect to the Hamming distance (the number of bit differences), see Table 1.1.

Traveling Salesman Problem

In the most general case of the traveling salesman problem (TSP), an undirected, weighted graph $G(V, E)$ is given, and the search space is the set of *tours* i.e. circles that visit all of the points in V (the set of *towns*). The task is to find an element of S in which the weights sum up to the minimal value. We can assume that an arbitrary pair of towns is connected in G since, with



$$
\begin{aligned}
C &= \{0,1\}^2 \\
S &= [a,b] \\
\\
\rho(00) &= a \\
\rho(01) &= \frac{2a+b}{3} \\
\rho(10) &= \frac{a+2b}{3} \\
\rho(11) &= b
\end{aligned}
$$

Figure 1.3: An illustration of the binary coding of real domains.

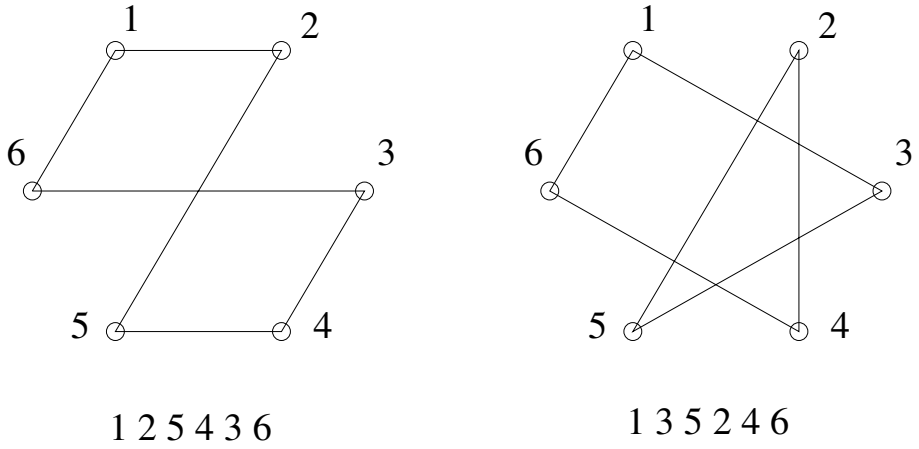


Figure 1.4: An illustration of the permutation encoding of the TSP.

properly chosen weights, any number of edges can be inserted without affecting the optimal solution(s).

Several representations of the TSP are discussed in the literature (Radcliffe and Surry, 1995). The most trivial one can be obtained as follows: let us label the towns with the set $\{1, 2, \ldots, |V|\} \subset \mathbb{N}$. There are $2|V|$ permutations of V that correspond to the same tour. From these, let us chose the one which starts at town 1 and goes on towards the town with the less label out of its two adjacent towns. Let ρ assign this permutation to the solution at hand (see Fig. 1.4).

Artificial Neural Networks

EAs are often used to optimize neural network structures and/or weights i.e. the learning is partially or totally controlled by the EA. If S is a set of possible artificial neural network (ANN)

number	Gray code	H.-dist.	Binary code	H.-dist.
0	0000		0000	
1	0001	1	0001	1
2	0011	1	0010	2
3	0010	1	0011	1
4	0110	1	0100	3
5	0111	1	0101	1
6	0101	1	0110	2
7	0100	1	0111	1
8	1100	1	1000	4
9	1101	1	1001	1
10	1111	1	1010	2
11	1110	1	1011	1
12	1010	1	1100	3
13	1011	1	1101	1
14	1001	1	1110	2
15	1000	1	1111	1

Table 1.1: Gray and binary codes of integers in [0,15] and the Hamming distance (number of bit differences) of the adjacent codewords for both case.

structures then a learning method (e.g. backpropagation) is necessary in order to measure the value of the given structure (i.e. every evaluation of g involves a whole learning process and than the performance of the tuned network is measured). If the weights are also optimized by the EA then the cost of evaluating an ANN is much less. The present example illustrates the way an ANN structure can be represented for the EA. It has to be noted that the original version of this representation (Tang et al., 1995) contained the weights of the ANN as well. The search space S is a class of feed-forward ANNs with 3 input and 1 output neurons, at most M hidden layers and at most 3 neurons in each layer. The representation contains layer control and neuron control bits. If the ith layer control bit is on, then the ith layer is present in the network. Than, the corresponding neuron control block is used to determine the structure of the layer. Figure 1.5 gives an example for $M = 4$. In this case, every individual contains 4 layer control and $4 \cdot 3 = 12$ neuron control bits so $C = \{0, 1\}^{16}$.

(Radcliffe, 1991b; Mandischer, 1993; Yao, 1993) are good references for the readers interested in ANN representations. (Thierens, 1998) offers a solution to handling the functional redundancy of many coding approaches (the problem of *competing conventions*). A promising direction of research is finding representations that code the process of creation of the network instead of the network itself. Such codings are more compact and may be more expressive in certain cases. (Lucas, 1995) is a step towards this direction.

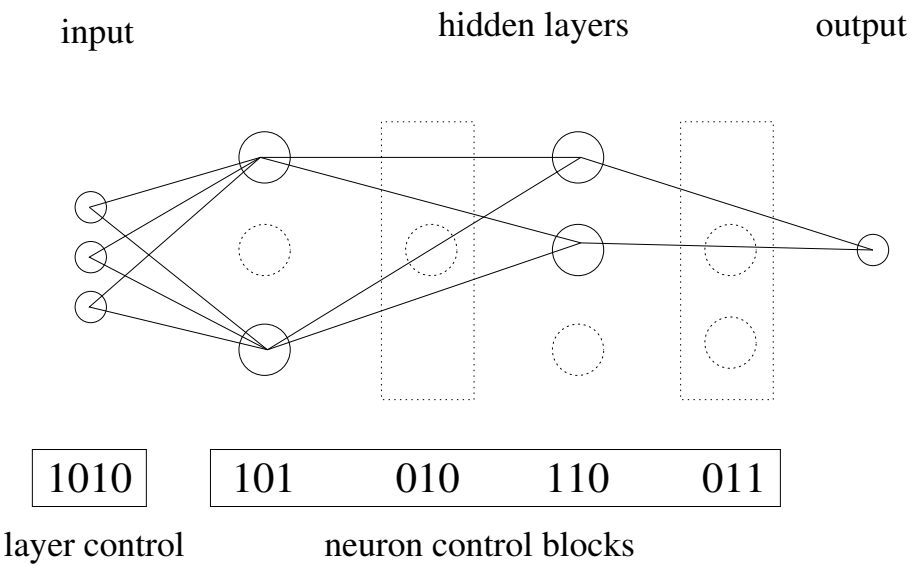


Figure 1.5: An illustrative example of ANN structure encodings.

1.1.4 Operators

Line 5 in Fig. 1.2 has a key role; this is the point where new solutions are generated using *operators* based on the old ones testing unseen areas of the search space. Genetic operators always have "parents" that are randomly drawn from P_t' and results in an "offspring" that is placed into P_t''. These operators are normally not deterministic, so the general formulation of the m-variable operator op for generating k offspring is

$$op \ : \ C^m \longrightarrow C^k.$$

The traditional operators are *crossover* and *mutation*. Crossover has two variables while mutation has one. Multi-parent recombination is also used (Eiben et al., 1994). It is possible to apply many operators one after another, e.g. first crossover, then mutation. Like in Section 1.1.3 a couple of examples are given here for illustration. Note that general, representation independent operators have also been defined (Radcliffe, 1994). An example of such operators will be given in Section 1.2.2.

Binary Representation

In the case of binary representations C has the form $C = \{0, 1\}$.

n-point crossover. See (Eshelman et al., 1989) and Fig. 1.6. This type of crossover has a parameter P_c (crossover probability). The operator is performed with a probability P_c. If the operation is not performed, one of the parents is returned as the offspring.

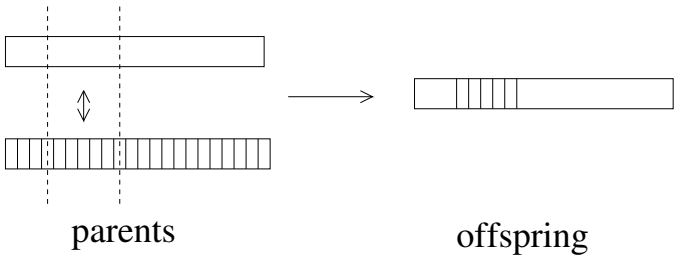


Figure 1.6: The n-point crossover operator for $n = 2$.

Uniform crossover. See (Syswerda, 1989). Here, the bits of the parents are swapped with a given probability P_c position by position. Thus, unlike n-point crossover, uniform crossover transmits bits independently.

Mutation. The bits are inverted with a probability P_m position by position. This kind of mutation is thus equivalent to a uniform crossover with the inverse individual and $P_c = P_m$. P_m is usually small (see (Bäck, 1993)) while P_c is generally close to 1. The operator is illustrated in Fig. 1.7.

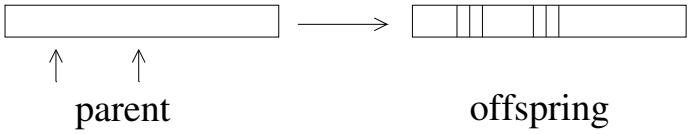


Figure 1.7: The traditional mutation operator.

Permutation Representation of the TSP

Several operators are known for this domain (Goldberg, 1989). All of them are very similar to those shown in the binary case; the only difference is that a repair mechanism is needed since in this case C is not in the form $C = \Sigma^n$.

Generalized n-point crossover. The operator is illustrated in Fig. 1.8 for $n = 2$. First we try to proceed as we did in the binary case. However, in every position, we check whether the town we wish to insert is compatible with those already accepted. Generally, after this first phase the child is incomplete. In the second phase we try to fill in the gaps using the other parent's genetic material in the corresponding positions. Usually, it is still not enough. In the final patching phase we complete the child using an arbitrary method.

Permutation mutation The order of two arbitrary chosen towns is reversed with a probability P_m.

```
parent 1     1  2  3 | 4  5  6 | 7  8
parent 2     1  5  4 | 3  8  7 | 2  6

phase 1      1  5  4   -  -  6   2  -
phase 2      1  5  4   3  8  6   2  -
phase 3      1  5  4   3  8  6   2  7
```

Figure 1.8: Example of generalized n point crossover.

1.1.5 Selection, New-pop

`Selection` and `New-pop` are closely related so it is reasonable to discuss them together under the name *selection methods*. The basic idea of every selection process is to increase the average performance of the population by selecting good individuals with a high and bad ones with a low probability. This is done by (implicitly or explicitly) assigning a probability distribution to the population. The probability of selecting an individual x under the given distribution is the *fitness* of x. Some of the selection methods use the objective function values of the individuals directly (i.e. for $x \in C$ they use $g(\rho^{-1}(x))$) while others use only the order information with respect to the function g. The methods of the first type are very sensitive to g. In this case, the *fitness function*[1] of the individuals has the form of a parametrized transformation

$$f(x) = T(g(\rho^{-1}(x)), \boldsymbol{p})$$

where $\boldsymbol{p}$ is a parameter (vector) that can be used to control the selection process during the search. Let $|P_t| = |P'_t| = |P''_t|$ and let `New-pop` be only the single assignment $P_{t+1} = P''_t$. Such selection methods are called *generational selections*. *Proportionate*, *ranking* and *tournament* selections are discussed. For more details the reader should refer to (Goldberg, 1991; Blickle and Thiele, 1995).

The special case when $|P'_t| = 2$ and $|P''_t| = 1$ is called GENITOR or *steady state* selection (Syswerda, 1991). Here, `New-pop` has to delete an individual of P_t and replace it with the single element of P''_t. Every selection method has an *elitist* version which means that the best individual of P_t is guaranteed to survive.

Proportionate selection. This method uses g directly but transformed in such a way that every element of P_t is positive. T is usually a linear transformation:

$$f(x) = T(g(\rho^{-1}(x)), (a, b)) = ag(\rho^{-1}(x)) + b$$

where parameters a and b are used to ensure the positivity and control the selection pressure (e.g. if $a = 0$ then there is no pressure). The method is a random sampling of P_t according to

[1] The notation is a little confusing; generally the fitness function value of an individual is not its fitness. f has to be transformed "further" to obtain the fitness (a probability distribution).

the probability distribution P over P_t where

$$P(x) = \frac{f(x)}{\sum_{y \in P_t} f(y)}$$

Ranking selection. Here, a probability distribution over $\{1, 2, \ldots, |P_t|\}$ is fixed in advance and assigned to the population that is sorted according to the objective function g. The fixed distribution is linear or exponential but always monotone. The selection algorithm is again a random sampling with respect to the fixed distribution.

Binary tournament selection. Here, no explicit distribution is used. P'_t is filled by repeatedly drawing two elements from P_t randomly and inserting the better one (according to g) in P'_t. This process has the advantage of being extremely simple and easy to implement and having low computational complexity. Experimental investigations show (Goldberg, 1991) that its performance is also acceptable.

1.1.6 Other Heuristics

The algorithm-skeleton given in Fig. 1.2 represents a much larger class of search methods than EAs. This observation is not new, see (Eiben et al., 1995). As Glover puts it: "This terminology has acquired the distinction of embracing nearly every kind of method conceivable" (Glover, 1996). In this section a couple of examples are given to illustrate this idea.

Tabu search Tabu search is a method with the main characteristic of using both long term and short term memory (Glover and Laguna, 1998). Though EAs are usually called memoryless, it is not fair for several reasons. The first is that a population of individuals serves as a kind of long term memory about the history of the search. This memory can be used in all procedures of the algorithm in principle in several ways. Another kind of memory are the parameters, e.g. mutation probability. These parameters can be adapted on-line, during the search.

Nevertheless, the skeleton of tabu search is the same as the one shown in Fig. 1.2 provided that the procedures can access global variables that serve as memory. Selection corresponds to choosing the relevant memory components, recombination corresponds to creating a new solution based on these components. A great number of techniques are available to implement this, but the most well-known example for memory is the *tabu list* which consists of solutions to avoid. The recombination here is to create a solution that is different from all the members of the tabu list. The new population is created by updating the memory and the current solution according to the result of the evaluation. Evolutionary analogies do not necessarily work here of course. The general framework in Figs. 1.1 and 1.2 can be implemented in very different ways, many of which are not "evolutionary".

Stochastic hillclimbing This very simple method can be seen as a special EA with a population size of one, and with only mutation. Surprisingly, different versions of this method are

competitive to other heuristics (Mitchell et al., 1994; Juels and Wattenberg, 1996; Yagiura and Ibaraki, 1996; Ishibuchi et al., 1997; Eiben et al., 1998b; Boettcher and Percus, 2000). It is interesting that having a larger population was introduced by Holland for modeling complex adaptive systems, and not for function optimization (De Jong, 1993). In fact many traditional approaches used only one individual originally.

Simulated annealing This method is based on an analogy from physics (Aarts and Korst, 1989). When cooling a metal slowly, its atomic structure will be close to optimality. The population has one element here, and there is only mutation, but in the selection phase a special parameter is used called the *temperature*. If the temperature is high that the probability of accepting the worse solution is higher. If the temperature is zero, then only the better solution is accepted.

Scatter search and path relinking Path relinking is a generalization of scatter search (Glover and Laguna, 1998). The idea is that several good solutions are maintained and creating a new one is done by the recombination of these good solutions. It can be considered as a special case of multi-parent recombinations. In the case of scatter search the search space is a multi-dimensional real set, where this recombination is the linear combination of the good solutions. In the abstract case the method is called path relinking and the recombination is done by generating paths in the abstract search space between the good solutions (which is in fact done by combining their attributes just like in the case of crossover).

A last note The problem is that the division of the scientific field of heuristic search methods reflects historical rather than scientific boundaries. This makes communication harder and as a result unnecessary parallel work is done. Change cannot be expected soon because of social factors: just like in the European Union, people have to learn to live with the co-existence of several languages. It is very interesting to note that the situation is exactly the same in the field of *problems*, though it is much less evident. The usual classifications uses labels like SAT, subset sum, timetable generation, etc. However, it may well be that two *instances* of the SAT and the subset sum problem class resemble each other more than two instances of the same class. The boundaries are historical as well, just like in the case of methods, though the problem may be much deeper here. Chapters 4 and 5 address questions that are closely related to this problem.

1.2 Models

The models in the literature can be divided into two groups. In the first group we can find models that use the brute force approach and give exact mathematical results under certain assumptions that usually mean serious simplification. There are a lot of models of EAs that try to predict a certain feature of the working of the algorithm such as the diversity of the population (see e.g. (White and Flockton, 1995) for a summary of models of GAs, and (Rudolph, 1997) for a general discussion). An exact mathematical model of an algorithm called the *simple GA* (SGA)

has been developed as well (Vose, 1992; Vose, 1999). Unfortunately, it is not tractable so cannot be used to simulate the SGA on nontrivial problems.

The other group contains models that are indeed models. That means that they introduce new terms and relations and attempt to provide an insight into the working of the algorithm. In this section, we consider models belonging to this second group. These models make predictions based on some properties of the search problem or the space. Four models are discussed; the first is Goldberg's building block hypothesis (Goldberg, 1989) based on Holland's term schema (Holland, 1975). The second is based on Radcliffe's term *forma* (Radcliffe, 1994; Radcliffe and Surry, 1995; Radcliffe, 1992). The third is an interesting work of Vose (Vose, 1991). It is discussed in Section 1.2.3. Finally fitness landscapes are mentioned and fitness distance correlation (FDC) is discussed (Jones and Forrest, 1995). These theories were originally developed for explaining GAs, so we adopt the original terminology here. It does not restrict the generality of the discussion.

1.2.1 Building Block Hypothesis

Before formulating the approach, let us illustrate the ideas that lie behind the scene. This model explains evolution as a process that combines together certain features of individuals via the crossover operator. For example, in Africa, an "optimal" individual has black skin, eyes and hair. Starting from a uniform random sample of the people of the Earth as the initial population, men with black skin will survive with a relatively higher probability. The same is true for the eyes and the hair. Therefore, in the succeeding generations the combination of "black features" is very likely, and the process finally results in the optimal solution. It is clear that crossover has a central role here, and mutation is used only to correct the sampling error of the starting population (i.e. to ensure that features missing from the starting population may appear later). This model also suggests using the minimal alphabet for coding, i.e. the optimal chromosome space is thought to be $C = \{0,1\}^n$ for some $n \in \mathbb{N}$. This is so because, for a fixed search space S, this representation range maximizes the scope of the crossover operator by containing the longest codewords. Thus, we fix $C = \{0,1\}^n$ and define features of the codewords (*schemata*) that are thought to be processed effectively by crossover. Than, the building block hypothesis is discussed examining the predictive power of the model. Finally, a summary of the drawbacks is given. Throughout this section, the application of 1-point crossover, traditional mutation and proportionate selection with $f = g \circ \rho^{-1}$ is assumed unless otherwise stated[2].

Schemata. Schemata are denoted by the elements of $\{0,1,*\}^n$ where $*$ is a "don't care" symbol. An $H \in \{0,1,*\}^n$ denotes the subset of C in which the elements contain 0 (or 1) in the positions where H contains 0 (or 1). For instance, for $n = 3$, $*** = \{0,1\}^3$, $*01 = \{001, 101\}$. The *schema theorem* describes how schemata are processed during the genetic

[2]The positivity of f has to be assumed too due to the proportionate selection, but it is only a simplification of the discussion. In practice, a lower bound on $f = g \circ \rho^{-1}$ can usually be given (recall that C is finite).

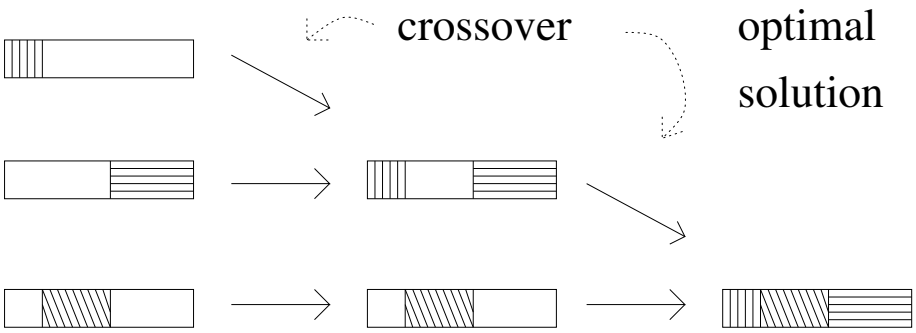


Figure 1.9: Illustration of the building block hypothesis.

search if proportionate selection is used:

$$E(|H|_{t+1}) \quad \geq \quad |H|_t \frac{\overline{f}_t^H}{\overline{f}_t} \left(1 - \sum_{\omega \in \Omega} p_\omega^H\right) \tag{1.1}$$

where

- $E()$ denotes an expectation value
- $|H|_t$ is the number of individuals in P_t that are members of schema H
- $\overline{f}_t^H$ is the *observed fitness* of schema H in time t, i.e. the average fitness of all the individuals of P_t that are members of H
- $\overline{f}_t$ is the average fitness of P_t
- Ω denotes the set of genetic operators
- the propability p_ω^H quantifies the potential disruptive effect on schema membership of the application of operator $\omega \in \Omega$

Above average schemata with small disruption coefficients receive higher proportion in the next population as stated by (1.1). Such schemata are rather numerous; an exact mathematical discussion can be found in (Bertoni and Dorigo, 1993). This effect is referred as *implicit parallelism*.

Prediction with the Building Block Hypothesis

The building block hypothesis is illustrated in Fig. 1.9. It suggests that the optimal solution is made up of *building blocks* i.e. short above average schemata that are combined together during the search. Note that there is another sense in which the term building block is used. This other sense also involves the notion of linear separability (Radcliffe, 1997), i.e. requires that there is no interdependence (or *epistasis*) between the building blocks. In an even more general setting

buildings blocks may even overlap (Kauffman and Levin, 1987; Altenberg, 1997b; Altenberg, 1994). Here we will use this term as Golberg introduced it originally. This hypothesis allows us to make predictions in the sense that special problems can be designed that are expected to be hard or easy for the GA. The first type of problems are the GA-deceptive problems (Liepins and Vose, 1991b; Whitely, 1991; Goldberg, 1989) while the easy problems are called royal road problems (Mitchell et al., 1994).

Deceptive problems. It is clear that (if the building block hypothesis is correct) if short, above average schemata provide bad solutions if combined together then the GA will be misled. Such problems can be designed explicitly using coefficients obtained by transforming the fitness function with the help of e.g. the Walsh transform (Khuri, 1994).

Here, the simplest possible deceptive function will be given for illustration, the so called *minimal deceptive problem*. Let $n \geq 2$ and let us fix two arbitrary bit positions. The notations of (1.1) will be used but indicating only the two fixed bit positions of the schemata; the other letters are fixed in $*$. Let

$$\overline{f}_{11} \quad > \quad \overline{f}_{00}, \overline{f}_{01}, \overline{f}_{10} \tag{1.2}$$

If we would like f to be deceptive then we must have

$$\overline{f}_{*0} = \frac{\overline{f}_{00} + \overline{f}_{10}}{2} \quad > \quad \overline{f}_{*1} = \frac{\overline{f}_{01} + \overline{f}_{11}}{2} \tag{1.3}$$

or

$$\overline{f}_{0*} = \frac{\overline{f}_{00} + \overline{f}_{01}}{2} \quad > \quad \overline{f}_{1*} = \frac{\overline{f}_{10} + \overline{f}_{11}}{2} \tag{1.4}$$

Since both cannot hold in the same time together with (1.2), without the loss of generality e.g. (1.3) can be assumed. Now we have

$$\overline{f}_{10} \quad < \quad \overline{f}_{00}, \overline{f}_{01}$$

from (1.2) and (1.3), therefore there are two types of *2-bit deceptive* problems:

$$\begin{array}{lll} \text{Type I} & : & \overline{f}_{01} > \overline{f}_{00} \\ \text{Type II} & : & \overline{f}_{01} \leq \overline{f}_{00} \end{array}$$

Computer simulations show (Goldberg, 1989) that the Type II problem can mislead the GA if the initial proportion of 11 in P_0 is small enough. However, in most of the cases the optimum is found so the building block hypothesis seems to be violated.

Royal Road Functions Royal road functions are supposed to be particularly easy for the GA. A well-known royal road function R1 is shown in Fig. 1.10. R1 is defined on $C = \{0,1\}^{64}$. An $x \in C$ gets 8 points added to its fitness for each schema in Fig. 1.10 that contains it. The optimum is the word containing only 1s and its fitness is 64 by definition. The performance

```
H1:  11111111**************************...*******  +8
H2:  ********11111111********************...*******  +8
H3:  ****************11111111**********...*******  +8
H4:  ************************11111111***...*******  +8
H5:  *****...*****11111111************************  +8
H6:  *****...************11111111****************  +8
H7:  *****...********************11111111********  +8
H8:  *****...******************************11111111  +8

opt: 11111111111111111111...11111111111111111111  64
```

Figure 1.10: The royal road function R1.

of the GA on R1 has been investigated (Mitchell et al., 1994) and has been compared to the performance of the stochastic hillclimbing algorithm (SHC)[3]. The SHC turned out to perform much better than the GA on R1 providing an additional argument against the building block hypothesis.

Criticism

The building block hypothesis and the schema-based approach has been criticized recently by several authors. Here, a brief summary of the problematic points of the hypothesis is given. (See also (Rudolph, 1997).)

Central role of n-point crossover. Successful experiments has been performed using operators that do not respect traditional schemata (Eshelman, 1991). On the other hand, the German school evolution strategies (ES) emphasizes the important role of mutation (Bäck et al., 1991).

Minimal alphabet. It has been shown that there are problems that are hard in an alphabet and easy in another (Kingdon and Dekker, 1995). Moreover, there are methods for alphabet optimization (Tóth et al., 1995) which itself is an argument against a priori fixing the minimal alphabet as the chromosome space.

Implicit parallelism. There has been a lot of controversy regarding this issue. The main problem is that in Holland's original discussion it is assumed that the (non-overlapping) schemata are processed independently; however, with finite populations it is not the case as pointed out by many (Grefenstette, 1991; Kargupta, 1995). For example, this problem is the reason of the failure of the building block hypothesis on the royal road functions; the linkage of schemata (the so called *genetic hitchhiking*) leads to premature convergence.

[3]The SHC starts with a random individual. In the hillclimbing step, the SHC inverts one random bit. If the new solution is better, then it replaces the old one.

Schema theorem. According to some authors (e.g. (Juliany and Vose, 1994)) the schema theorem is only a trivial statement about proportionate selection. It does not take into account that the schemata are not only disrupted but also created by the genetic operators. It is based on the *observed* fitness of the schemata but if the variance of the fitness of a given schema is high then the observed fitness is a bad approximation of the average fitness. Finally and most importantly it has been pointed out (Vose, 1991) that the schema theorem applies to every subset of the chromosome space if the disruption coefficients are counted properly so schemata do not have a special role.

Deceptive functions. The fact is deceptive functions have a very simple structure (Whitely, 1991). Care should be taken when stating that deceptive problems are the hardest problems for the GA. In (Venturini, 1995) a very simple modification of the GA is suggested for handling deceptivity.

1.2.2 Forma Analysis

Radcliffe's approach offers a solution to some problems of the schema based model. He points out that the subsets of the search space S defined by the schemata (provided that $C = \{0,1\}^n$ for some $n \in \mathbb{N}$) are not necessarily the most meaningful ones. The "useful" subsets are those that have a low fitness variance and they should be discussed independently of the representation. Instead, "clever" genetic operators should be used that concentrate on S, the chromosome space is of secondary importance. This approach solves many problems of the building block hypothesis, however it considers to be necessary to involve a priori knowledge about properties of the problem at hand or even empirical investigation of the features of the search space in order to make it possible to design the clever operators. As it will be shown in Chapter 4, it is not necessarily the case.

Of course to achieve an efficient computer implementation, it seems reasonable to design a representation that reflects the features of S. The model is built up with this idea in mind in the literature so we will follow this direction as well. First we discuss how can a priori observed good features of the search space be used to design a representation and then general genetic operators will be shown capable of handling these representations. Finally an example illustrating the predictive power of the model is given. Throughout this section, for the sake of simplicity but without the loss of generality it is assumed that the domain of ρ is S i.e. $\rho^{-1}(C) = S$.

Formae and Representation Design

The binary representation ($C = \{0,1\}^n$ for some $n \in \mathbb{N}$) can be defined by n equivalence relations given by the schemata with one defining (0 or 1) position. (The two complementary schemata corresponding to the same position (e.g. $0 * \cdots *$ and $1 * \cdots *$) define a classification (and thus an equivalence relation) over S.) This observation leads to a generalization of the binary representation; the *genetic representation*. Before giving the exact definition we need to examine some properties of sets of equivalence relations over S.

Definition 1. A set $\Psi = \{\psi_1, \ldots, \psi_j\}$ of equivalence relations over S *covers* S iff for every $x \in S$ at least one set Y of equivalence classes of the elements of Ψ exists such that

$$\bigcap_{y \in Y} y = \{x\}$$

Definition 2. Let $[\psi] = \{[y]_\psi \ : \ y \in S\}$ where ψ is an equivalence relation over S and for $y \in S$ $[y]_\psi$ is the equivalence class of y w.r.t. ψ.

Definition 3. Let Ψ be a set of equivalence relations over S that cover S. A $\rho \ : \ S \longrightarrow C(\subseteq \Sigma^n)$ representation is *genetic* iff there is a bijective mapping between $\Sigma \times \{0, \ldots, n-1\}$ and $\bigcup_{\psi \in \Psi} [\psi]$. A $\psi \in \Psi$ is called a *gene* and $[\psi]$ is the set of *alleles* of gene ψ.

An example of genetic representations is the permutation representation of the TSP (see Section 1.1.3) where the corresponding equivalence relations classify S according to which town is visited at a given place. An important property of some genetic representations is described in Definition 4.

Definition 4. A genetic representation $\rho \ : \ S \longrightarrow C(\subseteq \Sigma^n)$ is *orthogonal* iff $C = \Sigma^n$)

Clearly, the traditional binary representation is orthogonal, however, the permutation representation of the TSP is not since every town can be visited only once. Orthogonality is essential when designing genetic operators since in the case of non-orthogonal representations an operator normally needs some repair mechanism; the genes cannot be mixed "blindly". In the case of genetic representations the more general term *forma* replaces schemata.

Definition 5. Let ρ be a genetic representation with the set equivalence relations Ψ. The set of all *formae* is given by

$$\Xi = \{\bigcap_{y \in Y} y \ : \ Y \subseteq \bigcup_{\psi \in \Psi} [\psi]\}$$

In the case of the permutation representation of the TSP, an example of formae could be the set of tours that has a given town e.g. at the end of the tour.

The process of representation design proceeds in the opposite direction as would be indicated by the above discussion. The researcher first tries to exploit knowledge about the search space and to find a set of meaningful equivalence relations over it (i.e. that have low fitness variance (see Section 1.2.2) and cover S). He then defines the representation according to these equivalence relations. The operators do not have to be defined separately for each case since there are general operators that handle an arbitrary genetic representation. Such operators are discussed in Section 1.2.2.

RAR and $\mathbf{R}^3$

The definition of representation independent genetic operators is based on the observation that ad hoc operators used in different applications and representations tend to share some properties. Therefore, an exact description of these common properties may help to create general operators. Three of such features of crossover operators are described below: *respect, assortment* and *transmission*. (Operators of orthogonal representations normally have these features.)

respect A crossover operator *respects* a genetic representation if and only if every child it produces contains all the alleles common to its two parents.

transmission A crossover operator *transmits genes* w.r.t. a given genetic representation if and only if each allele in every child it produces is present in at least one of the child's parents. It is easy to see that an operator that transmits genes is respectful.

assortment A crossover operator is *assorting* if and only if it is possible for it to produce every child that contains only alleles present in the two parents.

For example the generalized n-point crossover for the TSP (see Section 1.1.4) is not transmitting due to the final patching phase, but it is clear that it respects the representation at hand.

Based on these abstract properties, several abstract operators have been defined. Two of these are the random assorting recombination (RAR) and the random respectful recombination (R^3). By definition, RAR always assorts and R^3 always respects w.r.t the genetic representation it is implemented for. Now we give the definition of RAR_w adopted from (Radcliffe and George, 1993).

Definition 6. Let the genetic representation at hand contain n genes, $x, y \in C$ and $w \in \mathbb{N}$. The algorithm of RAR_w is as follows.

1. Place w copies of each allele common to the two parents in a bag G together with w copies of "barred alleles" which are present in neither parent.

2. Place one copy of each allele in only one of the parents in the bag G together with a copy of its barred counterpart.

3. Repeatedly draw from the bag G, in a random order, barred and normal alleles. Normal alleles are included in the child whereas barred elements are excluded, in both cases subject to the primacy of earlier decisions (i.e. an element previously included cannot be excluded and vice-versa).

4. This process continues until either child is fully specified or the bag is empty. Should the bag empty before the child is fully specified, remaining genes are assigned at random.

It is clear that for any $w < \infty$ RAR_w assorts. For increasing values of w, RAR_w shows increasing respect. For $w = \infty$, $RAR_w = R^3$. Note that it is not important for w to be an integer; the definition can be generalized for any positive real number w. A final remark: the computer implementation is usually not similar to the process described in Definition 6 and of course may considerably differ in the case of different representations.

Prediction with Formae Analysis

Unlike the building block hypothesis (see Section 1.2.1) forma analysis emphasizes the role of the fitness variance in the formae. Low fitness variance minimizes the sampling error, so the GA gains reliable information about the performance of certain formae. Thus a representation

is predicted to be successful if its formae have low fitness variance in them. In (Radcliffe and Surry, 1995) several representations of the TSP were investigated and the one with the lowest fitness variance tended to give the best results under very different conditions. This winner representation is the so called corner representation where a tour is encoded by giving the adjacent cities as an unordered pair for every city. Therefore, the tours in Fig. 1.4 would be encoded as

$$(\{2,6\},\{1,5\},\{4,6\},\{3,5\},\{2,4\},\{1,3\})$$

and

$$(\{3,6\},\{4,5\},\{1,5\},\{2,6\},\{2,3\},\{1,4\})$$

respectively. Note that the genes of this representation contain $\left(\begin{array}{c} n \\ 2 \end{array} \right)$ alleles each (n is the number of cities) which is in high contrast with the minimal alphabet principle of the building block hypothesis.

Criticism

Clearly, forma analysis is a great step ahead; it points out that the minimal alphabet is not necessarily the best choice, takes into account that the schema theorem depends on the observed fitness (by considering the fitness variance of formae) and applies to arbitrary subsets of S (by choosing arbitrary subsets of S as a basis of the representation). Eludes the problem of deceptivity by suggesting a clever design of the representation instead of fixing it in advance. However, there are problems.

Old ideas behind. The main ideas remain the same; formae are thought to be combined together during the search process according to the idea of implicit parallelism. The aim of designing genetic operators that respect or assort formae is exactly this.

Design problem. In real world problems, extracting useful knowledge about the search space is generally almost impossible and testing the fitness variance of a given subset of S can be very expensive.

Fitness variance. Care should be taken when making predictions on the basis of fitness variance of formae of a given representation. For instance, the performance of the GA may be good even with a representation with formae of a very high fitness variance in some cases. The root of this problem is that there are other subsets of the chromosome space that are at least as suitable for modeling the search (see Chapter 4).

As a summary we can conclude that forma analysis is nothing else but a good generalization of the building block hypothesis for arbitrary representations extending it by advising on representation design and analysis.

1.2.3 The Random Walk Model

This section is devoted to a short article of Vose (Vose, 1991). The results of this paper have not been used for defining problems that are hard or easy for the GA nor for predicting its performance on certain problems to the best knowledge of the author. However it contains very interesting ideas that seem to be mistreated in the literature. The work is usually cited to mention that Vose proved that not only traditional schemata obey the schema theorem. He did, but this result occupies only the first three pages (Section 1 and 2). The main results are on the remaining nine pages as indicated by the last sentence of Section 2:

> While not disputing the building block hypothesis, we offer an alternate theoretical explanation of why GAs have enjoyed practical success.

The schema theorem has no role in the second part of the paper. Moreover, its title is misleading; the paper has little to do with schemata and the building block hypothesis, instead, it is a totally novel approach. It is interesting that Vose himself does not seem to think (Vose, 1991) important in the sense we do. For instance, in (Juliany and Vose, 1994) the schema theorem and its drawbacks are discussed but the paper in consideration is not even mentioned. (Battle and Vose, 1993) also supports this claim.

Here, a summary of the approach is given in a somewhat generalized form; Vose uses the chromosome space $C = \{0,1\}^n$ and the fitness function but we will consider the search space S[4] and the objective function g[5] with an arbitrary representation. We will ignore the (straightforward) proofs of the theorems mentioned here.

The Model

Let H be an arbitrary subset of S and P be a finite population. Then, let $|H|_P$ be number of elements of P that are codes of members of H and let

$$\overline{g}_P(H) = \frac{1}{|H|_P} \sum_{\rho^{-1}(x) \in H} g(x)$$

be the average of H w.r.t. P.

Definition 7. A $H \subseteq S$ is *global* iff

$$(\exists P)(\overline{g}_P(H) > \overline{g}_P(S)) \;\Rightarrow\; (\forall Q)(|H|_Q > 0 \;\Rightarrow\; \overline{g}_Q(H) \geq \overline{g}_Q(S))$$

Globality means that H enjoys constantly positive selective pressure independently of the population it is represented in (except the case when it occupies the whole population). Definition 8 is needed for a characterization of the global subsets of S.

Definition 8. Let $H \subseteq S$. H is

[4] We assume that $|S| = |C|$; otherwise we could restrict ourselves to $\rho^{-1}(C) \subset S$.

[5] Let g be injective. It is not a serious assumption since S is finite; anyway, it is needed only for the sake of the simplicity of the notations.

- *monotone increasing* iff $H = g^{-1}([a, +\infty))$ and
- *monotone decreasing* iff $H = g^{-1}((-\infty, b))$

for some $a, b \in \mathbb{R}$. H is *not monotone* otherwise.

Theorem 1. *$H \subseteq S$ is global if and only if it is monotone (increasing or decreasing).*

Theorem 2. *Let $H \subseteq S$ be global and P be a finite population. If $\overline{g}_P(H) > \overline{g}_P(S)$ then for every predicate G,*

$$\overline{g}_P(G) > \overline{g}_P(H) \;\Rightarrow\; (G \cap P \subset H) \vee (\overline{g}_P(H \cap G) > \overline{g}_P(G))$$

To understand Theorem 2 let us assume that a $G \subset S$ is stronger than a global subset of S H w.r.t. population P. The theorem says that G cannot reduce the survival probability of H because it is either contained in H or its elements in H are responsible for the high strength. What about the genetic operators? Vose defines stability of subsets of S to handle this question w.r.t. the crossover operator.

Definition 9. Let $H \subset S$ and $S(H)$ be the probability that H is invariant under (a given implementation of) crossover of distinct members of H.

- H is *stable* iff $S(H) = 1$
- H is *semi-stable* iff $S(H) \approx 1$
- H is *unstable* iff $S(H) = 0$

As admitted by Vose, these classifications are coarse. The aim is to identify features of subsets of S that describe how crossover disrupts them. Clearly, if the global subset of S H is stable then once established in the population, it is permanent.

The *random walk* analogy is simply to consider $\overline{g}_P(S)$ as a random variable that can move to the left with a diminishing or zero probability depending on the stability of the monotone increasing subsets of S (i.e. if they are stable or semi-stable, respectively).

Finally, let us examine an implication of the approach; namely the *sequential* way of looking at the optimization process. It is the common feature to the *wave model* (see Chapter 4).

Theorem 3. *Let G and H be monotone increasing subsets of S which are both represented in some population P. Then,*

$$G \subset H \;\Rightarrow\; \overline{g}_P(G) \geq \overline{g}_P(H)$$

where strict inequality holds in the case of strict containment w.r.t P.

Now, let the objective function values contained in a population P be

$$g_1 < g_2 < \cdots < g_n$$

then the predicates

$$H_i = g^{-1}([g_i, +\infty))$$

are global and

$$\overline{g}_P(S) = \overline{g}_P(H_1) < \cdots < \overline{g}_P(H_n)$$

If these subsets of S are stable (or at least semi-stable) then the search can be considered as the successive domination of the population by the stronger and stronger subsets that contain each other, i.e.

$$H_i \subset H_j \quad (i < j)$$

Criticism

Two remarks should be made regarding the main conclusion of (Vose, 1991) worded by Vose as follows:

> The genetic paradigm will succeed when monotone increasing predicates are stable or semi-stable.

$\Leftarrow$ The role of genetic operators is considered to be only disruption as in the case of the schema theorem. However, as will be shown in Chapter 4, the genetic operators have very significant features besides the stability. Due to these features it is possible that the GA performs very poorly even when the monotone increasing subsets are all stable.

$\Rightarrow$ The approach suggests that if the monotone increasing subsets are stable then the GA is successful. It is not necessarily the case as shown in Chapter 4. The problem is that Vose considers only monotone increasing subsets that are defined on the whole search space while the GA can watch only a very tiny slice of this space. This means that sometimes safe search can be performed even with rather unstable monotone increasing subsets involved.

1.2.4 Fitness Distance Correlation

Fitness distance correlation (FDC) was introduced as a measure of problem hardness (Jones and Forrest, 1995). This measure is based on the notion of *fitness landscapes*. A fitness landscape is defined simply by adding a distance measure to the search space giving it structure. This structure is intended to reflect the structure which is implicitly defined by the search algorithm under consideration. Using this distance function it is possible to talk about e.g. the neighborhood of a solution, which is an important notion of many heuristic search methods. This structure also makes it possible to think of the fitness function as a landscape, i.e. that has hills, ridges, etc. Terminology like e.g. "ruggedness" of a fitness function comes from this landscape analogy. It is interesting to note that this analogy can be very misleading especially in the case of high dimensional or binary spaces as results from coding theory mentioned in Section 2.4.3 show. Loosely speaking, high dimensional sets are like a good city-car: they can be very small from the outside but huge from the inside. Or to put it another way: according to a well known example, an n-dimensional orange (in the Eucledian space) has only skin as n goes to infinity. This effects illustrate why it is practically impossible to think about the neighborhood relations in terms of the landscape analogy.

In spite of the above difficulties fitness landscapes are very useful for gaining some insight into the structure of problems as the popularity of FDC shows (though see (Altenberg, 1997a)). For other measures see (Manderick, 1997).

Definition of the FDC

The FDC is the correlation of two properties of a solution. The first property is the distance of the solution from the locus of a fixed global optimum. The other property is the distance of the fitness of the solution from the value of the global optimum. For the sake of completeness, the definition of the correlation of two discrete random variables x_1 and x_2 is

$$C = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_1 - m_1)(x_2 - m_2)}{s_1 s_2}$$

where N is the number of possible values, m_1, m_2 and s_1, s_2 are the means and standard deviations of x_1 and x_2 respectively.

Illustration of the FDC

From a practical point of view, to calculate the FDC one needs the place and the value of a global optimum and a sample of the solution space along with the corresponding fitness values. The process of this calculation has an interesting byproduct, an two dimensional plot, which depicts the sample points in the space of the two properties under consideration. This plot is probably even more interesting than the FDC itself, and definitely gives much more insight to the problem structure. To illustrate this we give two examples that have a fairly different structure. Both examples are defined over the binary domain $\{0,1\}^n$. The distance used is the Hamming distance in both cases. Following the usual practice, when creating the plots a small random noise was added to every point to illustrate the density of points at a given place.

Needle in the Haystack The first example is a simple needle-in-the-haystack problem. The space has a simple linear structure but the global optimum is at the place which is predicted worst by this simple structure. The function used here is defined in (Liepins and Vose, 1991b) over a 10 bit space. The two dimensional plot depicting the interdependence of the fitness and distance from the global optimum is shown on Fig. 1.11. It is clear that the correlation coefficient is -1, i.e. there is negative correlation, which indicates a deceptive problem. The search algorithms that use the Hamming neighborhood for search are expected to run into a local optimum.

Subset Sum For the definition of the problem see Section 2.5.3. Figure 1.12 was calculated over a complete 12 bit space. It can be seen at first sight that the correlation is 0. However the plot needs more explanation than in the previous case. Problems having a coefficient of 0 are usually considered very hard. It is because this indicates that the distance from the global optimum and the fitness is uncorrelated, so there is no information in the neighborhood structure. Analyzing the plot we can see that there is a lot of very good quality solutions, which

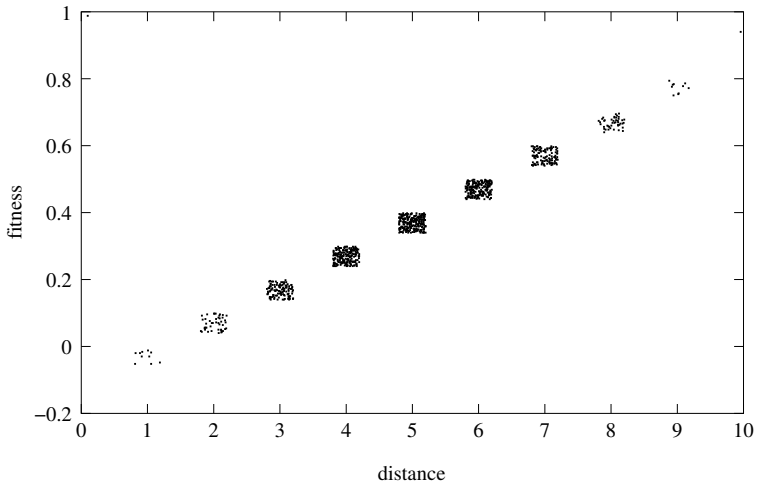


Figure 1.11: The correlation of fitness and distance from the global optimum for a needle-in-the-haystack problem.

have a very different genotype; a plot like this clearly indicates that there is a problem with the encoding of the problem. The surprising observation is that this problem is pretty easy for the EA, tough the complexity depends on some parameters. The reason is that in general many solutions are global optima. Many sections discuss the subset sum problem from the point of view of problem difficulty, but probably Chapter 4 is the most relevant here.

Criticism

FDC is a method that is very useful in many situations, especially if the plot is also taken into account. There are serious problems however. Section 4.3 offers a solution to these problems.

Global optimum. To apply the methods the global optimum has to be known. In different constructed problems this information may be available but in general in the case of a black box function it is not.

Prediction. To predict that a problem is difficult the value of the FDC is used. If it is close to 0 than the problem is considered hard. In many cases, for example for some instances of the subset sum problem, this forecast is not valid. Many solutions can be global optima and the structure of the space can be such that one of these optima can be found easily. If combined with other measures, like fitness correlation coefficient (Manderick et al., 1991), we may be able to refine this prediction.

A counterexample. A counterexample is given in (Altenberg, 1997a) as a special constructed function, the *ridge* function. Its structure is such that the majority of the space shows a deceptive

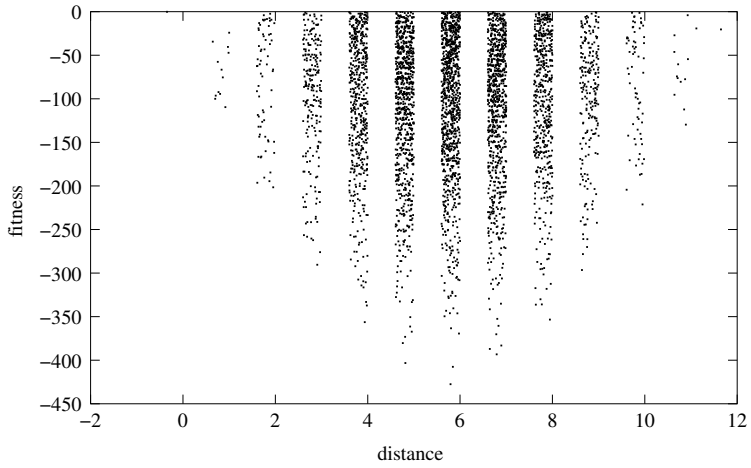


Figure 1.12: The correlation of fitness and distance from the global optimum for the subset sum problem.

structure but there is a path that contains only a few solutions but which leads to the global optimum. The key is that from every point of the space, this “ladder” can be reached relatively easily, which is quite counterintuitive. This is in connection with the failure of the landscape analogy, I refer again to Section 2.4.3 where interesting results can be found from coding theory that explain this effect.

Part II

Adaptive Exploration of Function-Structure

Chapter 2

Exploring Structure with GAS

This chapter introduces a niching technique called GAS (S stands for species) which dynamically creates a subpopulation structure (taxonomic chart) using a *radius function* instead of a single radius, and a 'cooling' method similar to simulated annealing. GAS offers a solution to the niche radius problem with the help of these techniques. A method based on the *speed* of species is presented for determining the radius function. Speed functions are given for both real and binary domains. We also discuss the sphere packing problem on binary domains using some tools of coding theory to make it possible to evaluate the output of the system. Finally two problems are examined empirically. The first is a difficult test function with unevenly spread local optima. The second is an NP-complete combinatorial optimization task, where a comparison is presented to the traditional genetic algorithm.

2.1 Introduction

In recent years much work has been done with the aim of extending genetic algorithms (GAs) to make it possible to find more than one local optimum of a function and so to reduce the probability of missing the global optimum. The techniques developed for this purpose are known as *niching techniques*. Besides the greater probability of the success of the algorithm and a significantly better performance on GA-hard problems (see (Beasley et al., 1993)), niche techniques provide the user with more information on the problem, which is very useful in a wide range of applications (decision making, several designing tasks, etc.).

2.1.1 Best-Known Approaches

Simple iteration runs the simple GA several times to the same problem, and collects the results of the particular runs. *Fitness sharing* has been introduced by Goldberg and Richardson (Deb and Goldberg, 1989). The fitness of an individual is reduced if there are many other individuals near it and so the GA is forced to maintain diversity in the population. *Subpopulations* can also be maintained in parallel, usually with the allowance of some kind of communication between them (see, for example, (Davidor, 1991b)). The GAS method has developed from this approach. The *sequential niche technique* is described in (Beasley et al., 1993). The GA (or any other

optimizing procedure) is run many times on the same problem, but after every run the optimized function is modified (multiplied by a derating function) so that the optimum just found will not be located again.

2.1.2 Problems

These techniques yield good results from several viewpoints, but mention sholud be made of some of their drawbacks, which do not arise in the case of our method, GAS.

Simple iteration is unintelligent; if the optima are not of the same value relatively bad local optima are found with low probability, while good optima are located several times which is highly unnecessary. *Fitness sharing* needs $O(n^2)$ distance evaluations in every step, besides the evaluation of the fitness function. It cannot distinguish local optima that are much closer to each other than the *niche radius* (a parameter of the method); in other words, it is assumed that the local optima are approximately evenly spread throughout the search space. This latter problem is known as the *niche radius problem*. The *sequential niche technique* also involves the niche radius problem. The complexity of the optimized function increases after every iteration due to the additional derating functions. Since the function is modified many times, "false" optima too are found. The method seems difficult to use for combinatorial problems or structural optimization tasks, which are the most promising fields of GA applications.

GAS offers a solution to these problems including the niche radius problem, which is the most important drawback of all of the methods mentioned earlier.

2.1.3 Outline

In section 2.2 we give the description of GAS at the level that is needed for understanding of the following parts. The reader who is interested in more technical details should refer to `ftp://ftp.jate.u-szeged.hu/pub/math/optimization/GAS/` for more information or GAS itself. In section 2.4 we give a possible solution to the niche radius problem with the help of the GAS system. Both real and binary problem domains are discussed. In section 2.5 we present experimental results. Two problems are examined. The first demonstrates how GAS handles the uneven distribution of the local optima of the optimized function. The second is an NP-complete combinatorial problem, where a comparison is presented to the traditional GA.

2.2 Species and GAS

2.2.1 Basic Ideas and Motivations

The motivation of this work was to tackle the problem of finding *unevenly spread* optima of multimodal optimization problems. For this purpose, a subpopulation approach seemed to be the best choice.

The obvious drawback of subpopulation approaches is that managing subpopulations need special algorithms and the system is relatively difficult to understand and maybe to use as well.

There are considerable advantages, however. Every subpopulation may have its own attributes that make it possible for them to adapt to the different regions of the fitness landscape. The subpopulations perform effective local search due to the mating restrictions that usually allow breeding only inside of a subpopulation, and the different subpopulations can even communicate with each other.

In our method GAS, every subpopulation (or species) is intended to occupy a local maximizer of the fitness function. Thus, new species are created when it is likely that the parents are on different hills, and species have to be fused when they are thought to climb the same hill (heuristics will be given later). To shed some light on the way GAS copes with unevenly spread optima, it is natural to use a terminology that is well known from the field of simulated annealing. Thus, when illustrating our definitions and methods, we will talk about the 'temperature' of species, the ability of escaping from local optima. In our system, we made the 'temperature' an explicit attribute of every species (it is the *attraction* of species, see Definition 12). This allowed us to offer an algorithm that 'cools down' the system while species of different 'temperatures' are allowed to exist at the same time. The basic idea of the algorithm is that 'warmer' species are allowed to create 'cooler' species autonomously discovering their own local are of attraction.

Finally, let us mention that due to our theoretical results, the large number of parameters of GAS can be reduced to a couple of easy-to-understand ones (see section 2.4).

2.2.2 Basic Definitions

In this section some basic definitions will be given that are necessary for introducing the algorithm. These definitions capture the relevan aspects of the structure of the search space from the point of view of GAS. Using the notations in the Introduction of (Rawlins, 1991), let D be the problem domain, $f : D \to \mathbb{R}$ the fitness function and $g : \{0,1\}^m \to D$ for some $m \in \{2,3,\ldots\}$ the coding function. (GAS searches for the *maxima* of f!)

The algorithm is based on some assumptions about the search space, which define its bias. This structure is given by a distance function ($d : D \times D \to \mathbb{R}$) and the term section (section : $D \times D \to P(D)$, where $P(D)$ is the power set of D). For a given search space it is possible to choose an arbitrary distance function and the definition of the shortest path (section) between two points. Let us give some examples for simple spaces.

Example 1. $D \subseteq \mathbb{R}^m$, D is convex.

$$\text{section}(x,y) = \{z : z = x + t(y-x),\ t \in [0,1]\}$$

Example 2. $D = \{0,1\}^m$ (So if $x \in D$ then $x = (x_1,\ldots,x_m)$)

$$\text{section}(x,y) = \{z : \text{if } x_j = y_j \text{ then } z_j = x_j\}$$

The second example is much less trivial: note that it allows a huge number of paths for reaching one solution from another. In abstratct spaces like the binary domain, it may well be that the appropriate choice of the definition of section is different for different problems. But

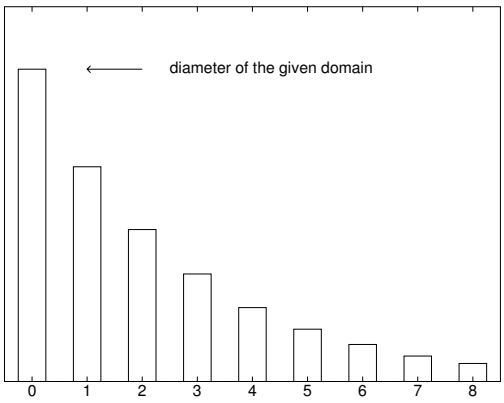


Fig. a

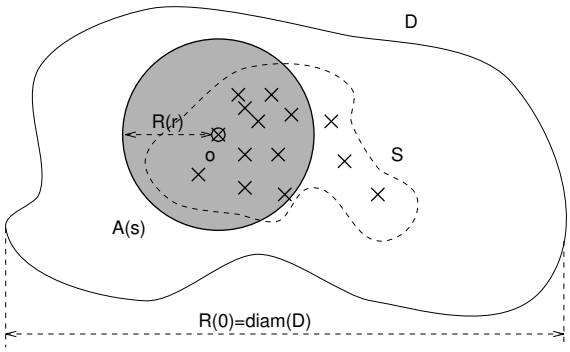


Fig. a

Figure 2.1: a: A possible radius function. b: Terms related to species.

even in usual cases like the real domain a different definition may be useful, which is based on some other coordinate system, for example polar coordinates.

Now let us give another definition which captures another aspect of the search space which is connected to the distribution of the local optima.

Definition 10. $R : \mathbb{N} \to \mathbb{R}$ is a *radius function* over D if it is monotonic decreasing, positive, $R(0) = \max\{ d(e_1, e_2) : e_1, e_2 \in D\}$ and

$$\lim_{n\to\infty} R(n) = 0.$$

Fig. 2.1a exemplifies these properties. Without knowing the algorithm it is hard to imagine the function of this definition, but the idea is that the radius function will be used to control the speed of 'cooling'. In fact, it gives the 'temperature' of the system in a given step (see section 2.3). At the beginning of the search the rough structure of the evaluation function is explored, and as the search proceeds, finer and finer details are found. The schedule of cooling is determined by this function.

From now on we will assume that a radius function R is fixed. At the implementation presented here, the radius function does not change over time during the optimization, i.e. it is static. Understanding the following definitions is crucial.

Definition 11. A *species* s over D is given by the triplet (o, l, S), where S is a population over D and the members of S are the *individuals* of s; $o(\in S)$ is the *center* of s and is such that $f(o) = \max f(S)$; $l(\in \mathbb{N})$ is the *radius index* or the *level* of s, and so the *radius* of s is $R(l)$. Recall that in GAs a population is a multiset (or bag) of individuals (e.g. $S =< x_1, x_1, x_2 >$).

Definition 12. $s = (o, l, S)$ is a species. Let $A(s) = \{a \in D : d(a, o) \leq R(l)\}$ be the *attraction* of s.

Fig. 2.1b illustrates the terms defined above. Species with small attraction behave as they

were 'cooler'; they discover a relatively small area, their motion in the space is slower but they can differentiate between local optima that are relatively close to each other. Note that for a species $s = (o, l, S)$, o is 'almost' determined by S. If the maximal number of different maximizers in a population would be one, Definition 11 would be redundant, however, it is possible that two different individuals have the same maximal function value. In this case the center is a random member of this maximal subpopulaltion. Also note that it is not necessary that $S \subseteq A(s)$.

Definition 13. Let T be a graph with the vertex set $V(T)$, where $V(T)$ is a set of species over D. T is a *taxonomic chart* (t.c.) if T is a tree and there is an $s_r = (o_r, 0, S_r)$ root in T, and if $(s_r =)s_0, s_1, \ldots, s_n$ is a path in T, then for the corresponding levels $l_0 < l_1 < \ldots < l_n$ holds.

Note that the root s_r of of every t.c. has the level 0 which means that its area of attraction $A(s)$ covers the whole domain D (see Definitions 10 and 12).

This term defines the structure in which the subpopulations are stored (thus the set of species has a corresponding tree structure). This structure is generated and maintained by GAS and forms part of the output. It is called a taxonomic chart to keep the biological terminology consistent, but in fact it is simply a tree with species in its nodes. Thus we can talk about the children and the father of a given species. The only restriction is that the children of a species must have a higher level than their father. This restriction limits the height of the tree to the maximal allowed level value. The recursive construction of the main algorithm will be based on this tree structure.

2.2.3 The Algorithm

Let $V(T_0)$ (T_0 is a t.c.) contain only $s_r = (o_r, 0, S_r)$, where S_r is randomly chosen. The algorithm in Fig. 2.2 shows how GAS creates a T_{n+1} t.c. from a given T_n t.c.

```
begin activity
   while (population size of T_n <
          maximum allowed) do
   begin
      choose two parents
      create two offspring
      update t.c.
   end
   dying_off
   fusion
end activity
```

Figure 2.2: The basic algorithm that creates T_{n+1} from T_n.

Before describing the parts of the algorithm, we should make a few remarks.

- It is the flexibility of steady state selection (Syswerda, 1991) that allows the algorithm to create and manage species, as will be shown later.
- The algorithm can be implemented in parallel on two levels: the level of the `while` cycle and the level of the `procedure`. (However, our implementation is not parallel.)

Let us now examine the parts of the algorithm.

Population Size. The population size of a given T t.c. is $\sum_{s=(o,l,S)\in V(T)} |S|$. In other words it is the sum of the sizes of all the species.

Choose Two Parents. From a given T, we first choose a vertex (a species) with a probability proportional to the number of the elements of the vertices. Then, we choose two parents from this species, using the traditional probability (proportional to the fitnesses of the elements of the species).

Create Two Offspring. From individuals p_1 and p_2, we create p_1' and p_2' by applying onepoint crossover and mutation operators to the parents.

Update t.c. Since this is the point where new species are created, this is the most important step. We have to decide here whether to separate the given two parents into two different species and we have to find species for the two newly created offspring. If we decide to separate the parents, we must find new existing species for them or create new species for them. The placing-back algorithm is shown in Fig. 2.3. The notations of the algorithm: $p_1,\ p_2$ are the parents, $p_1',\ p_2'$ are the two offspring, e is a random point on the section that connects p_1 and p_2 (note that p_1' and p_2' are not on this section in general), and s_p is the original species of the parents. We always mean $s_x = (o_x, l_x, S_x)$ on s_x for any symbol x. Function `move(p,s)` moves `p` to `S` and updates `o` if necessary. Parameter `strict` determines the precision of the search, i.e. the 'temperature' of the system. Increasing `strict` decreases 'temperature'. The way of using this parameter is described in section 2.3.

It is clear that for a concave or for a unimodal one-dimensional fitness function GAS will never create a single species.

Dying_off. `Dying_off` deletes as many elements from the population of the t.c. as were inserted in the `while` cycle keeping the population size constant. The method used for selecting elements to die is based on the ranking defined by the transformed fitness function $\hat{f}$:

$$\hat{f}(e) := \frac{f(e) - (\text{a global lower bound of } f \text{ on the whole population})}{|S|}$$

where e is in species $s = (o, l, S)$.

This means that species of small size have more chance to survive (and to grow). The precision of the procedure (i.e. the level of competition) can be varied during the optimization

```
if f(e) < f(p1),f(p2) then
   for x=p1,p2,p1',p2' do
      if (there is a child node s_c of s_p
                such that x is in A(s_c)) then
         move(x,s_c)
      { With the restriction that p1 and p2    }
      { must not be put into the same species. }
   for x=(a parent not put in so far) do
      create a new child
      s=(x,max{l_p +1,strict},<x>) for s_p
{ else: The parents are left in s_p. }

for x=(an offspring not put in so far) do begin
   s:=s_p
   while (x is not in A(s)) do s:=father node of s
   { if s=s_r then A(s)=D! }
   move(p,s)
end
```

Figure 2.3: The algorithm that places parents and offspring back in the population updating the t.c.

process. In section 2.3 we discuss how to use this possibility. `Dying_off` has no effect on the species structure (by definition) and does not delete the best individual of a species.

Fusion. The result of `fusion` depends on R and `strict` described earlier. After executing `fusion` for a given T t.c., we get T', for which the following will be true: if $s_1,\ s_2 \in V(T')$, then $d(o_1, o_2) \geq R(\text{strict})$. `Fusion` simply unites species of T that are too close to each other, and `strict` tells it what is too close. If s_1 and s_2 are united, the result species is $s = (o, \min\{l_1, l_2\}, S_1 \cup S_2)$, where $f(o) = \max\{\ f(o_1),\ f(o_2)\}$ and o is o_1 or o_2. In view of the tree structure, the species with the lower level absorbs the other. If the species have the same level, either of them may absorb the other.

2.3 Optimization with GAS

For global optimization with GAS, we suggest the algorithm shown in Fig. 2.4. For determination of the vector of evaluation numbers x and the radius function R, we suggest a method in section 2.4 based on the *speed* of species with a given radius in a given domain.

The main `for` cycle performs the 'cooling' operation. Increasing `strict` results in new species with smaller radii (see Fig. 2.3). The basic philosophy is to increase diversity at the beginning of every cycle and then perform optimization of the newly discovered areas. This kind of oscillation can be observed in biological systems as well.

```
begin GAS
   create a starting t.c. T_0
   for strict=1 to ST_m    { 0 < ST_m(=strict_max) < 8 }
      new species
      evolution
      stabilize
      iterate evolution until reaching
      x_strict function evaluations
end GAS

begin evolution
      for i=1 to 10 do activity
      immigration
      for i=1 to 5  do activity
end evolution
```

Figure 2.4: The high-level optimization algorithm.

We now describe the species-level genetic operators, used in the algorithm shown in Fig 2.4.

Immigration. For every species $s = (o, l, S)$ in a given t.c., $|S|/2$ randomly generated new individuals are inserted from $A(s)$. Immigration refreshes the genetic material of the species and makes their motion faster. It has a kind of greasing effect.

New species. This switch alters the state of the system towards managing species creation. It randomizes `dying_off` and relaxes competition by decreasing the lower bound of the fitness function, and so decreases the relative differences between individuals. According to some biologists (Csányi, 1982), species are born when the competition decreases; strong competition results in very similar individuals and new niches cannot be discovered this way. Our experiments support this opinion.

Stabilize. The effect of this is the contrary of `new species`. It prohibits the creation of new species and increases competition.

As a summary, we give here some heuristical arguments that support the subpopulation structure approach and use a radius function instead of a single radius.

- The number of distance calculations grows with the size of the t.c. instead of the size of the population.
- Application of species–level operators (e.g. fusion, immigration) becomes possible.
- Lower-level (closer to root) species manage to create new species in their attraction.

- The advantages of the technique based on the radius function and increasing `strict` (see Fig. 2.4) are similar to those of the 'cooling' technique in the simulated annealing method.

Finally, to make our discussion more rigorous, we give the definitions of stability of species and t.c. These definitions are not really necessary for the present discussion in the sense that will not be used in any strict mathematical environment. However, when stability is mentioned, it is ment in this sense. The impatient reader is free to skip these definitions.

Definition 14. $W \subseteq D$. Species s is *stable in* W if $o \in W$, and if $o_1, o_2, \ldots$ is a series of new centers inserted by GAS to s during running then it is impossible that for some i $o_i \notin W$.

Example 3. It is clear that s is stable in $W = \{e \in D : \ f(e) > f(o)\}$.

Definition 15. $e_0 \in D$, e_0 is a local optimum (with respect to d) of f. s is *stable around* e_0 if, for every $o_1, o_2, \ldots$ series of new centers inserted by GAS to s during running, $o_n \to e_0$ $(n \to \infty)$ with probability 1.

Example 4. $W \subseteq D$, $e_0 \in W$. If s is stable in W and e_0 is the global optimum of f in W (i.e. e_0 is a local optimum of D or else s could not be stable in W) and there are no more optima of f in W, then s is stable around e_0. This example would need a proof but we do not give it here because it is marginal from our present point of view.

Definition 16. T is a t.c. T is *stable* if every species of T is stable around distinct local optima of f.

Definition 17. T is a t.c. T is *complete* if T is stable and there is exactly one stable species around every local optimum of f.

2.4 Theoretical Results

In this section we discuss the theoretical tools and new terms that can be used due to the exact definition of the t.c. data structure and GAS algorithm.

2.4.1 Speed of Species

We do *not* assume that the optima of the fitness function are evenly spread; we create species instead that "live their own lives" and can *move* in the search space and find the niche on which they are stable. It can be seen that from this point of view determining the radius function R depends more upon the *speed* of the species than on the number of spheres (niches) of a given radius that can be packed into the space. The speed of a given species $s = (o, l, S)$ will depend on its radius $R(l)$. The larger the radius is the faster the species can move towards its stable state so the fewer the number of iterations it needs to become stable. This idea will be used when simultaneously dividing the available number of function evaluations among the species and setting the values of the radius function R.

The solution of the sphere packing problem mentioned above is the base of setting the niche radius parameter of the methods mentioned in the Introduction. This method has several drawbacks. One of these is that for example in the case of binary spaces it is possible to pack a huge umber of spheres to relatively small spaces so the resulting radius is too small. In such spaces we cannot hope that all the local optima will be found in general. However, this value is useful when evaluating the output of the system since it tells us what percentage of the possible number of optima we have found. In section 2.4.3 we discuss such packing problems in the case of binary domains. This discussion sheds some light on the basic differences between binary and real domains and suggests that our intuitions that work in real spaces may well be misleading in a binary domain.

Real Domains.

In real domains we have $D \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$. Let us fix a dimension number n and a species $s = (\vec{o}, l, S)$ and let us denote the radius of s by r (i.e. $r = R(l)$). (Recall, that for a species $s = (\vec{o}, l, S)$ the center of s, $\vec{o}$, is an n-dimensional real vector: $\vec{o} = (o_1, \ldots, o_n)$.)

The following suggestion for the definition of *speed* is an approximation. It is assumed that the fitness function f is the projection $f(\vec{x}) = x_1$ and GAS simply selects new individuals from the attraction of s, $A(s)$, randomly with a uniform distribution instead of generating them using parents and genetic operators and drops them into the species one by one. Note however, that this does not mean that GAS is approximated using a blind random search since the center of the species is always the best individual, and after generating the random element, the center is updated. This means that in the case of the projection function the expected value of the improvment remains constant while the species is moving upwards. The speed for a radius r and a dimension n will be the average step size towards the better region that results in generating one random individual.

Definition 18. The *speed* $v(r)$ of s is $(c_1 - o_1)/2$, where $\vec{c} \in \mathbb{R}^n$ is the center of gravity of the set

$$S_{n,r} = A(s) \cap \{\vec{x} \in \mathbb{R}^n : x_1 > o_1\}$$

To understand this definition, recall that in the case of the projection function the probability of improvement is $1/2$, i.e. hitting the half sphere which is above the center, i.e. $S_{n,r}$. If only this half sphere was sampled then the expected improvement would be the distance of the center of gravity of this half sphere and the center of the species. Since half of the hits result in no improvement on average, the expected improvement will be the half of this distance. In other words, let us choose a random element $\vec{x^*} = (x_1^*, \ldots, x_n^*)$ from $A(s)$ with a uniform distribution. Let $\xi = o_1 - x_1^*$ if $o_1 > x_1^*$, and $\xi = 0$ otherwise. Than $M(\xi)$ (the expected value of ξ) is $v(r)$. This means that $v(r)$ is given by the equation

$$v(r) = \frac{1}{2}\frac{1}{V(S_{n,r})}\int_{S_{n,r}} x_1 \, dx_1 \ldots dx_n \tag{2.1}$$

where $V(S_{n,r})$ is the volume of $S_{n,r}$. (Recall that if $\xi = 0$ then the center of s o is not changed by GAS.) It can be proved that

$$v(r) = \frac{\left(\begin{array}{c} n \\ \frac{n-1}{2} \end{array} \right)}{2^{n+1}} r \tag{2.2}$$

holds. In the general case (if n is even) (2.2), is defined with the help of the function $\Gamma(t+1)$, the continous extension of $t!$. $\Gamma(t+1) = \int_0^\infty x^t e^{-x}\, dx$, $\Gamma(1/2+1) = \sqrt{\pi}/2$ and $\Gamma(t+2) = (t+1)\Gamma(t+1)$.

Binary Domains.

Let $D = \{0,1\}^n$. Let us fix a dimension number n and a species $s = (o, l, S)$ and let us denote the radius of s by r (i.e. $r = R(l)$). We give a definition of speed similar to Definition 18. Like in the case of real domains, an approximation is used. It is assumed that the fitness of an individual $x \in D$ is given by the number of 1s in it, and GAS works as described in the case of real domains. As in the binary case, the speed for a radius r and a dimension n will be the average size of the first step of o after receiving one random individual. The difference is that in the case of binary domains, the starting center has to be fixed too since the average step sizes change as the center changes. More precisely, the average step size will decrease since the function is bounded from above unlike the projection function used in the real case. Let $e \in D$ such that the number of 0s is equal to or greater by one than the number of 1s depending of the parity of the number of bits. Let e be the fixed starting center. Let

$$S_{n,r} = \{e' \in D \,:\, d(e', e) \leq r\}$$

where the distance function d is the Hamming distance (the sum of the bit differences). Let us choose a random e^* from $S_{n,r}$ with a uniform distribution and let $\xi = d(e^*, e)$ if there are more 1s in e^*, and let $\xi = 0$ otherwise.

Definition 19. Let $v(r) = M(\xi)$ be the *speed* of species s in D if the radius of s is r.

We performed experiments to determine $v(r)$ (Fig. 2.5). It can be seen if $n \gg r$, then the equation

$$v(r) = \frac{3}{11}\sqrt{r} \tag{2.3}$$

seems to describe the speed. If r approaches n, the growing of the speed becomes slower than (2.3) would indicate.

2.4.2 Determining R and x

We use the notations of Fig. 2.4 here. Recall that ST_m is the number of steps in the 'cooling' procedure, the maximal value of `strict`, and x_i denotes the number of function evaluations at step i. Let us assume that the evaluation number N, the domain type and the corresponding

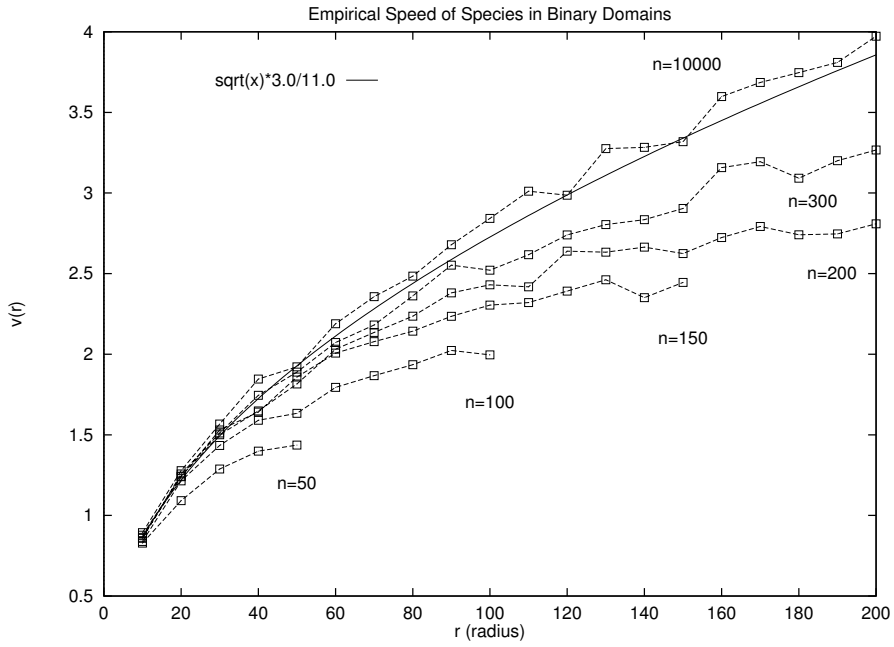


Figure 2.5: Speed in binary domains.

speed function v are given. We know that the function evaluations at the different levels sum up to the maximal allowed value:

$$\sum_{i=1}^{ST_m} x_i = N \tag{2.4}$$

We suggest a setting for which the system of equations

$$v(R(i))x_i = C \quad (i = 1, \dots ST_m) \tag{2.5}$$

holds where C is a constant (independent of i). This simply means that the species of the different levels receive an equal chance to become stabilized, i.e. if a species has a small radius and therefore a small speed then it will receive much more evaluations that allows it to find a stable place. From (2.4) and (2.5) it follows that

$$C = \frac{N}{\sum_{i=1}^{ST_m} \frac{1}{v(R(i))}}. \tag{2.6}$$

We note that C has an easily understandable meaning: it is the distance that a species of level i expectedly crosses during x_i iterations.

In GAS, the upper bound M of the number of species can be set. $M = [\text{population size}/4]$ by default. Now we can give the value of C:

$$C = R(0)M\nu \tag{2.7}$$

Recall that $R(0)$ is the diameter of the domain we examine. ν is a threshold value. Setting $\nu = 1$ means that every species receives at least sufficient function evaluations for crossing the

whole space, even if the number of species reaches M which makes the probability of creating a stable t.c. very high. In section 2.5 we examine the effect of several different settings of ν. Finally let

$$R(i) = R(0)\beta^i \quad (i = 1, \dots ST_m,\ \beta \in (0, 1)) \tag{2.8}$$

Choosing an R this way is attractive for several reasons. First, giving the whole function reduces to only one parameter, which has a clear meaning. Second, subproblems defined by the species will be similar to the original problem, giving the algorithm a recursive structure. Note that this setting may not be optimal for some problems that have special structure, though setting the maximal number of levels provides a possibility to adapt the algorithm to different situations even with this fixed radius function.

Using (2.6), (2.7) and (2.8), we can write

$$\frac{N}{R(0)M\nu} = \sum_{i=1}^{ST_m} \frac{1}{v(R(0)\beta^i)} \quad (\beta \in (0, 1)) \tag{2.9}$$

where everything is given except β. Since v is monotonous, the right side of (2.9) monotonically decreases as β increases and so reaches its minimum if $\beta = 1$. Using this fact, the feasibility condition of (2.9) is

$$\frac{N}{R(0)M\nu} > \frac{ST_m}{v(R(0))} \tag{2.10}$$

If (2.10) holds, (2.9) has exactly one solution. This property allows us to use effective numeric methods for approximating β.

Now, the number of parameters to set is significantly reduced. Section 2.5.1 gives a list along with explanations and settings used in the experiments.

2.4.3 Evaluating the Output

We based the setting of the parameters of GAS on the speed function. However, it is important to know the maximal possible size of a t.c. for a given radius function R (assuming an arbitrary large evaluation number and population size) since it tells us what percentage of the maximal possible number of optima we have found.

The problem leads to the general sphere packing problem and this has been solved neither for binary nor for real sets in the general case.

Real Case

In n-dimensional real domains Deb's method (Deb, 1989) can be used.

$$p = \left(\frac{\sqrt{n}}{2r}\right)^n$$

where r is the species radius, the domain is $[0, 1]^n$ and p is the number of optima, assuming that they are evenly spread. We note that this is only an approximation.

Binary Case

Results of coding theory can be used to solve the packing problem in binary domains since it is one of the central problems of this field. We will need the definition of binary codes.

Definition 20. $d, n \in \mathbb{N}$, $d \leq n$. $C \subseteq \{0,1\}^n$ is a $(n, |C|, d)$ *binary code* if $\forall c_1, c_2 \in C$: dist$(c_1, c_2) \geq d$ (The function "dist" is the Hamming distance, the sum of the bit differences.)

Definition 21. $d, n \in \mathbb{N}$. $A(n,d) := \max\{|C| \ : \ C \text{ is a } (n, |C|, d) \text{ binary code}\}$.

$A(n,d)$ has not yet been calculated in the general case; only lower and upper bounds are known. Such bounds can be found for example in (MacWilliams and Sloan, 1977; van Lint, 1992; McEliece, 1977). One of these is the Plotkin bound:

Theorem 4. *((Plotkin bound)) For $d, n \in \mathbb{N}$, we have*

$$A(n,d) \leq \frac{d}{d - \frac{1}{2}n} \quad \textit{if } d \geq \frac{1}{2}n$$

In a special case, the exact value is also known:

Theorem 5. *For binary codes and $m \in \mathbb{N}$, we have*

$$A(2^{m+1}, 2^m) = 2^{m+2}.$$

In Table 2.1 we show the Plotkin upper bounds for $2n = 32, 128$ and 1024. The values have been calculated according to the following formulas:

$$\begin{array}{lcl} A(2n, n+a) & \leq & \frac{n+a}{n+a-2n/2} = \frac{1}{a}(n+a) \\ A(2^{m+1}, 2^m) & = & 2^{m+2} \\ A(2n, n-a) & \leq & 2^{2a+1}2(n-a) \end{array}$$

2.5 Experimental Results

In this section we examine two problems. The first demonstrates how GAS handles the uneven distribution of the local optima of the optimized function. The second is an NP-complete combinatorial problem, where a comparison is presented to the traditional GA.

2.5.1 Setting of GA and GAS Parameters

In the following experiments, the settings of the traditional GA parameters are P_m (mutation probability) $= 0.03$ (see e.g. (Goldberg, 1989)) and P_c (crossover probability) $= 1$, while the population size $= 100$. In the `while` cycle of the basic algorithm (shown in Fig. 2.2), the maximum allowed population size is 110. For continuous domains, we used Gray coding as suggested in (Caruana and Schaffer, 1988).

The settings of the specific GAS parameters are the following:

d 2n	32	128	1024
n-3	3 328	15 616	130 304
n-2	896	3 968	32 640
n-1	240	1 008	8 176
n	64^*	256^*	2048^*
n+1	17	65	513
n+2	9	33	257
n+6	3	11	86
n+16	2	5	33
n+40	-	2	13

Table 2.1: Plotkin upper bounds for $A(2n, d)$. The indicated values are exact.

- R (radius function) and x (evaluation numbers) can be determined using the method described in section 2.4.

- M (maximal number of species in the t.c.) is set to $M =$ (pop. size)$/4$. Setting a larger value is not recommended since too many small species could be created.

- N ($\sum_{i=1}^{ST_m} x_i$) depends on the available time and computational resources. We used $N = 10^4$.

- ν (treshold) and ST_m (maximal strict level) are the parameters we tested so we used several values (see the descriptions of the experiments).

For simplicity, we run `evolution` only once after `new species` (see Fig. 2.4) but we note that increasing that number can significantly improve the performance in some cases. The cost of one `evolution` is 275 evaluations after `new species`, and 200 after `stabilize` at the above settings.

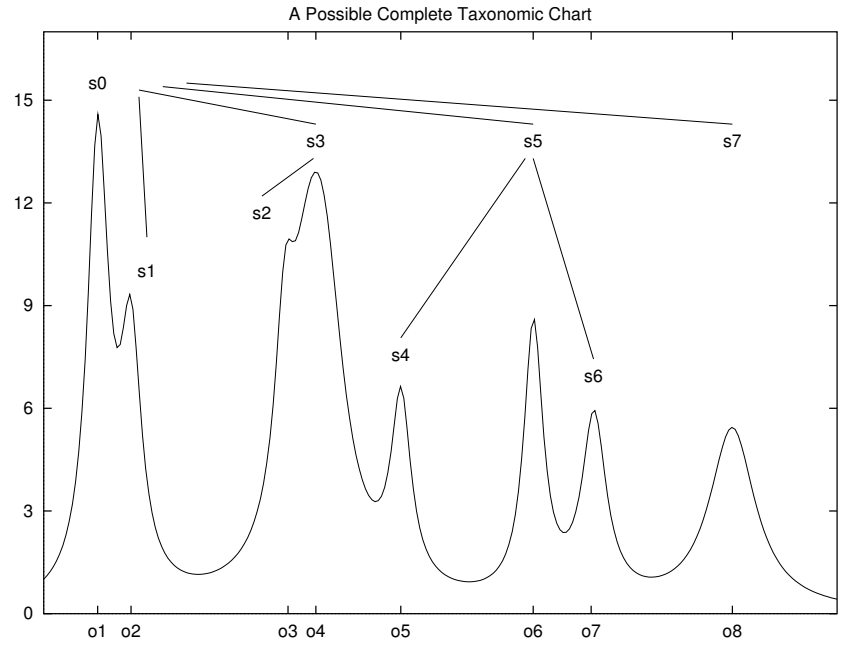


Figure 2.6: The function with unevenly spread optima.

2.5.2 A Function with Unevenly Spread Optima

The problem domain D is $[0, 10]$. The fitness function $f : D \to \mathbb{R}$.

$$\mathbf{a} = \begin{pmatrix} 3.040 \\ 1.098 \\ 0.674 \\ 3.537 \\ 6.173 \\ 8.679 \\ 4.503 \\ 3.328 \\ 6.937 \\ 0.700 \end{pmatrix} \quad \mathbf{k} = \begin{pmatrix} 2.983 \\ 2.378 \\ 2.439 \\ 1.168 \\ 2.406 \\ 1.236 \\ 2.868 \\ 1.378 \\ 2.348 \\ 2.268 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} 0.192 \\ 0.140 \\ 0.127 \\ 0.132 \\ 0.125 \\ 0.189 \\ 0.187 \\ 0.171 \\ 0.188 \\ 0.176 \end{pmatrix} \quad f(x) = \sum_{i=1}^{10} \frac{1}{(k_i(x - a_i))^2 + c_i}$$

f (shown in Fig. 2.6) is a test function for global optimization procedures suggested in (Törn and Žilinskas, 1989).

We have determined R and x for $\nu = 1/4, 1/2, 3/4$ and 1 (see Table 2.2). ST_m is 8 in every case. Recall that according to the algorithm in Fig. 2.4 the elements of x must be divisible by 200 (the cost of `evolution` after `stabilize`) and the sum of them must be $10^4 - ST_m \cdot 275$.

We run the corresponding algorithms 100 times. The numbers of stable species that converged to one of the local optima are shown in Table 2.3. The most important result is that *no* unstable species appeared in the output even for $\nu = 1/4$. The best results are observed in the

	$\nu = 1$		$\nu = 3/4$		$\nu = 1/2$		$\nu = 1/4$	
	R	x	R	x	R	x	R	x
1	6.61	200	6.333	200	5.978	0	5.334	0
2	4.369	200	4.011	200	3.573	200	2.845	0
3	2.888	400	2.54	200	2.136	200	1.517	200
4	1.909	600	1.609	400	1.277	400	0.809	200
5	1.261	800	1.019	800	0.763	600	0.432	600
6	0.834	1200	0.645	1200	0.456	1200	0.23	1000
7	0.551	1800	0.409	1800	0.273	2000	0.123	2000
8	0.364	2800	0.259	3000	0.163	3200	0.066	3600

Table 2.2: Radius and evaluation numbers for $\nu = 1/4, 1/2, 3/4$ and 1.

case of $\nu = 1/4$. Here, even o_3 was found 2 times in spite of its very small attraction. Fig. 2.7

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
$\nu = 1$	100	0	0	100	60	97	48	94
$\nu = 3/4$	100	1	0	100	65	87	72	94
$\nu = 1/2$	100	34	0	100	74	99	58	98
$\nu = 1/4$	100	25	2	100	85	100	90	100

Table 2.3: Number of stable species around the local optima.

shows the average number of species detected before increasing `strict` (after stabilizing for the old `strict`). From these values, we can gain information on the structure of the optima of the fitness function. For example, for radii greater than 3, very few species were created, which means that the optima are probably closer to each other than 3.

2.5.3 An NP–complete Combinatorial Problem

We study the subset sum problem here. We are given a set $W = \{w_1, w_2, \ldots, w_n\}$ of n integers and a large integer C. We would like to find an $S \subseteq W$ such that the sum of the elements in S is closest to, without exceeding, C. This problem is NP-complete.

We used the same coding and fitness function as suggested in (Khuri et al., 1993): $D = \{0,\ 1\}^{128}$. If $e \in D$ ($e = (e_1, e_2, \ldots, e_{128})$), then let $P(e) = \sum_{i=1}^{128} e_i w_i$, and then

$$-f(e) = a(C - P(e)) + (1 - a)P(e)$$

where $a = 1$ when e is feasible ($C - P(e)) \geq 0$), and $a = 0$ otherwise. Here, $\forall w \in W\ 1 \leq w \leq 1000$ and C is the sum of a randomly chosen subset of W (every element is chosen with a probability 0.5). We do not need a coding function here since D is the code itself.

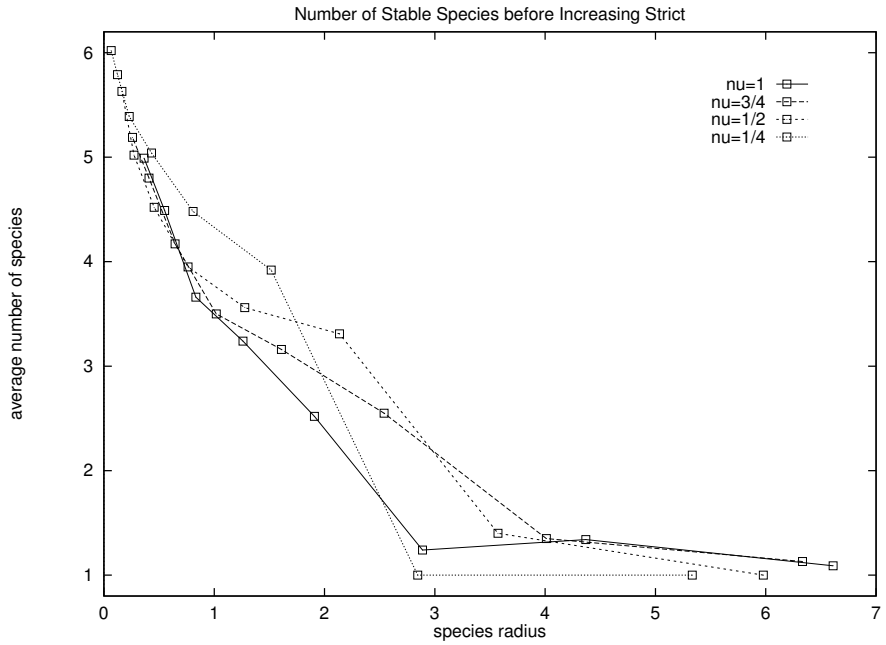


Figure 2.7: The species number increasing history (average of 100 runs).

We tested several values of ST_m. Table 2.4 shows R and x for $ST_m = 1, 2, \ldots, 6$. The value 8 is not feasible and 7 is also very close to that bound. $\nu = 1$ in every case. We run the corresponding algorithms 50 times. For comparison, in experiments on the same problem with two times more (i.e. $2 \cdot 10^4$ instead of 10^4) evaluation numbers in (Khuri et al., 1993), 0.93 optimal solutions were found per run. Here, this value is at least one for every ST_m, and for $ST_m = 2$ it is 2.62 (see Table 2.5). Besides this, many near-optimal solutions were found (as shown in Fig. 2.8) so we received much more information with only 10^4 function evaluations.

	$ST_m = 1$		$ST_m = 2$		$ST_m = 3$		$ST_m = 4$		$ST_m = 5$		$ST_m = 6$	
	R	x	R	x	R	x	R	x	R	x	R	x
1	2	9600	20	2600	47	1800	72	1400	93	1200	109	1200
2			3	6800	17	2800	41	1800	67	1400	92	1200
3					6	4600	23	2400	49	1600	79	1400
4							13	3200	35	2000	67	1400
5									26	2400	57	1600
6											48	1600

Table 2.4: Radii and evaluation numbers for $ST_m = 1, 2, \ldots, 6$.

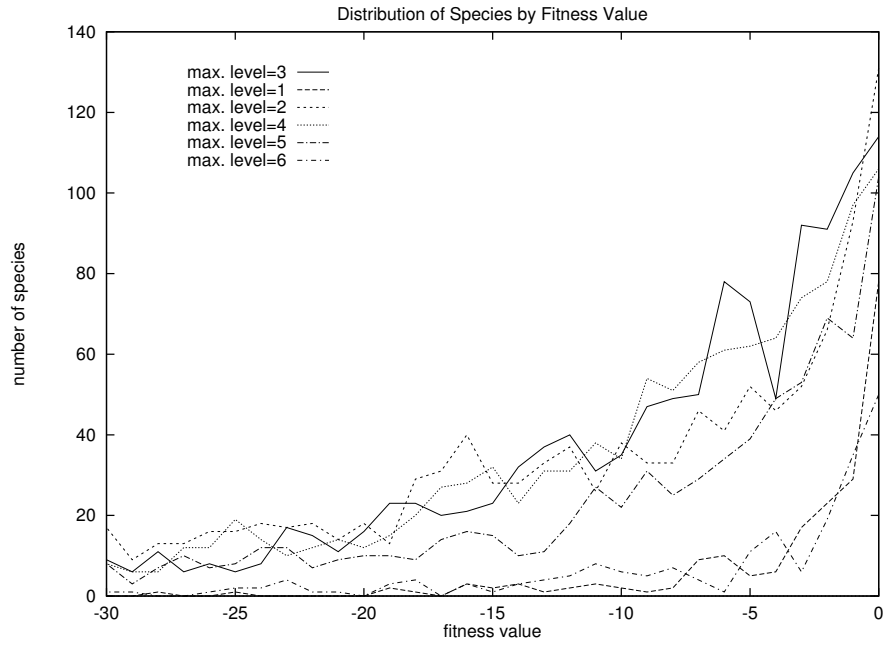


Figure 2.8: Number of near-optimal solutions found during the 50 runs.

2.6 Summary

In this chapter we have introduced a method called GAS for multimodal function optimization (or multimodal heuristic search). GAS dynamically creates a subpopulation structure called a taxonomic chart, using a radius function instead of a single radius, and a 'cooling' method similar to simulated annealing. The cooling method ensures that the structure of the search space is discovered step by step starting from a very rough approxmation and finishing with a detailed one.

The setting of the parameters of the method was based on the speed of species instead of

ST_m	opt. found/ run	avg. fitness of all spec.	number of species
1	1.56	-3.194	201
2	2.62	-15.616	1250
3	2.1	-10.866	1250
4	2.12	-12.568	1238
5	2.08	-12.84	846
6	1.0	-6.943	211

Table 2.5: Result of the experiment (50 runs).

their relative size to the search space, and the definition of speed was based on the structure of the space. The structure of the space is given by the user trough a distance function. Speed functions were given for both real and binary domains.

We performed experiments for a difficult test function with unevenly spread local optima and for an NP-complete combinatorial problem. In both cases our results are encouraging though much work will have to be done to examine the effects of the parameters of the method more thoroughly.

Chapter 3

UEGO, a General Paradigm

In this chapter UEGO, a new general technique for accelerating and/or parallelizing existing search methods is suggested. The skeleton of the algorithm is a parallel hill climber. The separate hill climbers work in restricted search regions (or clusters) of the search space. The volume of the clusters decreases as the search proceeds which results in a cooling effect similar to simulated annealing. Besides this, UEGO can be effectively parallelized; the communication between the clusters is minimal. The purpose of this communication is to ensure that one hill is explored only by one hill climber. UEGO makes periodic attempts to find new hills to climb. Empirical results are also presented which include an analysis of the effects of the user-given parameters and a comparison with a hill climber and a GA.

3.1 Introduction

In this section a short introduction to the history and motivation behind developing UEGO is presented, but first let us state what the acronym means. UEGO stands for *Universal Evolutionary Global Optimizer*. However, it must be admitted from the start that this name is not over-informative, and the method is not even 'evolutionary' in the usual sense. In spite of this we have kept the name for historical reasons.

3.1.1 Roots

The predecessor of UEGO was GAS, a steady-state genetic algorithm with subpopulation support. For more details on GAS the reader should consult Chapter 2 or (Jelasity and Dombi, 1998). Let us note however that this chapter is self contained and does not assume any further knowledge about GAS; it will not be mentioned outside of this section only in connection with the empirical comparison results.

GAS has several attractive features. Perhaps the most important of them is that it offers a solution to the so-called niche radius problem which is a common problem of many simple niching techniques such as *fitness sharing* ((Deb, 1989) or (Deb and Goldberg, 1989)), *simple iteration* or the *sequential niching* (Beasley et al., 1993). This problem is related to functions that have multiple local optima and whose optima are unevenly spread throughout the search

space. With such functions the *niche radius* cannot be set correctly since if it is too small the search becomes ineffective and if it is too large those local optima that are too close to each other cannot be distinguished. The solution of GAS involves a 'cooling' technique which enables the search to focus on the promising regions of the space, starting off with a relatively large radius that decreases as the search proceeds.

However, the authors of GAS came in for a number of criticisms, one being that the algorithm was too much complex, and another that parallel implementation turned out to have many pitfalls associated with it.

Although UEGO is based on GAS there are two major differences that were motivated by the need for a better parallel implementation and the requirement of using domain specific knowledge in an effective way.

The structure of the algorithm has been greatly simplified. As a result the parallel implementation is much easier and the basic ideas become more accessible. This is important because, as our results will show, UEGO performs similarly or better than the GA and the simple stochastic hill climber (SHC) on our test problems, and at the same time it can be parallelized better than these methods (Ortigosa, 1999; Ortigosa et al., 2001).

3.1.2 Basic Ideas

The basic idea is that in multimodal optimization problems where the objective function has multiple local optima and the structure of these optima should be discovered beside the global optimum, it may be useful to ensure that the optimizer does not waste its time exploring the same region multiple times but simultaneosly new and promising regions are found. This goal can be achieved by applying a non-overlapping set of clusters which define sub-domains for the applied optimizer. Based on the results of the optimizer, the search process can be directed towards smaller regions by creating a new set of non-overlapping clusters that consists of smaller sub-domains. This process is a kind of cooling method similar to simulated annealing. A particular cluster is not a fixed part of the search domain; it can move through the space as the search proceeds. The non-overlapping property of the set of clusters is maintained however.

UEGO is abstract in the sense that the 'cluster-management' and the cooling mechanism has been logically separated from the actual optimization algorithm, so it is possible to implement any kind of optimizers that work inside a cluster. This allows the adaptation of the method to a large number of possible search domains using existing domain specific optimizers while enjoying the advantages of having muliple non-overlapping clusters which ensures that search effort is focused on interesting regions.

Here an SHC shall be utilized as the optimizer algorithm. This choice is supported by results that show that the performance of the SHC is similar to that of the GA in many cases and sometimes may even be better (e.g. (Mitchell et al., 1994; Juels and Wattenberg, 1996; Yagiura and Ibaraki, 1996; Ishibuchi et al., 1997)). In (Eiben et al., 1998b) a GA with very small population size (1) has been suggested for the graph coloring problem, which is in fact an SHC. Our results confirm that the SHC can indeed outperform the GA at least on the problems and parameter settings we considered.

3.1.3 A Note on Terminology

In the following sections the term *species* will be used instead of e.g. *cluster, zone, region, subdomain* etc. This may be strange since in evolutionary computation this term normally means a population of similar individuals while here it denotes a subset of the search domain. This is not a major problem however; a species in our sense is nothing else but the set of all possible members that are similar according to some similarity measure which is in fact a function of e.g. the application domain. We think that the behavior of a species as will be defined later has strong biological analogies.

The actual number of solutions in a species is given by the applied optimizer. In the case of SHC it is one but e.g. in the case of a GA it may be larger.

3.1.4 Outline of the Chapter

Section 3.2 describes UEGO; the basic concepts, the general algorithm and the theoretical tools that are used to set the parameters of the system based on a few user-given parameters. Sections 3.3 and 3.5 discuss the experimental results that describe the effects of these parameters of the algorithm on the quality of the results and compare UEGO with a simple GA (GAS), a stochastic hill climber (SHC) and a multistart hill climber (MHC) using a set of test functions. Section 3.6 then provides a short summary.

3.2 Description of UEGO

In this section the basic concepts, the algorithm, and the setting of the parameters are outlined. In UEGO, a domain specific optimizer has to be implemented. Wherever we refer to 'the optimizer' we mean this optimizer.

3.2.1 Basic Concepts

In the following it will be assumed that the parameters of the function take values from the same interval. This is easy to achieve for any function via normalization.

A key notion in UEGO is that of a *species*. A species can be thought of as a window on the whole search space. This window is defined by its *center* and a *radius*. The center is a solution, and the radius is a positive number. Of course, this definition assumes a *distance* defined over the search space. The role of this window is to 'localize' the optimizer which is always called by a species and can 'see' only its window, so every new sample is taken from there. This means that the largest step made by the optimizer in a given species is no larger than the radius of the given species. If the value of a new solution is better than that of the old center, the new solution becomes the center and the window is moved.

The radius of a species is not arbitrary; it is taken from a list of decreasing radii, the *radius list*. The radii decrease in a regular fashion in geometrical progression. The first element of this list is always the diameter of the search space which will ensure that the largest species always contains the whole space independently of its center. The diameter is given by the

```
uego
        init_species_list()
        optimize_species( n[1] )
        for i = 2 to levels
              create_species(new[i]/length(species_list))
              fuse_species( r[i] )
              shorten_species_list( max_spec_num )
              optimize_species( n[i]/max_spec_num )
              fuse_species( r[i] )
        rof
ogeu
```

Figure 3.1: The basic algorithm of UEGO.

largest distance between any two possible solutions according to the distance mentioned above. If the radius of a species is the ith element of the list, then we say that the *level* of the species is i.

During the optimization process, a list of species is kept by UEGO. The algorithm is in fact a method for managing this *species-list* (i.e. creating, deleting and optimizing species); it will be described in Section 3.2.2.

3.2.2 The Algorithm

Firstly, some parameters of UEGO will be very briefly mentioned more details of which can be found in Section 3.2.3.

As we mentioned earlier, every species has a fixed level during its lifetime. Species-level operators may change this level however as will be described. The maximal value for the level is given by a parameter called `levels`. Every valid level i (i.e. for levels from $[1,$`levels`$]$) has a radius value (r_i) and two function evaluation numbers. One is used when new species are created at a given level (new_i) while the other is used when optimizing individual species (n_i). To define the algorithm fully, one more parameter is needed: the maximal length of the above-mentioned species list (`max_spec_num`).

The basic algorithm is shown in Figure 3.1. Now the procedures called by UEGO will be described.

`Init_species_list.` Create a species list consisting of one species with a random center at level 1.

`Create_species(evals).` For every species in the list, create random pairs of solutions in the 'window' of the species, and for every such pair take the middle of the *section* connecting the pair. If the objective function value of the middle is worse than the values of the pair, then the members of the pair are inserted in the species list. Every new inserted species is assigned the actual level value (`i` in Figure 3.1).

The motivation behind this method is simple: to create species that are on different 'hills' so ensuring that there is a valley between the new species. Of course this is a heuristic only. In higher dimensions it is possible (in fact typical) that many species are created even if the function is unimodal. This is an unlucky effect which is handled by the cooling process to ensure that at the beginning the algorithm does not create too many species capturing only the rough structure of the landscape. The parameter of this procedure is an upper bound of the function evaluations. Note that this algorithm needs a definition of *section* in the search space.

`Fuse_species(radius)`. If the centers of any pair of species from the species list are closer to each other than the given radius, the two species are fused. The center of the new species will be the one with the best function value while the level will be the minimum of the levels of the original species to be fused. Of course this method does not ensure that no species will overlap after fusion though the amount of the overlapping regions is typically highly decreased.

`Shorten_species_list(max_spec_num)`. Delete species to reduce the list length to the given value. Higher level species are deleted first.

`Optimize_species(evals)`. Start the optimizer for every species with the given evaluation number (i.e. every single species in the actual list receives the given number of evaluations). See Section 3.2.1.

It is clear that if for some level i the species list is shorter than the allowed maximal length, `max_spec_num`, the overall number of function evaluations will be smaller than n_i (see Figure 3.1, `optimize_species`). In our implementation we use the difference of the actual number of function evaluations and n_i to find more species. This technique has no effect when there are many species but if the number of species is small, a lot of extra effort is devoted to finding new ones.

Finally, let us make a remark about a possible parallel implementation. The most time-consuming parts of the basic algorithm is the creation and optimization of the species. Note that these two steps can be done independently for every species, so each species can be assigned to a different processor. Note also that a species is defined by its center and its level, so the amount of information used in communications is really small. In GAS algorithm a species is a set of individuals and there are relations among species and even among individuals, so it is quite difficult to send a species or an individual to another processor. The complexity of the possible parallel approach would be high enough. As our experimental results will clearly show, sequential UEGO performs slightly better than the SHC and the GA even when the number of species is as high as 200.

3.2.3 Parameters of UEGO

The most important parameters are those that belong to the different levels: the radii and two function evaluation numbers for species creation and optimization (see Figure 3.1). In this section a method is described which sets these parameters using a few easy-to-understand parameters set by the user. The experimental sections will provide further guidelines on the meaning

and setting of these remaining user-given parameters.

We will now make use of the notation introduced in Section 3.2.2. The user-given parameters are listed below. Short notations (in brackets) that will be used in equations in the subsequent sections are also given.

`evals` **(N):** The maximal number of function evaluations the user allows for the whole optimization process. Note that the actual number of function evaluations may be less than this value.

`levels` **(l):** The maximal level value (see Figure 3.1).

`threshold` **(ν):** The meaning of this parameter will be explained later.

`max_spec_num` **(M):** The maximal length of the species list.

`min_r` **(r_l):** The radius that is associated with the maximal level, i.e. `levels`.

The parameter setting algorithm to be described can use any four of the above five values while the remaining parameters are set automatically.

Speed of the optimizer. Before presenting the parameter setting method, the notion of the *speed* of the optimizer must be introduced. As explained earlier, the optimizer cannot make a larger step in the search space than the radius of the species it is working in. Furthermore if the center of a species is far from every local optimum then these steps will be larger while if the center is already close to a local optimum then the steps will be very small. Given a certain number of evaluations, it is possible to measure the distance the given species moves during the optimization process assuming that the species is suboptimal. This distance can be approximated (as a function of the radius and evaluations) for certain optimizers using ideal landscapes (such as linear functions) with the help of mathematical models or experimental results. This naturally leads to a notion of speed that will characterize a given domain (assuming e.g. a linear landscape) and will depend on the species radius. Speed will be denoted by $v(r)$. As we will not give any actual approximations here, the reader should refer to Chapter 2.

The parameter-setting method is based on *principles* that are supposed to be intuitive and reasonable. The main reason of using such principles is to reduce the number of parameters to a small set in which every parameter has a clear meaning. These principles are now described below.

Principle of equal chance. At a level, every species moves a certain distance from its original center due to optimization. This principle ensures that every species will receive the number of evaluations that is enough to make it move at least a fixed distance assuming that the speed of this motion is $v(r)$. A species will not necessarily move that far but the definition of speed is such that if the species is far from the local optima then it will move approximately the given distance. This common distance is defined by $r_1\nu$. The meaning of `threshold` can

now be given: it directly controls the distance a species is allowed to cover, so it actually controls the probability that they will eventually represent a local optimum: the further a species can go the higher the probability of reaching a local optimum is (and the more expensive the optimization is). Recall that r_1 is always the diameter of the search space. Now the principle can be formalized:

$$\frac{v(r_i)n_i}{M} = r_1\nu \quad (i = 2, \ldots, l) \tag{3.1}$$

Principle of exponential radius decreasing. This principle is quite straightforward; given the smallest radius and the largest one (r_l and r_1) the remaining radii are expressed by the exponential function

$$r_i = r_1(\frac{r_l}{r_1})^{\frac{i-1}{l-1}} \quad (i = 2, \ldots, l). \tag{3.2}$$

Principle of constant species creation chance. This principle ensures that even if the length of species list is maximal, there is a chance of creating at least two more species for each old species. It also makes a strong simplification, namely that all the evaluations should be set to the same constant value.

$$new_i = 3M \quad (i = 2, \ldots, l) \tag{3.3}$$

Now we can develop a method to determine all the parameters from the user given parameters. First let us express N (the number of all function evaluations available) using the parameters of the algorithm. Let us define $new_1 = 0$ for the sake of simplicity since new_1 is never used by UEGO. We can write the equation

$$\sum_{i=1}^{l}(n_i + new_i) = (l-1)3M + \sum_{i=1}^{l} n_i = N \tag{3.4}$$

making use of (3.3) in the process. One more simplification is possible; set $n_1 = 0$ whenever $l > 1$. Note that if $l = 1$ then UEGO reduces to the optimizer it uses for optimizing the species. Expressing n_i from (3.1) and substituting it into (3.4) we can write

$$(l-1)3M + \sum_{i=2}^{l} \frac{Mr_1\nu}{v(r_i)} = N \tag{3.5}$$

Using (3.2) as well, it is quite evident that the unknown parameters in (3.5) are just the user given parameters and due to the monotony of this equation in every variable, any of the parameters can be given using effective numerical methods provided the other parameters are known. Using the above principles the remaining important parameters (n_i, new_i and r_i) can be evaluated as well. Note however that some of the configurations set by the user may be infeasible.

3.3 Experiments with Real Functions

In this section experimental results on real functions will be presented. Due to the stochastic nature of UEGO, all the numerical results given in this work are average values of fifty executions, obtaining an enough statistic sample of experiments. From this data set the corresponding confidence intervals (95%) were computed (see (Sokal and Rohlf, 1981)). These confidence intervals have not been represented because it would have messed up the plots since there are multiple curves in each graphic, and also these intervals were too narrow, so they could not be distinguished from the average values.

In this section, a set of four test functions and two classical multimodal functions (Griewank, Rastrigin) have been used to evaluate UEGO. These test functions have different characteristics w.r.t. dimensionality and the number of local optima so, it is possible to illustrate the effect of these characteristics on the performance of UEGO. Comparisons with the ancestor of UEGO, GAS (see Chapter 2) and the simple and multistart version of a hill climber (SHC and MHC) described in (Solis and Wets, 1981) will also be given.

3.3.1 Test Problems

A first experimental stage was carried out on a set of four new defined functions. We chose not to use well-known benchmark functions for testing. The reason for this is that we agree with the ideas discussed in (Hooker, 1995), namely that for doing scientific tests it is more convenient to use functions that differ only in controllable features. This allows the analysis of the effect of only one separated feature of the test problem, e.g. the number of local optima. There is another reason: using widely accepted benchmark problems prevents developing methods that perform well only on special kind of problems. We believe that it is more important to characterize the ideal problem for optimization methods than trying to show that they outperform other methods on as many benchmark problems as possible. In a second stage, UEGO was evaluated and compared to other algorithms, using Griewank and Rastrigin functions (see Section 3.4.1).

In the experiments to be discussed here we wanted to examine the effects of dimensionality and the number of local optima. Therefore we used four test functions that are characterized in Table 3.1. The construction of these functions starts with a user-given list of local optimum

Table 3.1: Characteristics of the four test functions.

	F1	F2	F3	F4
Type	$[0,1]^2 \rightarrow \mathbb{R}$	$[0,1]^2 \rightarrow \mathbb{R}$	$[0,1]^{30} \rightarrow \mathbb{R}$	$[0,1]^{30} \rightarrow \mathbb{R}$
# of maxima	5	125	5	125

sites (o) and the corresponding function values (f_o). All the function values have to be positive. In the first step, we define bell shapes for every site to create the local optima. The height of a bell is given by the function value f_o of its site o, and its radius r is the distance of o from the

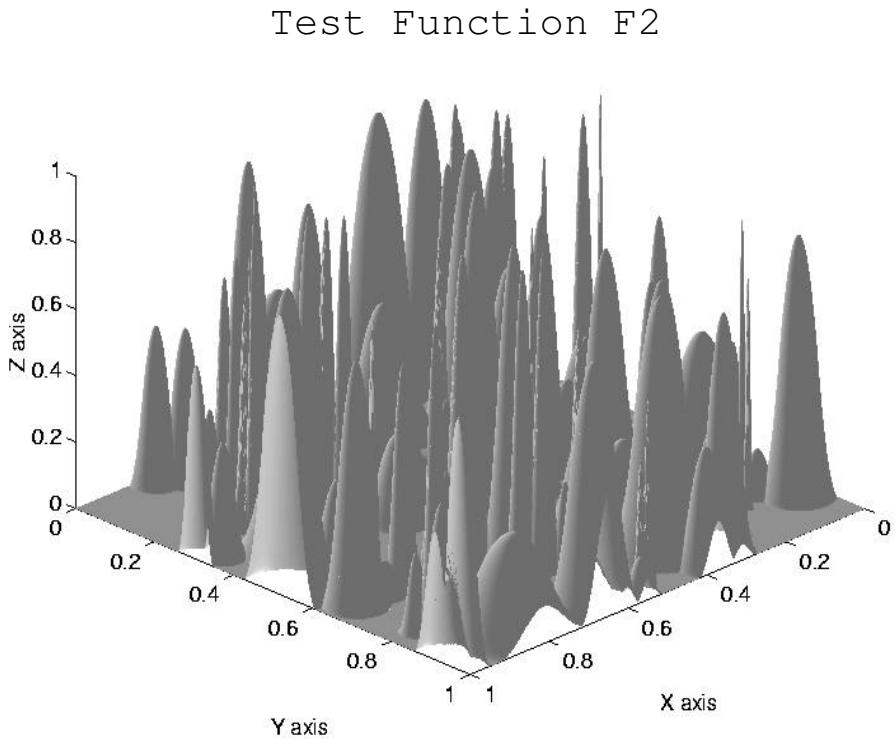


Figure 3.2: The plot of the test function F2.

closest site. The height of the bell at a distance x from o is $f_o g(x)$, where:

$$g(x) = \begin{cases} 1 - \frac{2x^2}{r^2} \text{ if } x < \frac{r}{2} \\ \frac{2(x-r)^2}{r^2} \text{ if } \frac{r}{2} \leq x < r \\ 0 \text{ otherwise} \end{cases}$$

The objective function is the sum of these bells. In the case of our test functions, the coordinates of the maximum sites and their values were randomly taken from $[0, 1]$ using a uniform distribution. The two-dimensional function F2 has been drawn in Figure 3.2.

3.3.2 The Optimizer and the Settings

For real function optimization the optimizer used by UEGO was the hill climber suggested in (Solis and Wets, 1981). The parameters of the hill climber algorithm were set as in (Solis and Wets, 1981). The parameter ρ_{ub}, that controls the maximal step size, was set to the radius of the species from which the optimizer is called. The accuracy of the search was set to $\min(\rho_{ub}/10^3, 10^{-5})$. No fine tuning of the parameters of the optimizer was done.

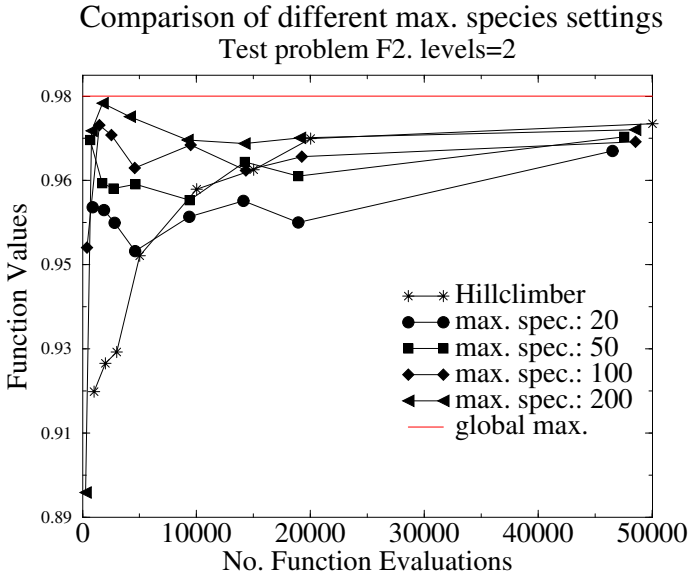


Figure 3.3: A sample from the results of our preliminary experiments.

3.3.3 The Experiments

In our preliminary experiments the minimal radius (`min_r`) was calculated automatically, the threshold (ν) was fixed to 1 in every run, and the effects of the remaining parameters were examined. However, it soon became clear that it was not the best choice: in the case of F2 the performance often decreased as the number of evaluations was increased. This effect can be seen in Figure 3.3, where the average of the most-fit species values of the objective function (for fifty runs of UEGO) as a function of the average value of the number of function evaluations has been represented (notice that the number of function evaluations is always less than `evals`). The reason of this strange behavior is that with the increasing number of evaluations the minimal radius became smaller and smaller according to principles described in Section 3.2.3, so the search slowed down. Another observation was that the performance seemed to be the best when the number of levels was two.

Due to these results we decided to examine the effect of the minimal radius and the maximal number of species with the number of levels fixed to two. The set of the tested values are shown in Table 3.2. Experiments were performed for all combinations of these parameter settings.

In these experiments, the behavior of UEGO was rather similar for functions F1 and F3, and all the tested values of minimal radii (`min_r`) resulted in almost identical performance, but the results were very sensitive to the values of maximal number of species (`max_spec_num`). The performance decreased with the increasing value of `max_spec_num`. In our experiments the optimal value for the `max_spec_num` parameter was the minimal (20). For F4, changes on the performance of UEGO w.r.t. `min_r` are almost negligible, as happens on F1 and F3.

On the other hand, for F2 the value of `min_r` did make a difference, mainly when the maximal number of species was relatively small and the performance of UEGO was very robust

Table 3.2: Tested values of UEGO parameters for the real functions.

evals	levels	max_spec_num	min_r	threshold
1000, 2000, 3000, 5000, 10000, 15000, 20000, 50000	fixed to 2	20, 50, 100, 200	.003, .005, .01, .03, .05, .08	automatically set

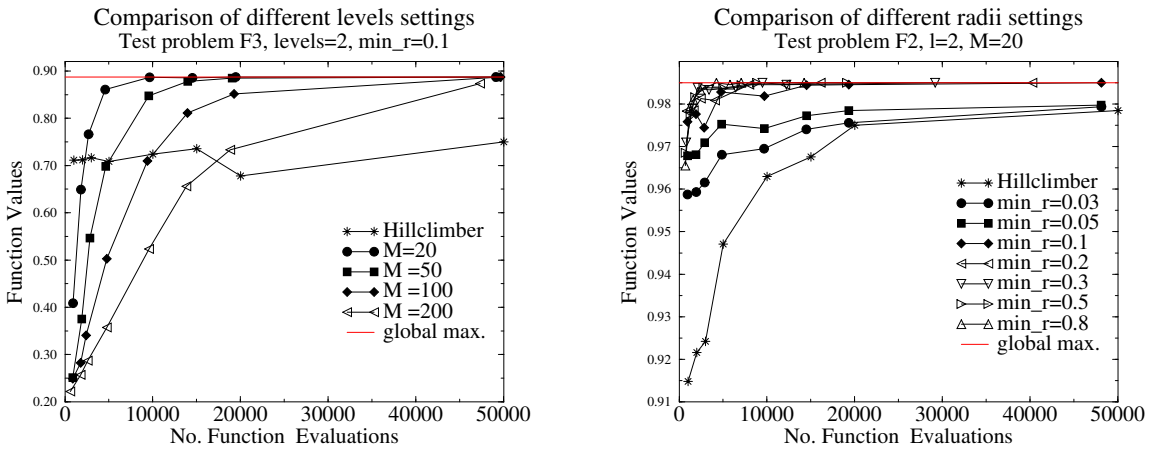


Figure 3.4: The effect of the different maximal species numbers on F3 and the different radii on F2.

for the greatest value of `min_r` (0.8), especially for `max_spec_num` equal to 20 and 50. Two characteristic plots illustrating these effects are shown in Figure 3.4.

The other goal was to find evidence that using more levels than two can be useful. To achieve this goal several values of `level` (2, 3, 5 and 10) were examined with the maximal species number fixed to 20. The experiments were performed for F2 since it was the only function on which the value of `min_r` had significant effect. The result clearly showed that with the optimal minimal radius UEGO was fairly robust w.r.t. the number of levels. However, when the minimal radius was set to a much smaller value than the optimum, higher values of `levels` outperformed the lower settings. This effect is illustrated in Figure 3.5. The first interesting phenomenon that needs explanation is that in the first set of experiments, results on F1, F3 and F4 were very similar w.r.t. `min_r` while on F2 we got a very different behavior. The first idea that comes to mind is that F1 and F3 have few local optima so it is quite reasonable that setting large maximal number of species would result in a poor performance since most of the evaluations are devoted to search for the non-existent peaks while this problem does not exist in the case of F2. This would also explain that the different values of `min_r` had no effect: most of the search was in fact random search.

For F4 test function UEGO needs more than 50,000 function evaluations for reaching the global optimum. For a small number of species, the algorithm converges relatively fast to a

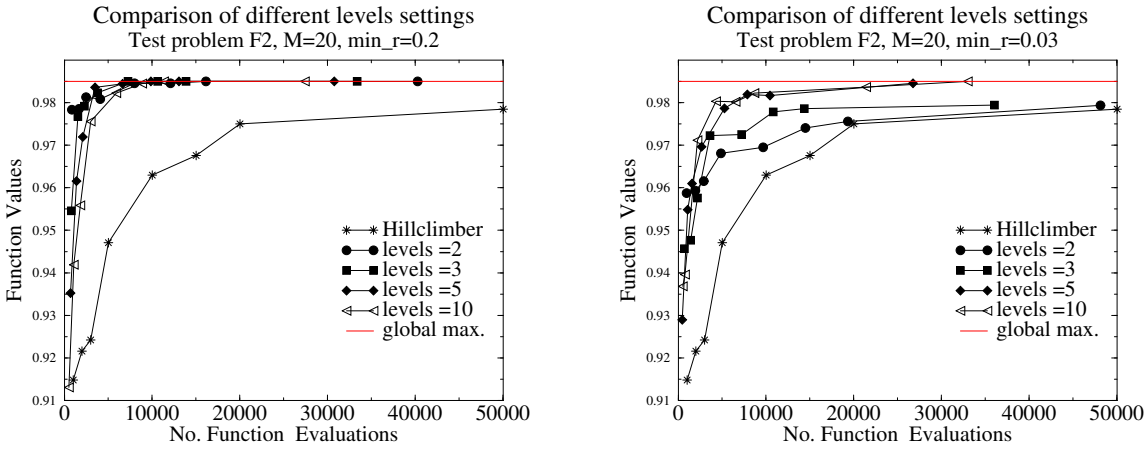


Figure 3.5: The effect of the different number of levels with the optimal and with a much smaller radius.

solution, but this solution is a local one. However, when the maximal number of species is great (100, 200), UEGO spends more function evaluations on reaching a solution, but this solution will be a global solution. For these values of maximal species number results in Table 3.3 will show that the algorithm reaches the real optimum in 40% of executions for `evals`=50000 function evaluations. Remember that F4 is a hard problem (30 dimensions and 125 local optima).

On the other hand, the optimal values of `max_spec_num` for F1 and F3 are 20. So, it seems that for very hard problems is convenient to use a large number of species in order to ensure the convergence to the solution. Therefore we suggest that UEGO should be used for optimizing highly multimodal functions because it ensures easy species creation.

The second phenomenon, namely that it is reasonable to use more levels if the minimal radius is smaller than the optimal minimal radius (for `levels`=2), is much easier to explain. The "cooling" mechanism of UEGO ensures that if the maximal level is high then in some phase of the search the radius is very likely to be near the optimal value while if the maximal level is 2 then the radius immediately becomes the value set by the user. This property is very useful since setting a higher maximal level may ensure a higher degree of robustness.

3.4 Comparing algorithms

In this section comparisons with a simple hill climber (SHC), a multistart hill climber (MHC) and GAS will be shown. The parameters of SHC, MHC and GAS algorithms were set as follows. The hill climber (SHC) was the optimizer used by UEGO; it means that SHC is UEGO with `levels`= 1. In the multistart case the number of restarts from a new random point was given by the optimal value of the `max_spec_num` for UEGO. The parameters for GAS are very similar to those of UEGO, so they were set to the Best UEGO parameters for every problem using `max_spec_num` as population size.

Results for UEGO were obtained with `levels` = 2 in every case. `max_spec_num` = 20,

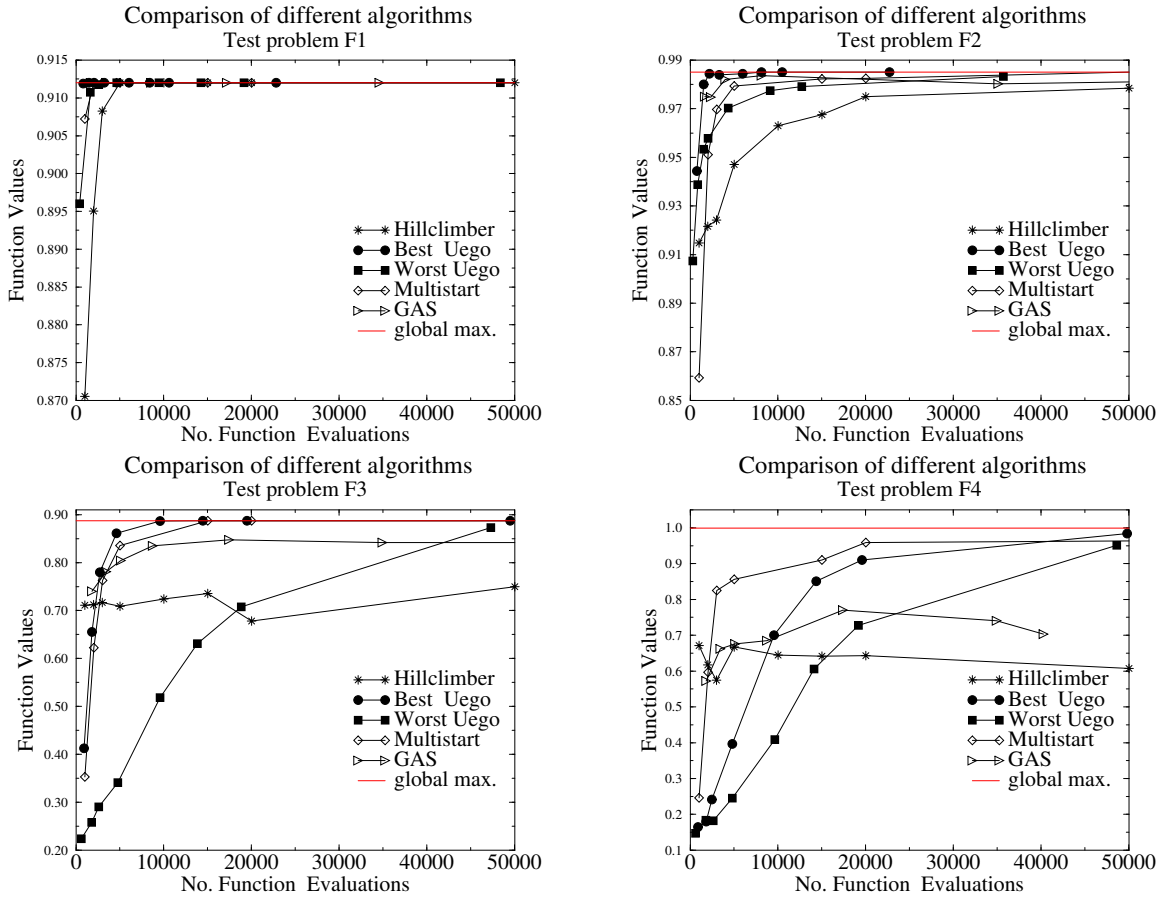


Figure 3.6: Comparison of the hill climber, the multistart hill climber, GAS and UEGO.

`min_r` = 0.8 for F1, and F3 test functions, and `max_spec_num` = 100, `min_r` = 0.2 for F2 and F4 test functions were the setting parameters for the results named as Best UEGO. Parameters for Worst UEGO were `max_spec_num` = 200, `min_r` = 0.003 for F1, and F3 test functions, and `max_spec_num` = 20, `min_r` = 0.03 for F2 test function, and `max_spec_num` = 200, `min_r` = 0.03 for F4 test function. Numerical results of these comparison experiments are shown in Figure 3.6 and Table 3.3.

Table 3.3: Comparison of the UEGO, the multistart hill climber and GAS. % means the percent of the runs reaching the global optimum.

UEGO			MHC			GAS		
Eval.	N.Spec.	%	Eval.	N.Spec.	%	Eval.	N.Spec.	%
F1, `levels`=2, `max_spec_num`=20, `min_r=0.8`								
811	2.62	88.5	1000	3.56	2	1673	2.30	76.4
1482	2.74	99.1	2000	3.42	98	3410	2.50	98.2
2062	2.86	100	3000	3.14	100	5075	2.54	100
3228	2.88	100	5000	2.84	100	8549	2.50	100
6038	2.94	100	10000	2.72	100	17194	2.76	100
8411	2.98	100	15000	2.46	100	26743	2.78	100
10621	3.24	100	20000	2.28	100	34580	2.82	100
22826	3.32	100	50000	1.02	100	86435	2.78	100
F2, `levels`=2, `max_spec_num`=100, `min_r=0.2`								
751	7.12	0	1000	8.22	0	1631	0.94	0
3020	7.44	0	2000	10.04	0	2371	1.42	0
2049	7.40	3	3000	12.16	0	3343	3.44	0
3019	7.82	48	5000	9.98	4	5997	5.12	0
5151	7.94	96	10000	7.86	56	8120	7.44	0
6228	8.52	100	15000	6.02	84	25276	9.32	0
17152	8.96	100	20000	4.38	100	35122	11.38	20
19246	9.74	100	50000	5.24	100	87930	11.46	28
F3, `levels`=2, `max_spec_num`=20, `min_r=0.8`								
918	2.96	0	1000	20*	0	1818	1.06	0
1807	3.28	0	2000	20*	0	3428	1.08	0
2708	3.54	0	3000	20*	0	5108	1.08	0
4597	3.72	42	5000	20*	0	8748	1.16	0
9596	3.98	68	10000	7*	0	17428	1.16	0
14467	4.64	98	15000	4.88	98	24914	1.46	10
19492	4.94	100	20000	4.86	94	37365	1.28	10
49477	4.98	100	50000	4.98	100	87569	1.00	18
F4, `levels`=2, `max_spec_num`=100, `min_r=0.2`								
899	5.8	0	1000	100*	0	1817	1.28	0
1808	6.5	0	2000	100*	0	3419	2.16	0
2455	8.0	0	3000	100*	0	5102	1.78	0
4812	10.4	0	5000	100*	0	8746	1.32	0
9564	11.8	0	10000	100*	0	17407	2.24	0
14350	12.6	10	15000	100*	0	28922	1.48	0
19587	15.0	16	20000	82*	2	34793	1.16	0
49804	20.4	40	50000	65*	4	40256	0.60	0

Table 3.3 shows average values of the number of function evaluations, number of local or global optima found (species) and the percentage of success in finding the global optimum. Setting parameter values for functions F1, F2, F3 and F4, were chosen as a result of the preliminary experiments. In this table, the symbol * means that most of maxima found were not local nor global maxima.

It can be seen that for function F1, when `evals` ≥ 3000, all the three algorithms reached 100% of success in finding the global optimum, however UEGO was able to find more local optima and it was at least 60% computationally less expensive than MHC and GAS algorithms. For function F3, with only five maxima, GAS was able to find the global optimum in a very few runs (18% success), while UEGO and MHC were fully successful when `evals`=50,000. They found all the local and global optima in most of the runs. Similar results were obtained for function F2, but in this case UEGO outperformed MHC in the number of species (local optima). Finally, results for the hardest function F4, clearly show that though UEGO did obtain only 40% of success in finding the global optimum, it was able to find 20 real local optima while MHC and GAS failed in all the cases.

In short, it can be said that UEGO is slightly better than the multistart hill climber for F1, F2 and F3 and for problem F4, UEGO is the only technique that reaches the global optimum. From Figure 3.6, it is clear that Best UEGO outperforms SHC and GAS for all the functions.

These results indicate that it is reasonable to use the clustering technique to create starting points since the performance does not decrease but, unlike the multistart hill climber, UEGO provides a great number of reasonably good solutions that are at least as far from each other as the minimal radius (see Table 3.3). This property is very useful in several applications, for example in decision problems: the expert decision maker has a lot of good solutions to choose from. Also remember that the number of restarts of the hill climber was optimal. GAS is also outperformed which justifies our efforts to eliminate the drawbacks of GAS.

3.4.1 Results for Griewank and Rastrigin functions

In this section some comparisons of above algorithms for Griewank and Rastrigin functions will be shown. First of all we tried to find the best parameters for UEGO, so we decided to test UEGO using several values of `max_spec_num`, `levels` and `min_r` (in this case, values of `min_r` are normalized to the domain of definition of each function). The ranges of the parameters are shown in Table 3.4. Experiments were performed for all combinations of these parameter settings, and the threshold was automatically set.

Table 3.4: The values of the UEGO parameters for the test functions.

evals	levels	max_spec_num	min_r
10000, 15000, 20000, 30000, 50000, 200000	2, 3, 5, 10	5, 10, 50, 200	0.1, 0.2, 0.5, 0.8,

Both functions are ten-dimensional and they have several optima, so these test functions are

hard multimodal problems. The best performance over these test functions was reached when the number of `levels` was 10 because in this case the convergence is rather slow and the probability for finding the best solutions is high. The optimal `max_spec_num` tends to be high. Particularly, for comparisons we chose the values 50 and 200 for Griewank and Rastrigin test functions respectively. The optimal `min_rad` was 0.1 and 0.2, respectively.

Table 3.5 shows some comparisons among UEGO, MHC and GAS. For each algorithm the number of function evaluation (Eval), the most-fit species function value (Val) and the number of species found (Sp.) are shown. It is interesting that the real optimum (whose value is zero) was not found for any of the algorithms, for any of the parameter combinations tested. However, UEGO found the global optimum when more function evaluations were allowed, i.e. for Rastrigin test function, UEGO needs about 25,000,000 function evaluations to find the global optimum. We tried to find the optimum using MHC and GAS algorithms as well, but they did not reach it in those experiments, even when the number of function evaluations was higher than 100,000,000. For Griewank function, the three algorithms found the global optimum in 1,000,000 function evaluations. The number of global and local optima found by UEGO and MHC were quite similar.

To summarize the results: it has been demonstrated that both the clustering and the level-based "cooling" techniques of UEGO show some advantages over its predecessor GAS. On the other hand, UEGO can be parallelized almost as effectively as the multistart hill climber. Without going into the details we mention that a parallel version of UEGO using a simple asynchronous Master Slave model has been implemented and executed on a Cray T3E using up to 33 processors (Ortigosa, 1999; Ortigosa et al., 2001) and for a set of eleven test functions an almost linear speed up was obtained. Using 33 processors the values of the speed up range between 27 and 32.

3.4.2 A Note on Parameter Setting

As shown clearly by the previous sections, there is no unique best way to set the parameters of UEGO. The optimal setting depends on the problem structure. This fact is not surprising since the heuristics used in the algorithm make explicit assumptions about the structure of the domain. It is not special to UEGO either since general theoretical results about the relationship between search domains and optimizers (Wolpert and Macready, 1997) predict this effect.

If information is available on the number and distribution of local optima of the problem to be solved then it is possible to set the parameters based on the design principles of UEGO and on the empirical results discussed in Section 3.3.3. If there are many local optima then the maximal number of species should be high and if they are close to each other then the minimal radius should be small. Of course, if the structure of the problem is unknown then preliminary experimentation is necessary. One possible strategy suggested by our experiments is to use a small minimal radius, more levels, e.g. 10, and to use a high maximal number of species, e.g. 200. The number of species created during the optimization gives a hint about the number of local optima, though this is only a heuristic since e.g. having many species is not sufficient nor necessary for having many local optima especially if the number of function evaluations is small.

Table 3.5: Comparisons for Griewank and Rastrigin Test Functions

UEGO			MHC			GAS		
Eval.	Val.	Sp.	Eval.	Val.	Sp.	Eval.	Val.	Sp.
Griewank Test Function								
2153	-0.1891	42.34	10000	-0.4805	50	15003	-0.4738	6.70
3434	-0.0874	42.96	15000	-0.3260	50	23876	-0.3658	10.54
4400	-0.0591	46.86	20000	-0.1797	50	33129	-0.2545	11.86
6267	-0.0315	48.84	30000	-0.0630	50	54823	-0.1258	11.78
17717	-0.0236	49.10	50000	-0.0200	50	85640	-0.0914	11.70
33057	-0.0103	49.18	200000	-0.0180	50	231462	-0.0243	11.80
Rastrigin Test Function								
5043	-49.49	200	10000	-55.61	200	15311	-27.59	20.92
5686	-45.01	200	15000	-42.46	200	24214	-22.25	49.76
6757	-45.63	200	20000	-35.71	200	33289	-18.98	49.78
8467	-47.29	200	30000	-34.69	200	55122	-16.76	49.70
11918	-41.77	200	50000	-25.87	200	85746	-14.40	49.42
36827	-19.23	200	200000	-20.92	200	235875	-14.23	49.68

3.5 Experiments with the Subset Sum Problem

In this section we will discuss the performance of UEGO on an NP-complete combinatorial optimization problem: the subset sum problem. A comparison with GENESIS (Grefenstette, 1984) and a hill climber will be presented. As another result of the experiment the behavior of the parameters of UEGO will be illustrated.

3.5.1 Problem and Coding

In the case of the subset sum problem we are given a set $W = \{w_1, w_2, \ldots, w_n\}$ of n integers and a large integer M. We would like to find a $V \subseteq W$ such that the sum of the elements in V is closest to, without exceeding, M. This problem is NP-complete. Let us denote the sum of the elements in W by SW.

We created our problem instances in a similar way to the method used in (Khuri et al., 1993). The size of W was set to 50 and the elements of W were chosen randomly with a uniform distribution from the interval $[0, 10^{12}]$ instead of $[0, 10^3]$ (as was done in (Khuri et al., 1993)) to obtain larger variance. According to the preliminary experiments, the larger variance of W results in harder problem instances which is important since comparing methods on almost trivial problems makes little sense. The problem instance used here turned out to be so hard that none of the methods employed could find an optimal solution. Based on the results of

Table 3.6: The values of the UEGO parameters for the subset sum problem.

evals	levels	max_spec_num	threshold	min_r
3000, 10000, 30000, 100000, 300000	2, 3, 5, 10	5, 10, 20, 40, 100, 200	automatically set	fixed to 1

Section 4.2, M was set to $SW/2$. As will be shown, this is the most GA-friendly setting, so there is no bias against GENESIS introduced by the problem instance.

We used the same coding and objective function as suggested in (Khuri et al., 1993). For a solution ($x = (x_1, x_2, \ldots, x_{50})$),

$$f(x) = -(a(M - P(x)) + (1 - a)P(x))$$

where $P(x) = \sum_{i=1}^{50} x_i w_i$, and $a = 1$ when x is feasible (i.e. $M - P(x) \geq 0$) and $a = 0$ otherwise. Note that the problem is defined as a *maximization* problem.

3.5.2 The Optimizer and GA Settings

In UEGO, the optimizer was chosen to be a simple SHC as was discussed in the Introduction. In our implementation the SHC works as follows: mutate every bit of the solution with a given probability (but mutating one bit at least), evaluate the new solution and if it is better than or equal to the actual solution, it becomes the new actual solution. This type of SHC worked best in (Mitchell et al., 1994), as well. The mutation probability was set at $4/n$ where n is the chromosome length. This value was the same in all the experiments carried out including those with GENESIS. The other GA parameters were a population size of 50, 1-point crossover with probability 1, and elitist selection.

3.5.3 The Experiments

One of the two main goals of these experiments was to analyze the effects of the user-given UEGO parameters described in Section 3.2.3. To perform this analysis, several values were chosen for each parameter (see Table 3.6) then UEGO was run 50 times for every possible combination of these values. This meant that $5 \cdot 4 \cdot 6 \cdot 50 = 6000$ experiments were performed for one problem instance. Three problem instances were examined but since the results were similar in each case, only one problem instance is discussed below. Figure 3.7 shows the effects of the different parameter settings. As the plots are typical it was inferred that the parameters of UEGO must be fairly robust for this particular problem class. It has to be noted that this does not automatically imply that similar robustness would be observed on other domains as well (see Section 3.4.2).

The other goal of the experiments was to make a comparison. Figure 3.8 shows the relevant results. Note that it was difficult to select the best and the worst performance because the curves

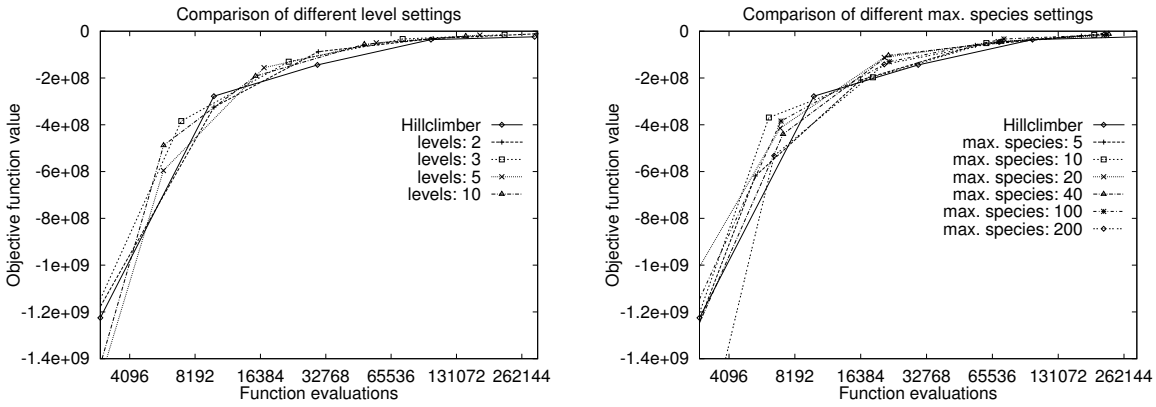


Figure 3.7: With the various level settings, `max_spec_num` is 100 and for the different max. species settings `levels` is 3.

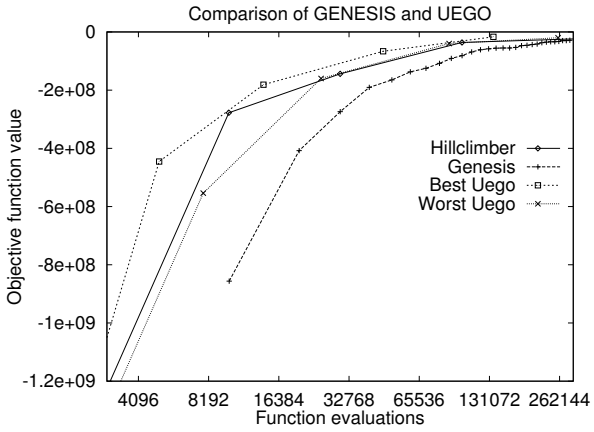


Figure 3.8: The parameters for the best UEGO were `max_spec_num`=20 & `levels`=10, and for the worst `max_spec_num`=5 & `levels`=2.

cross, but the plots give a good approximation. Here SHC is simply UEGO with the setting of `levels`=1.

3.6 Summary

In this section UEGO, a general technique for accelerating and/or parallelizing existing search methods was discussed. As was shown, most of the parameters of the system are hidden from the user due to an algorithm for calculating those parameters from a couple of simple parameters. This algorithm is based on *principles* stated in Section 3.2.3 and the *speed* of the applied optimizer.

Experimental results were given for real and combinatorial problems. It was shown that the

user-given parameters are robust in the case of the subset sum problem and the advantages of the UEGO clustering technique and the level-based "cooling" technique were demonstrated on the real domain.

Other experimental results were also given, such as the comparison of the technique with several methods. In the case of the subset sum problem it was shown that UEGO is slightly better than a GA and an SHC. On the real domain UEGO outperformed the multistart hill climber in the sense that either the quality of the global optimum was better or the number of local optima found by UEGO was larger. GAS was outperformed w.r.t. both aspects. The later indicates that the modifications in order to eliminate the drawbacks of GAS has been successful at least in the case of the problems we considered.

Part III

Representation and the Vocabulary of Science

Chapter 4

Real Building Blocks

This chapter contains ideas that are related to possible ways of describing the working of an evolutionary algorithm. This work is useful for illustrating the epistemological problem mentioned in the Introduction, i.e. that for a meaningful description one needs terminology which might not be available *a priory*. The subset sum problem is used troughout for illustration. The first two sections describe two ways of looking at the search process and the last section tackles a slightly different problem: search space visualization for explaining problem difficulty.

4.1 Implicit formae in Genetic Algorithms

This section discusses the new term *implicit forma*, which is useful for explaining the behaviour of genetic algorithms. Implicit formae are special predicates over the chromosome space. These prediactes are not defined as schemata of the representation at hand though they depend on the representation. The new term is a generalization of the concept of formae such that every approach connected to formae (e.g. fitness distribution) is also relevant to implicit formae. After a short theoretical discussion, three examples are given for illustration, including the subset sum problem which is NP-complete.

4.1.1 Introduction

An understanding of how genetic algorithms (GAs) work is of major importance from the point of view of both theory and application. For a long time, the concept of schemata played the central role in GA theory (Holland, 1975). However, it is now clear that this concept is itself not enough for a prediction of the behaviour of the GA (White and Flockton, 1995); at least some generalization of the concept is necessary for both binary and general representations.

For general representations, the concept of formae (e.g. (Radcliffe, 1992)) has been introduced. This approach is especially useful in the design of genetic operators, but it should be mentioned that formae are very similar to schemata in the sense that they are strongly connected to the representation at hand (though the representation is normally designed using the previously chosen formae).

This section shows that besides the carefully designed formae there are other factors are also important when modeling the search. These factors seem to be treated as properties of formae in the literature, e.g. the variance of fitness (Radcliffe and George, 1993; Radcliffe and Surry, 1995) or noise (Kargupta, 1995). While understanding that these are useful tools for gaining information about the quality of a given representation of the problem at hand, we try to provide a deeper insight into the search process by introducing the term *implicit forma*. We believe that this approach will help in predicting and especially in explaining the behaviour of GAs in several problem classes, including real-world applications.

In section 4.1.2, we give a brief introduction to forma analysis, restricting ourselves only to the basic definitions, and implicit formae are then discussed. In section 4.1.3, the new term is illustrated through three case studies. One of the examples is the subset sum problem, an NP-complete combinatorial problem. The relation of the GA and the bit-hillclimber algorithm is also discussed on the basis of implicit formae.

4.1.2 Formae and Implicit Formae

Formae

A discussion of formae is needed only to make it clear why the name *implicit forma* has been used to denote the properties discussed here. Thus, a very basic knowledge suffices. A detailed description can be found in (Radcliffe, 1992).

A representation maps the solution space S to a chromosome space C. Usually, every $x \in C$ can be regarded as an intersection of a set of predicates over C. If $C = \{0,1\}^n$, then these predicates are the schemata of order 1. If C is the chromosome space of the permutation representation of the traveling salesman problem, then these predicates are the subsets of the set of all permutations with a fixed town at a given position. Thus, a set of *alleles* (i.e. predicates that a chromosome may contain) can be assigned to every representation. A *forma* is simply the intersection of a subset of the alleles. The empty set is not a forma. It is clear that if $C = \{0,1\}^n$, then formae reduce to schemata, so a forma is a generalization of the concept of a schema.

From our point of view, the essence of the above definitions is that formae are predicates over the chromosome space C that are closely related to the representation at hand. When the emphasizing the difference between these formae and implicit formae is needed we will use the name *explicit formae*.

Implicit Formae

It has already been shown (Radcliffe, 1991a; Vose, 1991) that every predicate over the space of all chromosomes C behaves according to the schema theorem for some appropriate genetic operators; in other words, its proportion is approximately determined by its observed fitness. The forma analysis is connected to this result, i.e. representation-independent operators are designed (Radcliffe and Surry, 1995) to be "friendly" with the formae given by the representation. For some special GAs schemata have been introduced that are useful in modeling the search. One

example is *relative ordering schemata* (Kargupta et al., 1992) and another comes from *grouping GAs* (Falkenauer, 1994b; Falkenauer, 1994a) where groups can define schemata. These two examples can not be defined using fixed positions and wildcards, though they can be defined using unions of such fixed schemata. However, it is possible that some other *arbitrary* predicates over the chromosome space C are also treated as the formae, i.e. they obey the schema theorem and so are suitable for modeling the search. Examples of this phenomenon will be given in section 4.1.3. The existence of such predicates motivates our central definitions.

The definitions below are relative to a given GA implementation. On this implementation we mean the search space, the encoding of this space and a set of genetic operators (selection, crossover, mutation). However the definitions are independent of the objective function.

Notation 1. Let $\mathcal{P}(C)$ be the set of all predicates over the chromosome space C.

Definition 22 has a central role. It gives a possible property of elements of $\mathcal{P}(C)$ that will be important when talking about the search process.

Definition 22. The *degree of relevance* of a given predicate $P \in \mathcal{P}(C)$ with respect to a GA implementation is $r(= r(P))$ iff during the successive iterations of the GA and P as the objective function ($P(x) = 1$ if $x \in P$, 0 otherwise) starting from an infinite uniformly distributed random population, the proportion of P goes to r as the number of generations goes to infinity, where

$$P(x) = \begin{cases} 1 \text{ if } x \in P \\ 0 \text{ otherwise} \end{cases}$$

Definition 23. A given predicate $P \in \mathcal{P}(C)$ is an *implicit forma* iff $r(P) > p_0$ and is *neutral* iff $r(P) = p_0$, where $p_0 = |P|/|C|$.

This definition gives a possibility for measuring the "disruptive effects" of a given GA implementation on the given subset. It is important that this definition is relative to an implementation of the GA but it is *independent* of any fitness function that will be optimized by this GA. The fitness function used in the definition is fixed. It serves as a tool for grasping the disruptive effects of the GA implementation at hand. This notion can be considered as a generalization of schemata, or rather building block as building blocks are considered to be the subsets that are disrupted less by the GA (short, low order schemata). Of course, some drawbacks can be found here as well. For instance in the actual search process the sampling error must be considered. The rate of relevance is also important and would be worth discussing in more detail, but for our present purposes Definition 23 suffices. The definition of the degree of relevance refers to infinite populations. In practice, the intuitive relevance also depends on the size of the predicate since very small predicates might not get a sample at all.

Though here we focus on the experimental results, for illustration we give an analysis of predicate EVEN without the straightforward technical details.

Definition 24. $C = \{0,1\}^n$, EVEN $\in \mathcal{P}(C)$; EVEN(x) iff the number of 1s in x is even.

Through this simple example, we would like to emphasize an advantage of Definition 22: due to the very simple objective function that is applied in this definition an *exact* dynamic analysis of the relevance level can be given even for realistic problems and predicates.

Theorem 6. *Let the GA components be $C = \{0,1\}^n$, 1-point crossover with a probability P_c, generational and proportional selection without the transformation of the objective function (i.e. the fitness function equals the objective function) and no mutation. Then, for a large enough n,*

$$r(\textit{EVEN}) \approx (1 - P_c) + \frac{1}{2}P_c \tag{4.1}$$

Proof. If n is large enough, then for any $x \in C$ the probability that a randomly chosen half of x contains an even number of 1s is $1/2$. Let p_t be the proportion of EVEN in the tth generation, and let $g(p_t)$ be the expected proportion in generation $t+1$ without the effect of the genetic operators. Here, $g(p_t) = 1$. The disruption of the genetic operators under the above assumptions is

$$\mathrm{dr}(g(p_t)) = g(p_t)(1 - P_c) + \frac{1}{2}P_c$$

It is trivial that $\mathrm{dr} \circ g$ has a unique fixpoint x_0 in $(0,1]$ which is given by the equation $\mathrm{dr}(g(x_0)) = x_0$ and equals (4.1). □

Theorem 7. *Let the GA components be the same as in Theorem 6 except that the fitness function is the objective function incremented by 1. Then, for a large enough n,*

$$r(\textit{EVEN}) \approx \frac{1}{2}\left(1 - \frac{3}{2}P_c + \sqrt{(\frac{3}{2}P_c)^2 - P_c + 1}\right) \tag{4.2}$$

Proof. The same as the proof of Theorem 6, except that $g(p_t) = 2p_t/(p_t+1)$.

□

A trivial corollary immediately follows from Theorems 6 and 7.

Corollary 1. *Under the assumptions of Theorem 6 or 7, if $P_c = 1$, then EVEN is neutral, and if $P_c \neq 1$, then EVEN is an implicit forma with the relevance level given by (4.1) and (4.2), respectively.*

It should be emphasized that an implicit forma is not necessarily a useful predicate, in the sense that it is not necessarily suitable for describing the search. Its usefulness depends on the particular objective function f, e.g. on the variance of f in it. Finally, note that the definition of implicit formae characterises only the possible set of predicates that might be usful for modeling the search with the given GA implementation since it is independent of the fitness function. Over a given domain the appropriate implicit formae have to be chosen for creating the model.

4.1.3 Implicit Formae at Work

In this section, three case studies will be presented. The first illustrates how (rather exotic) implicit formae can direct the search process. The second is the subset sum problem, where we analyze the GA from the basis of implicit formae. The third offers a possible way of creating problems in which the GA performs better than a simple hillclimber algorithm, again using implicit formae. Such problems have received much attention recently (Mitchell et al., 1994).

The following GA components are the same in all three examples: $C = \{0,1\}^{100}$, 1-point crossover, mutation with $P_m = 0.003$ and a population size of 100. $P_c = 0.6$ in the last example, otherwise $P_c = 1$. The selection used was elitist and proportional. To perform the experiments, GENESIS was used modified so that it could trace our non-traditional implicit formae. The algorithms were run until 10^4 function evaluations in every experiment in each case. All functions were maximized.

An Example for Illustration: the Equal Blocks Problem

The objective function f of this example was designed especially to illustrate the idea of implicit formae. However, it should be noted that it may very well happen that real problems have features like this one. Its domain is C and for an $x \in C$ $f(x)$ is counted as follows:

Let us fix an ideal block size $b = 5$. Let us divide x into blocks that contain only 1s or 0s (e.g. $111|0|11|000$). For every block containing b' elements let us subtract a penalty $|b-b'|$ from the objective function value and let us fix the optimum value at 0. It is clear that the optimal individual will contain 20 blocks with 5 elements in each. For illustration, we give the two optimal solutions of the 30-bit equal blocks problem:

$$000001111100000111110000011111, \;\; 111110000011111000001111100000$$

This task meets our needs because formae (i.e. schemata) have little meaning and high fitness variance. It may be thought that schemata like

$$*\ldots*0111110*\ldots*$$

have high fitness. However, their fitness variance is considerable because the function is extremely epistatic and is quite insensitive to shifting due to its inherent properties.

Twenty independent experiments were performed with the GA and also with the uniform random search. The averages of the solutions found were -23.3 and -212.2, respectively. An explanation of this result can be given on the basis of the existence of implicit formae. Let us define a predicate over C.

Definition 25. Let $[y,z]$-blocknumber$\in \mathcal{P}(C)$ and $x \in C$. $[y,z]$-blocknumber(x) is true iff the number of blocks contained in x is in $[y,z]$.

Figures 4.1b and 4.1a support the following hypotheses:

- $[20,30]$-blocknumber *is an implicit forma.* As shown in Fig. 4.1a, $[20,30]$-blocknumber gained a proportion of almost 100%. It is also interesting to note that (as shown in Fig. 4.1b) the expected and observed growth fits well. This also indicates that $[20,30]$-blocknumber is an implicit forma.

- $[20,30]$-blocknumber *has an important role in modeling the search process.* The typical S-curve in Fig. 4.1a is familiar from the analysis of the above average and low order schemata of low fitness variance.

To summarize the first example, it have been shown that an implicit forma the existence of which is not trivial from the representation played an important role in the search.

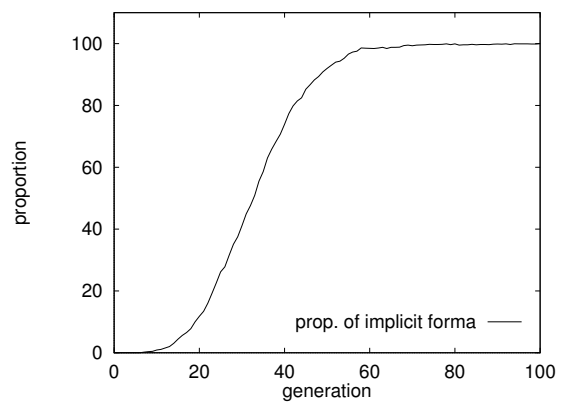


Fig. a

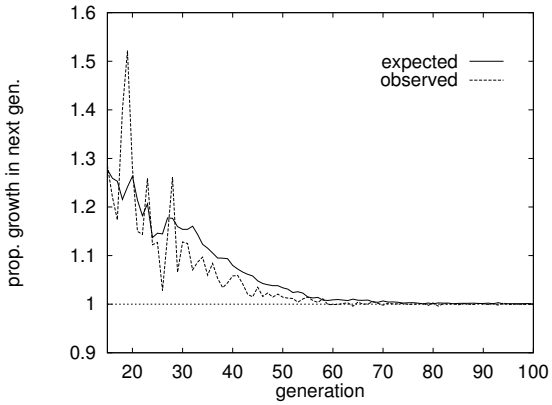


Fig. b

Figure 4.1: (**a**) the proportion growth and (**b**) the expected and observed growth of the implicit forma $[20, 30]$-blocknumber. Average of 20 independent runs.

A Real Example: the Subset Sum Problem

We study the subset sum problem here. We are given a set $W = \{w_1, w_2, \ldots, w_n\}$ of n integers and a large integer M. We would like to find an $S \subseteq W$ such that the sum of the elements in S is closest to, without exceeding, M. This problem is NP-complete.

We used the same coding and objective function as suggested in (Khuri et al., 1993): If $x \in C$ ($x = (x_1, x_2, \ldots, x_{100})$), then let $P(x) = \sum_{i=1}^{100} x_i w_i$, and then

$$-f(x) = a(M - P(x)) + (1 - a)P(x)$$

where $a = 1$ when x is feasible (i.e. $M - P(e) \geq 0$) and $a = 0$ otherwise.

When creating a problem instance, elements of W were drawn randomly from the interval $[0, 10^4]$ instead of $[0, 10^3]$ (as was done in (Khuri et al., 1993)) to obtain larger variance and thus a harder problem. The sum of all of the elements in W was 455784 and the sum to be created was 10^5. (It should be noted that optimal solutions do exist for the examined problem instance.)

After studying several experiments with the GA, a hypothesis seemed reasonable. The GA tends to sample individuals in which the number of 1s is close to $100 \cdot 10^5/455784 \approx 22$. That means that the numbers in W are treated as probability variables for which the expected value of the sum of any subset with 22 elements is 10^5. In other words, it is assumed by the hypothesis that the GA "figures out" how the problem instance was generated. After forming the above hypothesis, four algorithms were run 50 times independently:

GA As described earlier.

HYPO A direct implementation of the hypothesis. Every bit is set to 1 with a probability of $22/100 = 0.22$ independently.

RAND Uniform random search.

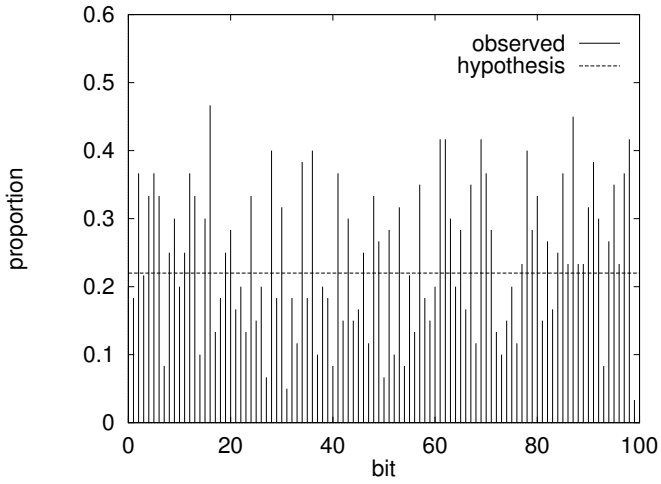


Figure 4.2: Proportion of value 1 for a given bit over the solutions of the 50 independent runs of the GA. The proportion indicated by the hypothesis is also shown.

HILLCLIMB Starting from a random solution, a randomly chosen bit is inverted and the new solution replaces the old one if it is not worse. This process is iterated.

The averages of the solutions were -4.36, -4.65, -27177 and -302.4, respectively. GA found 12, while HYPO found 6 optimal solutions during the 50 runs. The results clearly reveal that, the hypothesis is reasonable. However, the average number of bits in the 50 solutions of the GA is 28.9, which is slightly more than predicted. Figure 4.2 sheds some light on this issue. The higher peeks tend to belong to relatively small values from W, while the lower proportions indicate a relatively large value. This is because individuals containing large values tend to die off at the very beginning of the search.

It is now time to explain exactly what the hypothesis means. Clearly, it says nothing about any particular element or subset of W. The only important feature is the number of 1s in an individual, according to the hypothesis. To express this in our terminology, there are implicit formae, based on the number of 1s in a chromosome, that play a mayor role in the optimization process. This motivates the following definition.

Definition 26. Let $[y, z]$-1s$\in \mathcal{P}(C)$ and $x \in C$. $[y, z]$-1s(x) is true iff the number of 1s in x is in $[y, z]$

$[24, 34]$-1s was traced by GENESIS and the statistics are shown in Fig. 4.3. The graphs are very similar to those of the previous example, the equal blocks problem, so the conclusions are also very similar; in the case of the subset sum problem (with the GA components and the problem instance generation method used here), implicit formae play an important role.

When Will a GA Outperform Hillclimbing?

The title of this section is borrowed from (Mitchell et al., 1994). Here, using implicit formae, we will try to point out some basic differences between the GA and hillclimbing through a

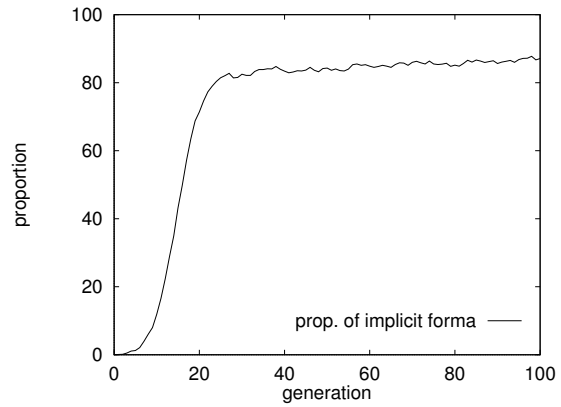


Fig. a

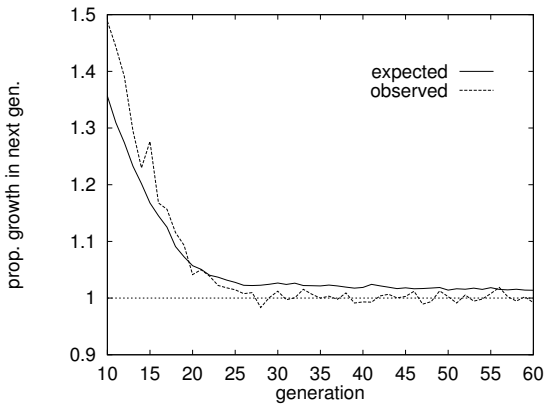


Fig. b

Figure 4.3: (**a**) the proportion growth and (**b**) the expected and observed growth of the implicit forma $[24, 34]$-1s. Average of 20 independent runs.

simple example. We believe that this approach can be generalized, however. Moreover, using the definitions given in (Radcliffe and Surry, 1995), more general representations could also be considered.

It is well known that functions that are easy for the GA (e.g. royal road functions) are easy (if not easier, see (Mitchell et al., 1994; Juels and Wattenberg, 1996)) for the bit-hillclimber, i.e. the algorithm HILLCLIMB in section 4.1.3. This is because HILLCLIMB can easily combine *explicit* formae (here schemata) in the case of such problems.

But what about *implicit* formae? As we have seen, the GA can "handle" several implicit formae besides the explicit ones, and these implicit formae are not necessarily implicit formae w.r.t. HILLCLIMB. The example of this section illustrates this effect. Let us consider the function

$$f(x) = \begin{cases} \|x\| \text{ if } \|x\| \text{ is even} \\ -\|x\| \text{ otherwise} \end{cases}$$

where $\|x\|$ is the number of 1s in x. This function is extremely hard for HILLCLIMB since every x with even $\|x\|$ is a local optimum from which HILLCLIMB cannot get out. On the other hand (as shown in section 4.1.2), EVEN is an implicit forma if $P_c < 1$ and $P_m = 0$. On the basis of this observation, P_c was set to 0.6.

Twenty independent experiments were performed with RAND, HILLCLIMB and the GA. The average best results were 67.7, 49.2 and 83.3, respectively. Observe that HILLCLIMB is considerably worse than RAND. Figure 4.4a indicates that EVEN is an implicit forma with a relevance level of approximately 0.66.

As shown in Fig. 4.4b, in spite of the constantly strong pressure, EVEN cannot go further than 66% after a quick increase at the very beginning of the search. However, the relevance level of 0.66 is enough to outperform both RAND and HILLCLIMB.

Let us make a final remark. It may be thought that EVEN is a very artificial property which will not be encountered in the case of a real problem. However, for instance, it is a well-known

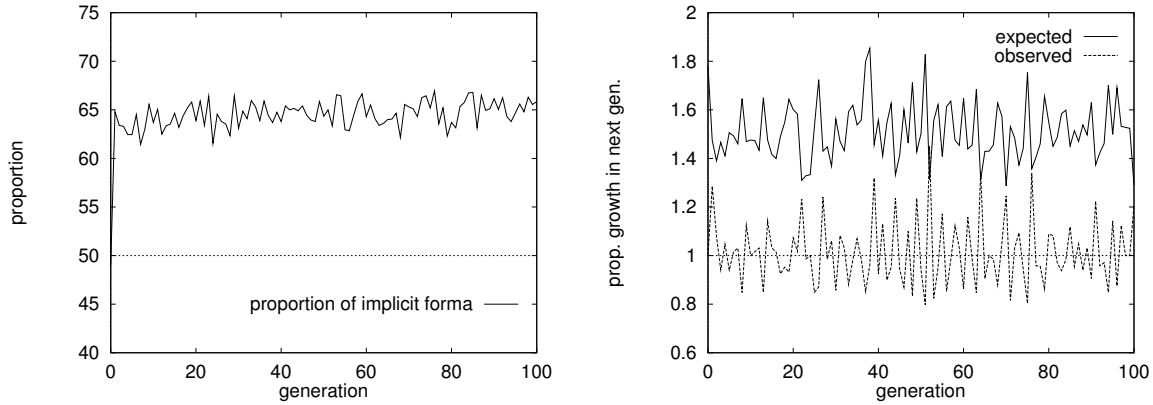


Fig. a **Fig. b**

Figure 4.4: (**a**) the proportion growth and (**b**) the expected and observed growth of the implicit forma EVEN. Average of 20 independent runs and a typical single run, respectively.

fact in chemistry that atoms that have an *even* number of nucleons in their nuclei are always more stable than those with an odd number of nucleons.

4.1.4 Happy or Sad Conclusions?

We have examined the implicit formae, the invisible forces that can direct the genetic search as strongly and definitely as explicit formae. We must be a little more precise here: *nothing* directs genetic search expect the fitness function, as the genetic operators know nothing about the expected proportions of some subsets of the search space. The operators can see only a relatively little population (selection) or a couple of solutions (recombintion, mutation). What we are talking about is *imagining* genetic search. It is very important though, since to analyse something first we need models to work with. The only problem is that in the case of a particular problem we know only the explicit formae and this can make modeling of the behaviour of the GA quite difficult if these formae have a small relevance level.

One solution could be to find implicit formae that might be suitable for modeling given domains and examine them with the tools of the GA analysis. This may be a lot of work since it is not trivial at all what the implicit formae of a given representation are, even if it is simple. In spite of this, for commonly used domains it may worth doing this analysis. However, for real applications, the representation (and thus the chromosome space C) and the operators tend to be different, difficult and problem-specific so the situation is not too hopeful. The other solution is to create representations automatically using machine learning techniques. Section 5.1 presents such an approach.

4.2 A Wave Analysis of the Subset Sum Problem

This section introduces the wave model, a novel approach on analyzing the behavior of GAs. Our aim is to give techniques that have practical relevance and provide tools for improving the performance of the GA or for discovering simple and effective heuristics on certain problem classes. The wave analysis is the process of building wave models of problem instances of a problem class and extracting common features that characterize the problem class in question. A wave model is made of paths which are composed of subsets of the search space (features) that are relevant from the viewpoint of the search. The GA is described as a basically *sequential* process; a wave motion along the paths that form the wave model. The method is demonstrated via an analysis of the NP-complete subset sum problem. Based on the analysis, problem specific GA modifications and a new heuristic will be suggested that outperform the original GA.

4.2.1 Introduction

This section introduces the wave model, a novel approach on analyzing the behavior of GAs. Our aim is to give techniques that have practical relevance and provide tools for improving the performance of the GA or for discovering simple and effective heuristics on certain problem classes.

This is very important since the models known from the literature are not capable of providing such information. There are measures of problem difficulty such as (Jones and Forrest, 1995), but they tend to be very expensive to calculate and do not provide much more information than the result of running the GA on the given problem. Other approaches suggest features that are responsible for problem difficulty such as deception (Whitely, 1991) or having long paths (Horn et al., 1994) but the identification of these features for nontrivial problems is hard and it is not clear, how to improve the performance based on the identified features. Exact models such as Markov chain analysis (Suzuki, 1993) are not tractable on nontrivial problems while the wave model is a trade-off between exhaustivity and practical usefulness. Forma analysis (Radcliffe and Surry, 1995) has similar practical motivations but while it still stands on the ground of the traditional building block hypothesis (Goldberg, 1989) the wave analysis is an attempt to shed some light on a rather different aspect of the search process.

In section 4.2.2 the basic concepts of the wave analysis will be discussed. In section 4.2.3 the practical application of the wave model is demonstrated. The problem class under consideration is the subset sum problem which is NP-complete. After analyzing this problem class, problem specific GA modifications and a new heuristic will be suggested that outperform the original GA. Finally, the results will be summarized.

4.2.2 The Wave Model

First the terminology should be clarified. The *wave analysis* is the process of creating a *wave model* of a fixed objective function or the elements of a characteristic set of functions from a problem class and then extracting the common features of the models. The GA implementation (selection and genetic operators) is also fixed. Thus, a wave model belongs to a problem

instance and a GA implementation and the wave analysis is a framework for creating and analyzing such models.

It has to be noted that the analysis is not an automated process. It is a framework that helps creating problem class specific models, but finding a good wave model remains a hard task. The evaluation of the results of the analysis (e.g. the description of the role of the genetic operators) is non-trivial as well. The utility of the approach is not providing trivial methods for gaining information about a problem. Instead, it is a "way of thinking" that makes it possible to learn from the GA how to solve problems, and to develop new, effective and problem class specific heuristics.

As it is widely known, the GA is a very flexible meta-heuristic that is successful on very different problem classes. Models of the GA try to capture the reasons of this flexibility. For example, the oldest, schema based approach suggested, that the search process is nothing else but the identification of 'building blocks' via selection and combining them together via the reproduction operators in an implicitly parallel way.

Using the wave analysis we look at the GA as a collection of heuristics and in the case of a given problem class we try to identify the one actually used by the GA. This approach simply means that the actual search process performed by the same GA implementation in different domains can have different models and these models can be converted to heuristics that may outperform the original GA (see Section 4.1). The wave model is *sequential* emphasizing the similarity between the GA and hillclimbing methods. In this framework the GA is in fact a very general and flexible hillclimbing method.

Now, let us fix the notations. Let S be the search space, $f : S \to \mathbb{R}$ the objective function, C the coding space and $g : S \to C$ the injective coding function. For the sake of simplicity, the notation $f(c)$ $(c \in C)$ will be used instead of $f(g^{-1}(c))$. Let P_0 be the initial population and P_i the population at step i. Let $\overline{f}(P_i)$ be the average function value of the individuals in population P_i. The objective function will be maximized.

Waves

Before introducing the concept of waves, an assumption will be made: $\overline{f}(P_i) \leq \overline{f}(P_j)$ if $i < j$ and the variance of f in the succeeding populations does not increase. This assumption is rather weak since it follows from the properties of the selection mechanisms commonly used in GAs (see e.g.(Blickle and Thiele, 1995)).

A wave needs a space in which it can spread. To construct this space, let us sort the elements of C along a one-dimensional line according to the partial ordering given by f. Then, every element in this ordering will be a subset of C with elements having the same function value. Observe that the above assumption means that during the search the population can be looked at as a *wave* that spreads towards the region with the better values. Such a wave is shown in Fig. 4.5.

The goal of creating a wave model is to extract the problem specific characteristics of this wave motion. The main method of achieving this goal will be a discretization in terms of characteristic features of C and the result of this will be called a *path*. Paths will be defined in the next section.

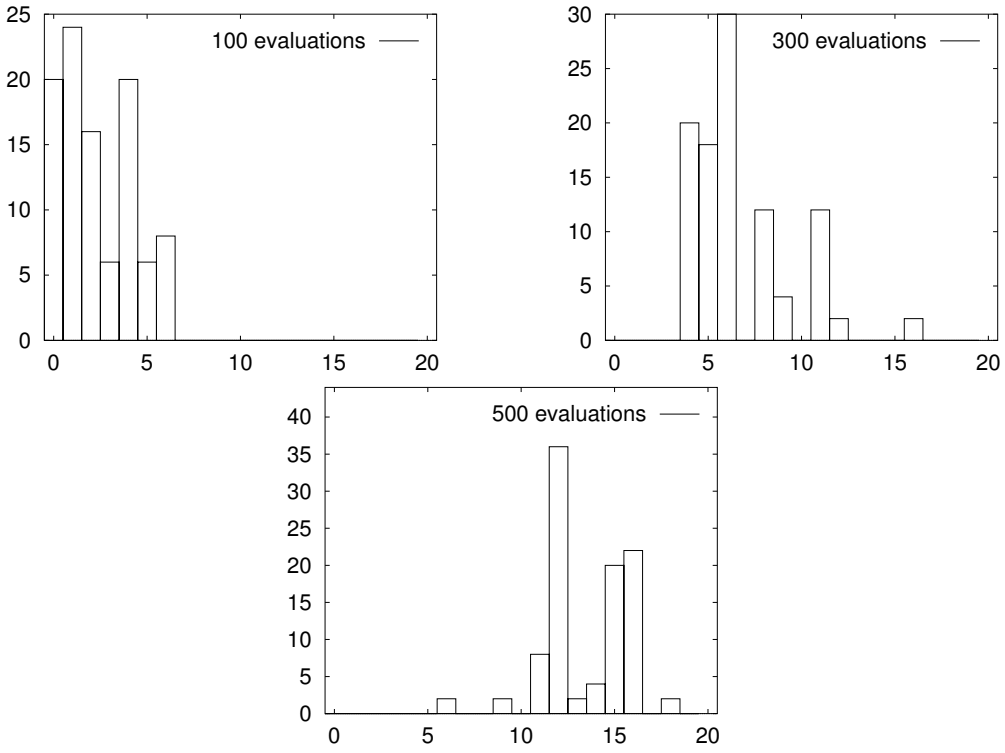


Figure 4.5: Here $S = [0, 1]$, $f(x) = x$. The population size is 50. Ranking selection and binary encoding were used. The evaluation number is 100, 300 and 500 respectively. The height of the box at point i indicates the proportion of $\text{Prefix}^1_{[i,i]}$ in the population (see Definition 30).

Paths

First, let us define a partial ordering over the subsets of C as it was done in (Vose, 1991).

Definition 27. Let $C_1, C_2 \subset C$. $C_1 < C_2$ iff $\max_{c \in C_1} f(c) < \min_{c \in C_2} f(c)$.

The next definition will be the basis of the definition of path.

Definition 28. Let $C_i \subset C$, $(i = 1, \ldots, k)$. The sequence $C_1, \ldots, C_k$ is an *increasing sequence of features* iff $C_i < C_j$ for every $i < j$.

Every path will be an increasing sequence of features but several restrictions have to be considered. The first and most natural property an increasing sequence must have to be a path is the wave motion property.

Definition 29. An increasing sequence of features $C_1, \ldots, C_k$ has the *wave motion property* iff for every i Pr($C_i \supseteq P_j$ for some j) ≈ 1. (where Pr() stands for probability and P_j is the population at step j).

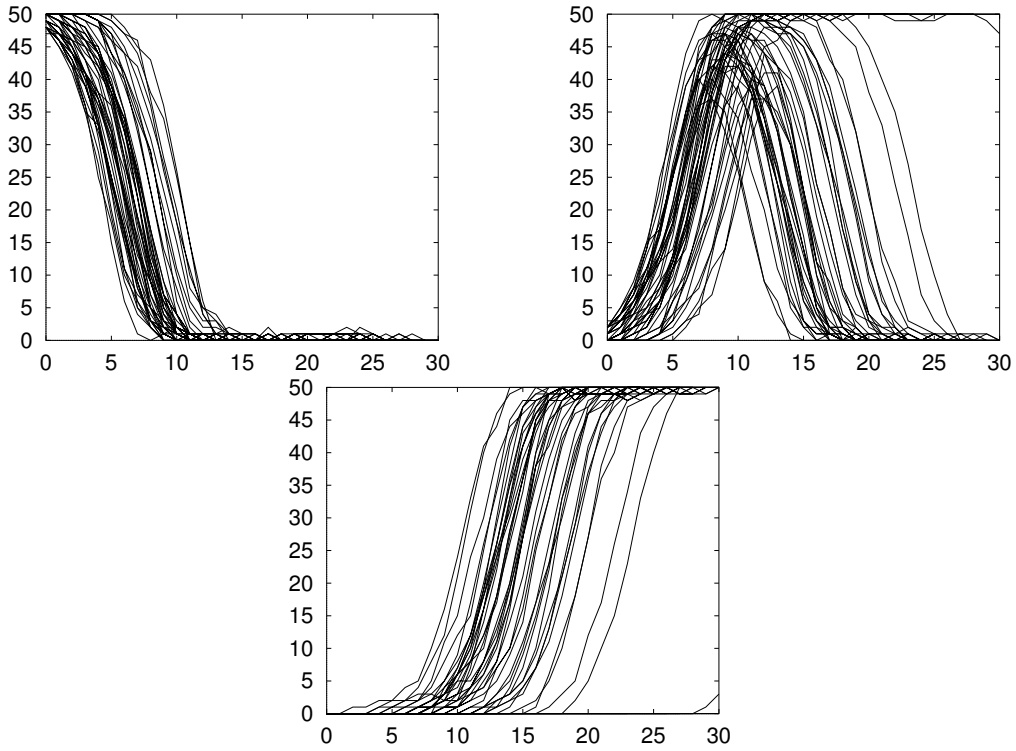


Figure 4.6: Here $S = [0, 1]$, $f(x) = x$. The population size was 50. Ranking selection and 20-bit binary encoding were used. The evaluation number was 1000. 50 independent runs were performed. The number of representatives of the features $\text{Prefix}^1_{[0,5]}$, $\text{Prefix}^1_{[6,12]}$ and $\text{Prefix}^1_{[13,20]}$ are shown as the function of generation index for every run.

Observe that the succeeding elements of the sequence with the wave motion property has to cover the population one after another because it has been assumed that the average fitness increases during the search and the sequence in question is increasing in the sense of Definition 28. The definition allows us to verify the wave motion property both empirically and mathematically. Figure 4.6 exemplifies the wave motion property. The definition of the elements of the increasing sequence illustrated in Fig. 4.6 is the following:

Definition 30. Let $c \in \{a, b\}^n$. c has the feature $\text{Prefix}^a_{[i,j]}$ if the first k letter of c is a $(i \leq k \leq j)$ and if $k < n$ then the $(k + 1)^{th}$ letter of c is b.

Example 5. Let $C = \{0, 1\}^4$. Then, using the traditional schema notation, $\text{Prefix}^1_{[1,2]} = 10{*}{*} \cup 110{*}$, $\text{Prefix}^0_{[2,4]} = 001{*} \cup \{0001, 0000\}$.

Let us shed some light on how to read the figures similar to Fig. 4.6. Every graph in the figures corresponds to a feature. A graph depicts the number of elements in the given generation (x-axis) having the feature in question. Instead of averaging the results, the graphs contain a continous line for every experiment performed. For example, Fig. 4.6 clearly shows that in generation 10 $\text{Prefix}^1_{[0,5]}$ is almost not represented in most of the experiments, $\text{Prefix}^1_{[6,12]}$

dominates the generation i.e. the wave is here in generation 10 and $\text{Prefix}^1_{[13,20]}$ starts gaining strength.

At this point a natural question arises: can we accept an increasing sequence as a model of the GA if the sequence in question shows the wave motion property. The answer is certainly no. The problem is that if $P_i \subset A$ holds for some feature A and for a population P_i then $P_i \subset B$ will also be true for any $B \supset A$. To overcome this difficulty, it has to be required that every element of the increasing sequence of features has to be *minimal* in the sense of the next definition:

Definition 31. An element of an increasing sequence with the wave motion property C_i is *minimal* if the replacing of C_i with any of its subsets results in a new sequence that does not have the wave motion property anymore.

Now the definition of a *path* can be given.

Definition 32. An increasing sequence of features is a *path* if it has the wave motion property and every element is minimal in it.

Path Decomposition

Definition 32 is still not sufficient for our purposes; some refinements have to be made. It may very well happen that a path says little about the process inside the GA and cannot be a basis of improving the performance of the search. The problem is connected with the multimodality of the objective function. To shed some light on this issue, let us consider the example shown in Fig. 4.7. Though it is a path, it is clear that if for the starting population P_0 $\subset \text{Prefix}^1_{[1,5]}$ holds then with the given settings no solutions will be generated that would start with a 0 and in fact the search will be identical with the earlier example shown in Fig. 4.6 so the sequence $\text{Prefix}^1_{[1,5]}$, $\text{Prefix}^1_{[6,12]}$, $\text{Prefix}^1_{[13,20]}$ is a path. Similarly, its 0-prefixed counterpart is a path as well. The above comments make it clear that the path in question has some kind of structure and the information about this structure is essential from the viewpoint of a good model. The above phenomenon motivates the next definition.

Definition 33. A path $C_1, \ldots, C_k$ is *complex* iff there are two paths $B_1, \ldots, B_k$ and $A_1, \ldots, A_k$ such that $B_i \cap A_i = \emptyset$ and $B_i \cup A_i = C_i$ $(i = 1, \ldots, k)$. If a path is not complex then it is *simple*.

Now the definition of the *wave model* can be given.

Definition 34. A *wave model* of the search performed by an implementation of the GA on a given objective function is a set of simple paths such that for every method used for generating the initial population P_0 there is exactly one path in which the first feature C_1 covers P_0 with a probability approaching 1 ($\Pr(C_1 \supset P_0) \approx 1$).

This definition of the wave model is very simple and could be refined in several ways. For example it says nothing about the relation of different paths or other possible types of decompositions of paths. However, for the present discussion it suffices since the focus is on the empirical results of section 4.2.3.

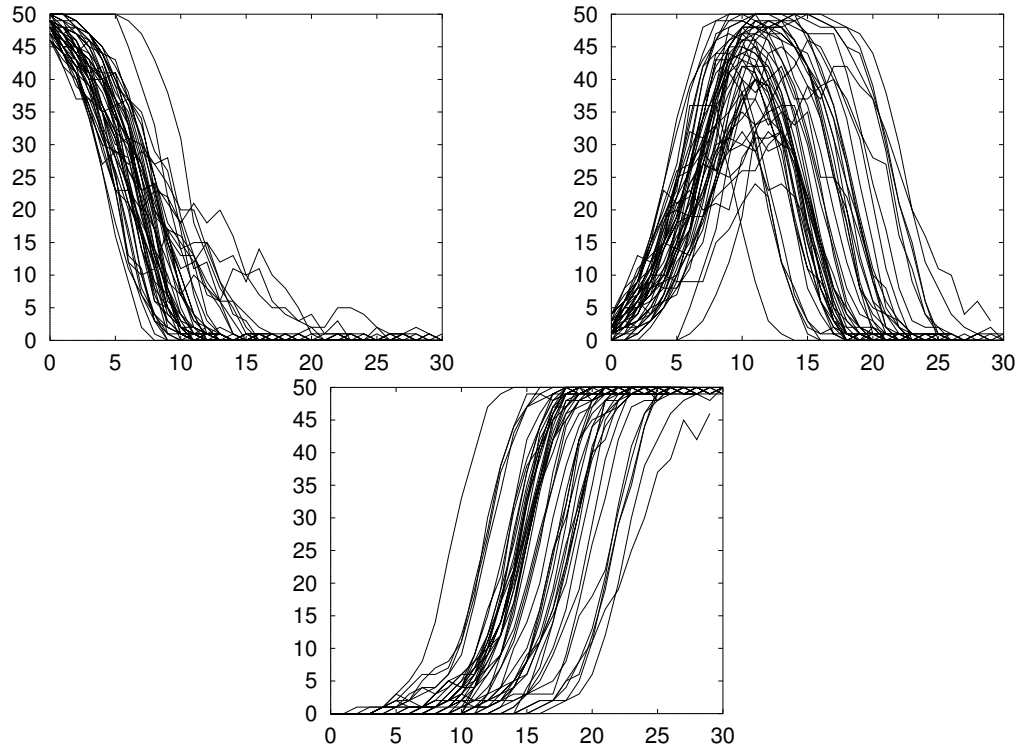


Figure 4.7: Here $S = [0, 1]$, $f(x) = |x - 0.5|$. The population size was 50. Ranking selection and 20-bit binary encoding were used. Only 1-point crossover was used with a probability of 1. The evaluation number was 1000. 50 independent runs were performed. The number of representatives of the features $\text{Prefix}^1_{[1,5]} \cup \text{Prefix}^0_{[1,5]}$, $\text{Prefix}^1_{[6,12]} \cup \text{Prefix}^0_{[6,12]}$ and $\text{Prefix}^1_{[13,20]} \cup \text{Prefix}^0_{[13,20]}$ are shown as the function of generation index for every run.

4.2.3 The Subset Sum Problem

In this section we demonstrate the wave analysis using the subset sum problem which is NP-complete. Then, using the wave model, the performance of the GA will be improved and a heuristic will also be given that outperforms the original GA.

Problem Description and Representation

In the case of the subset sum problem we are given a set $W = \{w_1, w_2, \ldots, w_n\}$ of n integers and a large integer M. We would like to find a $V \subseteq W$ such that the sum of the elements in V is closest to, without exceeding, M. This problem is NP-complete. Let us denote the sum of the elements in W by SW.

We created our problem instances in a similar way to the method used in (Khuri et al., 1993). The size of W was set to 100 and the elements of W were drawn randomly with a uniform distribution from the interval $[0, 10^4]$ instead of $[0, 10^3]$ (as was done in (Khuri et al., 1993)) to obtain larger variance. According to the preliminary experiments, the larger variance of W results in

harder problem instances. Five problem instances were generated (SUB1, SUB2, SUB3, SUB4 and SUB5). Since the value of M seemed to be interesting during the preliminary experiments, M-s was set in a different way for all the five instances. We set M_i (M corresponding to the i^{th} problem instance SUBi) to the closest integer to $SW_i \cdot i/9$ where SW_i is the SW corresponding to SUBi.[1] (It should be noted that exact solutions do exist for the examined problem instances.)

We used the same coding and objective function as suggested in (Khuri et al., 1993). C was $\{0,1\}^{100}$. If $x \in C$ ($x = (x_1, x_2, \ldots, x_{100})$), then let $P(x) = \sum_{i=1}^{100} x_i w_i$, and then

$$-f(x) = a(M - P(x)) + (1 - a)P(x)$$

where $a = 1$ when x is feasible (i.e. $M - P(e) \geq 0$) and $a = 0$ otherwise.

Wave Analysis

The experiments were performed with GENESIS (Grefenstette, 1984). The selection type was ranking selection. The operators were 1-point crossover and traditional mutation. The probabilities of the operators are 1 and 0.003 if not otherwise stated. The population size was 100 and the number of evaluations was 5000 in every experiment. The initial populations were generated by a uniform random sampling of C.

Before giving the analysis an important issue has to be discussed: the methods for identifying the features that would form the paths of the wave model. In general, it is a tough problem and requires a lot of work. In fact, it needs a scientific research: making a hypothesis, verifying it doing experiments with the GA, improving the hypothesis and so on. The difficulty is hidden in the fact that the set of possible features for given configurations of the GA is very large and mostly *undiscovered*. Schemta form only a (maybe small) subset of this collection of features. For example, the features that will arise in this work are fairly independent of schemata and other examples are given in Section 4.1. It is very likely that any automation of this feature-finding process (if possible) will involve very powerful and intelligent computational methods. Section 5.1 offers a framework of finding features using machine learning methods.

Now let us see the wave analysis of the subset sum problem. Experimenting with the GA, it has been found that on every problem instances the search has two phases: the distribution optimization phase and the hillclimbing phase.

Distribution optimization

This phase is connected to the size of M. The method for generating the initial population has a special bias regarding the number of bits. This factor has a gaussian distribution with a mean value of 50 (the half of the string-length). This means that most of the elements in P_0 have approximately 50 1s so the expected value of the fitness function is $SW/2$. If M is smaller than $SW/2$ then the initial population can be expected to have a poor performance. Distribution optimization means that the GA alters the distribution and the number of 1s to a better configuration. This phenomenon is the most characteristic in the case of SUB1 so we will

[1] Instances with an $M > SW/2$ have an 'almost' equivalent problem with an $M' = SW - M$. 'Almost' equivalent, because of the asymmetric construction of the objective function.

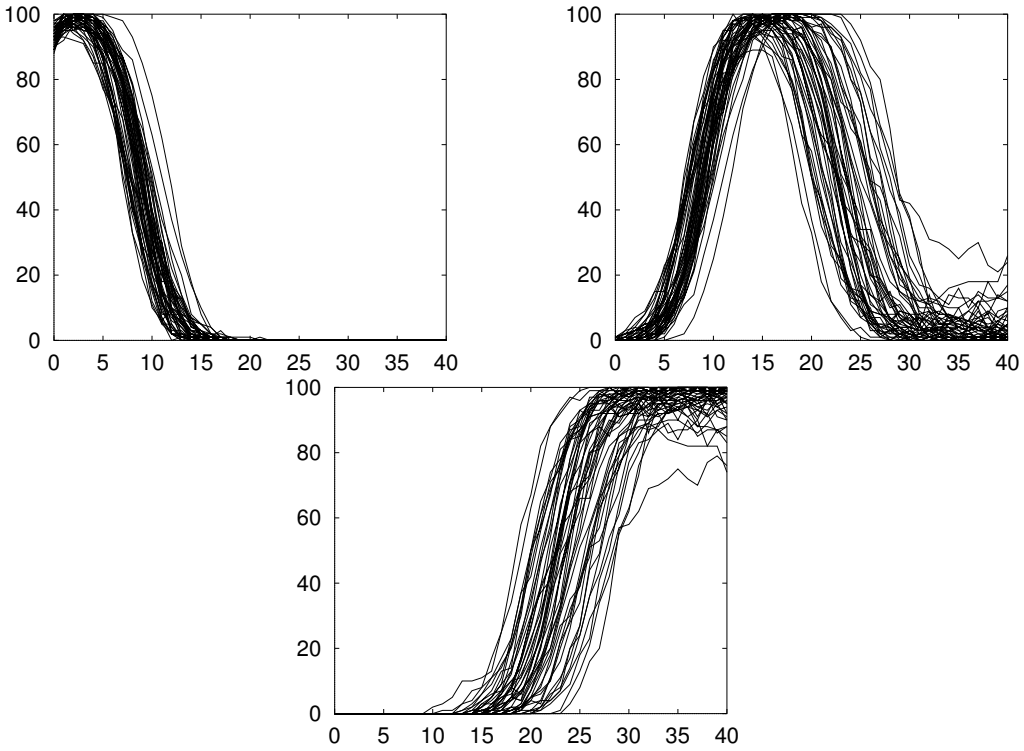


Figure 4.8: The number of representatives of $L_{[15,30]}$, $L_{[5,14]}$ and $L_{[0,4]}$ respectively. The results of 50 runs are shown as a function of generation index.

concentrate on this problem instance here. To construct a path for SUB1 let us first define a feature:

Definition 35. A $c \in C$ has the feature $L_{[i,j]}$ if the subset defined by c contains k of the largest 50 elements of W and $i \leq k \leq j$.

Then, we claim that the sequence $L_{[15,30]}$, $L_{[5,14]}$, $L_{[0,4]}$ is a path. The mathematical considerations implying that the above sequence is increasing with a high probability (w.r.t. the samples taken by the succeeding populations) are straightforward and elemental and therefore omitted. The empirical results shown in Fig. 4.8 imply the wave property. The minimality and simplicity of the path are also trivial if considering the definitions of these properties (see section 4.2.2).

Another argument beside this model is that it predicts[2] that the high mutation probability which has a bias towards the initial distribution of bits in the solutions detaining the wave motion of the above path will decrease the performance. Experimental results justify the prediction (see Fig. 4.9 and NAIV-M in Table 4.2).

[2]Prediction is possible because the path under consideration covers the whole search space and therefore does not allow any other paths to exist.

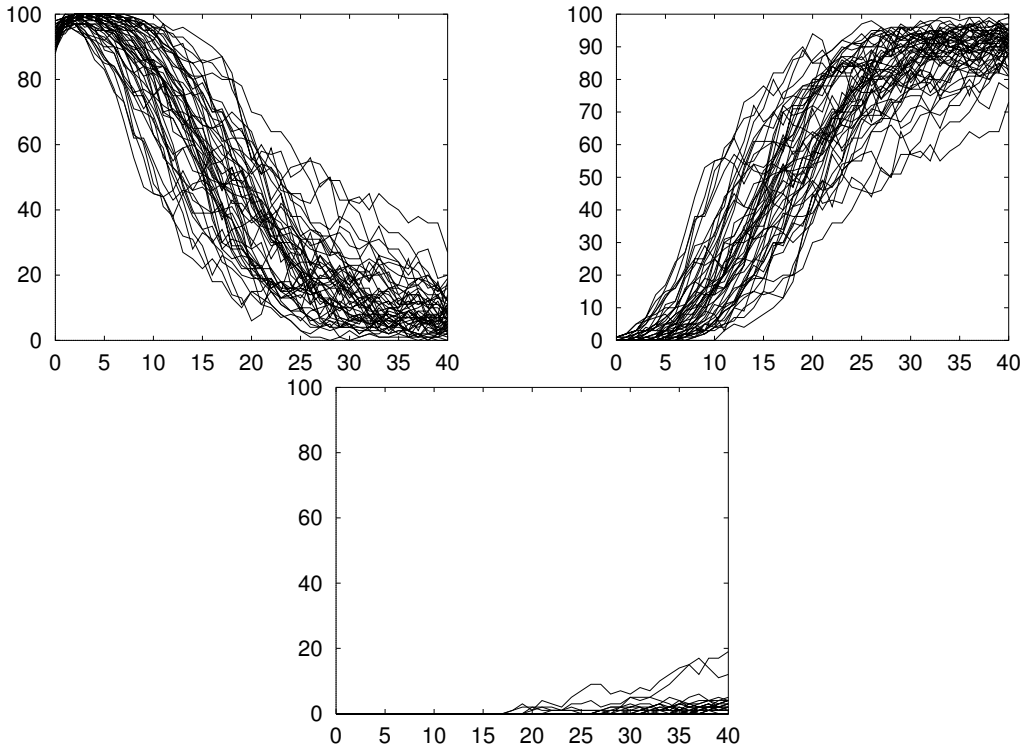


Figure 4.9: The proportions of $L_{[15,30]}$, $L_{[5,14]}$ and $L_{[0,4]}$ respectively. The probability of mutation is increased to 0.06. The results of 50 runs are shown as a function of generation index.

The Hillclimbing Phase

This phase begins when the optimization of the bit distribution in the solutions has been performed. The model of this second phase does not share the linear style of the first phase. On the contrary, we suggest that there is an enormous number of paths in the model of this phase that are built of relatively small sets and are highly problem instance specific. This is why this phase is called the hillclimbing phase; such path structure calls for a hillclimbing strategy. This claim is supported by section 4.2.3.

There are several arguments that support our suggestion regarding the path structure of this phase. First, every run on every problem resulted in a different solution that are considerably far from each other (see Table 4.1). The many optimal solutions found do not seem to have any common feature except the bit distribution. Results of coding theory (van Lint, 1992) also support that a great number of paths can exist without interfering with each other. As it was shown in Chapter 2, GAS, a GA with a special niching technique supporting the separate handling of different local optima outperformed the standard GA on this problem. Finally, the modifications of the GA that were made using this hypothesis were successful as it will be seen in section 4.2.3.

	Hamming distance			
	min.	max.	average	variance
SUB1	13	69	47.56	91.89
SUB2	24	63	47.55	50.5
SUB3	34	65	49.91	26.2
SUB4	34	68	49.71	26.77
SUB5	35	64	49.97	25.21

Table 4.1: The values correspond to sets of optimal solutions of problem instances found during all the experiments icluding the ones with the modifications of the GA (section 4.2.3). The minimum, the maximum, the average and the variance of the Hamming distences of pairs from these sets are shown.

The Wave Model

As the mindful reader has observed already, no exact wave models have been given for any of the problem instances under consideration. Since the aim of the wave analysis is to extract characteristic features of whole problem classes, the problem instance specific details (such as the exact path structure of the second phase for a given problem instance) are not important. What was given is a general characterization of the search on an arbitrary problem instance of a class of the subset sum problem.

Application of the Results

As it has been suggested, the search has two phases: the bit distribution optimization phase and the hillclimbing phase. It will be shown that both require extra computational effort that can be saved. In the following, the modified algorithms will be described.

OPTDISTR**, distribution optimization.** This phase can be totally eliminated by explicitly ensuring that the bit distribution is optimal from the very beginning of the search w.r.t the bias of the population initialization procedure. This was done by modifying the problem instances.[3] For a problem instance SUBi from the base set W_i the k_i largest elements have been deleted where k_i was such that the sum of the remaining elements of W_i was the closest to $2M_i$. A solution of a modified problem instance naturally defines a solution of the original problem instance with the same function value.

All the following algorithms in this section use these modified problem instances.

5X1000**, hillclimbing phase.** According to our model, there are a lot of paths in this phase. Since they are rather far from each other (see 4.1) and do not seem to show any common

[3]The algorithm of the modification is independent from the problem instances so can be looked at as a problem class specific modification of the GA.

	NAIV		NAIV-M		OPTDISTR	
	#opt	average	#opt	average	#opt	average
SUB1	4	-8.0	0	-6136.0	14	-5.26
SUB2	5	-7.64	0	-186.8	11	-3.92
SUB3	3	-10.5	2	-20.32	9	-3.98
SUB4	5	-7.94	6	-8.62	7	-5.5
SUB5	5	-9.6	8	-6.12	9	-6.7

	OPTDISTR-M		5X1000		HEUR	
	#opt	average	#opt	average	#opt	average
SUB1	12	-2.72	5	-5.8	15	-3.48
SUB2	9	-4.34	9	-4.0	9	-7.96
SUB3	9	-4.8	9	-3.35	4	-8.6
SUB4	10	-5.94	11	-4.0	4	-13.8
SUB5	5	-7.72	14	-3.76	2	-13.94

Table 4.2: The methods used are described in the text. NAIV is the GA used in section 4.2.3. '-M' means that only mutation was used with a probability of 0.06. The values correspond to the result of 50 independent runs. The number of optimal solutions found and the average of the results of the runs are shown.

structure it was assumed that it would be a good idea process them separately. Therefore the population size was reduced to 2 and the GA was run 5 times with 1000 evaluations in each to ensure that only one path is processed at a time. Then, the best solution was picked as a result. The only operator was mutation with a probability of 0.06. Note that this algorithm is rather similar to – though more flexible than – the stochastic hillclimber.

HEUR**, a heuristic.** To examine the effect of the optimal bit distribution a heuristic has been introduced which simply generated 5000 random individuals on the modified problem instances. This method is in fact equivalent to generating an initial population with 5000 elements.

Evaluation

It can be seen that the optimal bit distribution is essential; even the random search (HEUR) performed well though only the bit distribution was optimized.

The application of the information about the hillclimbing phase was useful as well. 5X1000 had the best average performance on almost every problem instance especially on SUB5 which is the hardest (the largest) problem instance since the smallest set is subtracted from W_5 due to the bit distribution optimization. Note that no fine tuning of the parameters have been performed to adapt the method to smaller problems. Table 4.2 clearly shows that the model has practical relevance.

4.2.4 Summary

In this section the wave analysis of GAs has been described. The wave analysis is the process of building wave models of problem instances of a problem class and extracting common features that characterize the problem class in question. A wave model is made of paths which are composed of subsets of the search space (features) that are relevant from the viewpoint of the search. The GA is described as a basicly *sequential* process; a wave motion along the paths that form the wave model.

The above mentioned features include but are not at all limited to schemata. In fact there are many that are independent of schemata such as those involved in the wave analysis of the subset sum problem presented here. Using this analysis, modifications of the naiv GA has been suggested that outperformed the original algorithm on the subset sum problem class.

4.3 Trajectory Structure of Fitness Landscapes

Characterization of trajectory structure of fitness landscapes is a major problem of evolutionary computation theory. In this section a hardness measure of fitness landscapes will be introduced which is based on statistical properties of trajectories. These properties are approximated with the help of a heuristic based on the transition probabilities between the elements of the search space. This makes it possible to compute the measure for some well-known functions: a ridge function, a long path function, a fully deceptive function and a combinatorial problem: the subset sum problem. Using the same transition probabilities the expected number of evaluations needed to reach the global optimum from any point in the space are approximated and examined for the above problems.

4.3.1 Introduction

This section is concerned with the characterization of fitness landscapes w.r.t. optimizers that use a stochastic hill-climbing heuristic. Members of the field of evolutionary computation belong to this class of optimizers. The definition of *hardness*, maybe the most important feature of a landscape is far from clear (see (Naudts and Kallel, 1998) for a summary of the available measures) so we need to clarify the problem we would like to tackle.

Theory or Practice?

The first question is whether one would like to give a *method* for characterizing fitness functions in practice i.e. a method which can be used to make predictions on interesting existing (maybe black box) problems or one would like to gain *theoretical insights* of the working of the optimizer.

In the first case the method needs to run fast; preferably faster than the optimizer itself otherwise it would be easier to run the optimizer and see what happens. There are attempts to give such practically useful measures like those based on the observed trajectories of several runs with some fixed set of parameters (Kallel, 1998), or epistasis variance (Davidor, 1991a).

These approaches are useful but from a theoretical point of view they have their limitations. The main problem is that these methods do not take the whole search space into account but instead only a very small fraction of it. The well-known measure, fitness distance correlation (FDC) (Jones and Forrest, 1995) does not really belong to this first class since it needs to know a global optimum in order to make predictions. If based only on a relatively small sample these measures can be highly misleading as described in (Naudts and Kallel, 1998) and as confirmed by our own experience.

Theoretically motivated measures need not be computed efficiently but they should be capable of providing a characterization of easy and hard functions and maybe useful in suggesting constructed easy and hard problems. While the characterization suggested here belongs to this class, techniques will be suggested that make it possible to examine relatively small problems empirically.

Multimodality

Another question is multimodality. Many problem characterisation methods make the assumption that the main criterion of the success of an algorithm is finding the global optimum. This practice has serious drawbacks however. The first is that there can be multiple global optima that may be arbitrarily far from each other which makes the original FDC meaningless. This problem is not serious since it is possible to choose the distance from the closest global optimum. The more serious problem is that in engineering practice where evolutionary computation has its main applications it is not always necessary to find a global optimum. Intuitively, it is enough to find solutions that are judged good enough by the engineers. This means that local optima are not simply obstacles in the way of success; their distribution and the structure of their attraction areas are essential from the point of view of problem characterization.

For example if a function has a unique global maximum that is a "needle in the haystack" but in the same time it has another local optimum which is almost as good as the global one but it has a large area of attraction then this function should be characterized as fairly easy. However, when based on the whole search space which should be the optimal setting since it contains the most information, FDC would predict that it is a hard problem if the global and local optima are far from each other. Our approach emphasizes the role of the local optima.

One Number?

The practice of searching for a single number as a measure of problem hardness is is similar to the efforts of psychologists to characterize human intelligence with a single IQ. Both have the drawback of oversimplification. Our approach emphasizes the role of the interpretation of certain figures and our measure of difficulty is in fact a function of the stopping criterion of the search but is it possible to take other factors into account as well.

The Idea

The basic idea is to examine the trajectories of the space w.r.t. a given operator and stopping criterion. The ending points of these trajectories form a very interesting set: these are the points

the search is expected to converge to. Statistical measures over this set w.r.t. some properties such as fitness or probability of being the result of the search are the best candidates for being a hardness measure. Here a measure of this kind will be suggested.

There are practical problems when such measures have to be computed since a huge amount of calculation is needed. As it was mentioned, it would be possible to obtain the trajectories of the search by running the algorithm and collecting the history of the process. Our approach is different. We define transition probabilities between the elements of the space. These values define the probability of every possible trajectory and also they make it possible to identify the points the algorithm is accepted to converge to. A heuristic for approximating the number of trajectories leading to a given point will be given and statistical measures based on this data will provide the hardness measure.

Based on these transition probabilities it is also possible to approximate the expected number of evaluations needed to get from a point to a given other point. This values can be used as distance measures and plots can be drawn that depict the convergence relations.

4.3.2 Basic Notions

This section introduces a model of stochastic hill-climbing search. This model will be used in two ways. The first application is an iteration formula used for approximating the expected number of steps of reaching the global optimum or any set of solutions from a point of the search space. The second and maybe more original application is to characterize fitness functions using the notion of *endpoints*. Let $S = \{s_1, s_2, \ldots, s_M\}$ be the search space. Let us fix S to be the binary space $\{0,1\}^l$.

A novel distance notion

Let us define an ordering of the solutions as was done in (Vose, 1991).

Definition 36. We say that $s_i \leq s_j$ iff $f(s_i) \leq f(s_j)$, where f is an arbitrary fitness function of type $S \to \mathbb{R}$. This means that s_M is a global optimum.

For every mutation operator the probability of getting from a given solution to another one can be given. For example the operator of the stochastic hill-climber is that every bit in the solution is flipped with a given t probability. In this case the probability of getting from a given solution to another one is $t^d(1-t)^{l-d}$, where d is the number of different bits and l is the length of the solutions. This model is used in this section. However, any mutation or other genetic operator can be chosen.

Let $Pr_{ik}^{(j)}$ denote the probability of getting from s_j to s_k via the application of the mutation operator j times. This notation is important for the main iteration formula. First of all we have to compute the values $Pr_{ik}^{(1)}$ for every index. Let s_i and s_k be solutions, let the probability of flipping a bit be t, let the Hamming distance between s_i and s_k be d and let the length of a bit

string be l. Then

$$Pr_{ik}^{(1)} = \begin{cases} t^d(1-t)^{l-d} & \text{iff } s_i < s_k \\ \sum_{s_i \geq s_k} t^d(1-t)^{l-d} & \text{iff } i = k \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Note that the case $i = k$ is very important, because $Pr_{ii}^{(1)}$ denotes the probability of no motion, i.e. the sum of the probabilities of constructing a worse solution by the mutation operator. The only exception is the case $i = M$, because at this point we have reached the global optimum and that is the end of the search, so we choose $Pr_{MM}^{(1)} = 0$. This guarantees that the following iteration formula will measure the distance from the global optimum appropriately. $Pr_{iM}^{(j)}$ is computed by the following iteration formula:

$$Pr_{iM}^{(j)} = Pr_{ii}^{(1)} Pr_{iM}^{(j-1)} + \sum_{s_k > s_i} Pr_{ik}^{(1)} Pr_{kM}^{(j-1)}, \tag{4.4}$$

where $j = 2, 3, \ldots$. This formula means that $Pr_{iM}^{(j)}$ is the probability of getting from s_i to the global optimum in j steps. If $i = M$ i.e. the solution is the global optimum then $Pr_{MM}^{(j)}$ need not be computed because it is 0. The expected value of the number of steps from a solution to the global maximum is given by the limes of the series

$$E_{iM}^{(j)} = E_{iM}^{(j-1)} + jPr_{iM}^{(j)} \qquad (j = 2, 3, \ldots). \tag{4.5}$$

This formula gives a new distance notion for the stochastic hill-climbing search which is in fact the expected number of function evaluations. This distance is more expensive to calculate than the Hamming distance, which is the basis of the FDC, but it provides more accurate and more informative results. In the numerical experiments the following formula was used for checking convergence:

$$jPr_{iM}^{(j)} < (j-1)Pr_{iM}^{(j-1)} \quad \text{and} \quad jPr_{iM}^{(j)} < \varepsilon. \tag{4.6}$$

Note that if the expected number of steps is zero then the solution at hand is a global optimum or the global optimum cannot be reached at all. The later is not possible with the operator we are using.

Endpoints

Our definition of endpoints is motivated by the very simple though quite common stopping criterion: if the best solution does not improve after a given number of evaluations then the program will stop. A solution will be called an endpoint if it is not expected to improve in a given number of steps.

Definition 37. s_i is endpoint if $P_{ii}^{(1)}$ is greater than a given bound (near 1).

Note that a given bound K corresponds to a stopping criterion of $1/(1-K)$ steps without improvement. Also note that the set of endpoints depends on this parameter; the parameter of the stopping criterion. The notion of endpoints depends only on the transition probabilities just like the expected number of evaluations.

Though the notion of endpoints is independent from the calculation of the expected number of evaluations, it is possible to generalize our above formulas to handle the set of endpoints instead of the global optima alone. Let Z denote the set of the endpoints. Then

$$Pr_{iZ}^{(j)} = \sum_{z \in Z} \left(Pr_{ii}^{(1)} Pr_{iz}^{(j-1)} + \sum_{s_k > s_i} Pr_{ik}^{(1)} Pr_{kz}^{(j-1)} \right), \tag{4.7}$$

where $j = 2, 3, \ldots$. Note that in this case $Pr_{zi}^{(1)}$ $(z \in Z)$ is set to 0 for all $s_i \in S$, similarly to the case of the global optimum in Eq. (4.3). Eq. (4.5) is changed to

$$E_{iZ}^{(j)} = E_{iZ}^{(j-1)} + jPr_{iZ}^{(j)} \qquad (j = 2, 3, \ldots). \tag{4.8}$$

Note that Eq. (4.7) gives the probability of reaching one of the endpoints from s_i in j steps and Eq. (4.8) gives the expected value of the number of steps from a solution to an endpoint.

Endpoints are essential since they are the possible results of the search. This motivates our approach that statistical measures of certain properties of these endpoints are the best candidates for being a good hardness measure. Clearly Eq. (4.8) says nothing about a given endpoint, only about the set of endpoints while the most important question is: what is the probability of reaching a given endpoint starting from a random element of the space? This probability can be approximated using the transition probabilities which define a weighted graph over the search space S. Let us determine a spanning forest in this graph in the following way: let the roots of the trees be the endpoints and for every other point let us select the outgoing edge (transition) with the highest probability. It is easy to see that this method provides us with a spanning tree with a maximal weight in $O(S)$ time. Note that the edges of these trees point *towards* their roots.

The endpoints of a given function describe the expected results of the stochastic hill-climber. The probability of reaching a given endpoint can now be approximated with the proportion of the points of the tree rooted from the given point in S. For instance if $|S| = 100$ and the tree of a given endpoint contains 10 points then we say that the probability of reaching that endpoint is 0.1. Using this probabilities it is possible to approximate the average fitness of the results of multiple optimization runs. This values can be compared to the actual observed values as will be done in Section 4.3.3.

It is also possible to characterize the deceptiveness of the problem with these probabilities. Let us introduce our *deceptiveness coefficient*, a number from $[0, 1]$, which is based on the notion of endpoints.

Definition 38. Let B_K be a set of bounds (minimal transition probabilities as parameters of being an endpoint) from $[a, 1)$ where $0 < a$ for a given bound $K \in B_K$. Let F_{min} be the minimum, F_{max} be the maximum of the fitness of endpoints, and let E be the expected value of fitness. Let

$$s_K = 1 - \frac{E - F_{min}}{F_{max} - F_{min}}.$$

If $F_{max} - F_{min} = 0$ then let s_K be 0. Then the deceptiveness coefficient is the mean value of s_K, where K takes the values of B_K.

This number characterizes the problems: 1 indicates that the problem is misleading, 0 means that it is very friendly. In section 4.3.3 this coefficient will be shown for some well-known problems. Note that this coefficient depends on a set of parameters. It is possible to give a coefficient for every stopping criterion by using the one element set containing the bound corresponding to that criterion.

4.3.3 Empirical results

In this section our method will be demonstrated via empirical results. A stochastic hill-climber was used with the same operator which has been mentioned in the section 4.3.2. First, a survey will be given of the studied functions and some explanation on how to read the iteration and the other figures. Finally our heuristic for determining the probabilities of the endpoints will be validated via some empirical results.

Studied functions

Some well-known functions have been examined with our method. These are the Ridge function, the Long Path problem, Liepins and Vose fully deceptive problem and the Subset Sum problem. The iteration figures (figures that show the expected number of evaluations for the space) will be shown for these functions and the deceptiveness coefficient will also be discussed. Note that in the iteration figures noise has been added to the distances and fitnesses so that identical points can be distinguished. For all the examined problems size was set to 10 bits and the probability of the mutation operator of the hill-climber was 0.1.

Ridge function

Ridge functions were introduced by Quick, Rayward-Smith and Smith in (Quick et al., 1998). In our experiments a 10-bit version was used. FDC predicts that ridge functions are very misleading while the hill-climber and the GA easily solves the function. Our results show that this function is easy. The ridge function hasn't got local optima as it is shown in the iteration figure. The iteration stopped after 488 steps, so the iteration figure shows that from every point in the search space the global maximum can be reached in 250 steps.

The iteration figure shows that there is a path in the search space, as follows from the construction of the ridge function. On the other hand, the points which are not on the path were gathered at the beginning of it. It is also clear that from an arbitrary point the global maximum can be reached only via walking through the path.

Let us examine the deceptiveness figure with respect to the number of endpoints. Obviously, from bound 1 to 0.96 there is only one endpoint, the global maximum so it is predicted that the hill-climber always finds the global optimum. One can easily see that when the bound is 0.96 there are only two endpoints and every other point tends to the worst one. This is why the deceptiveness is 1. When the bound is under 0.96 the number of the endpoints suddenly

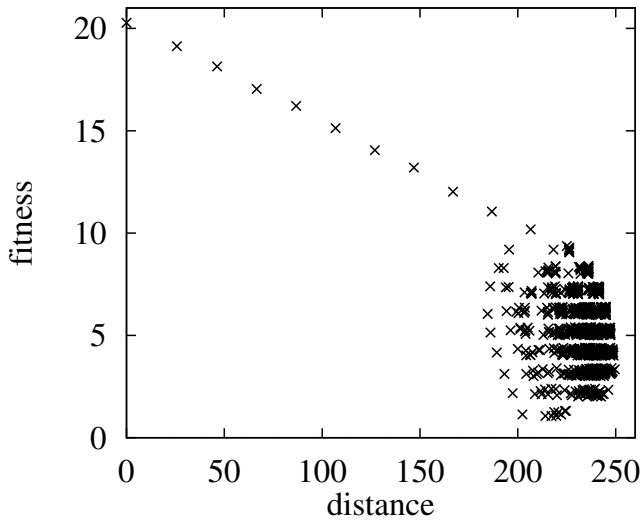


Figure 4.10: The iteration figure of the 10 bit Ridge function

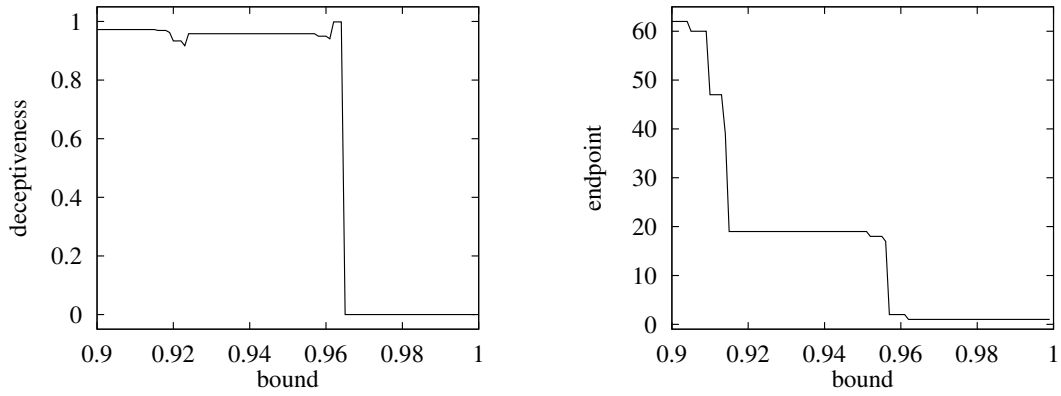


Figure 4.11: The deceptiveness figure and the number of endpoints of the 10 bit Ridge function

starts to grow, because all the points on the path become endpoints. This is the explanation of the jump of deceptiveness since as it has been mentioned the search process walks trough the path so most of the points tend to the beginning of it. Additional decreasing of the bound has essentially no effect on the value of deceptiveness because independently of the new endpoints all the remaining points tend to the beginning of the path.

Note that our coefficient which depends on the stopping criterion clearly shows that the friendliness of the ridge function as claimed by (Quick et al., 1998) heavily depends on the stopping criterion; the allowed probability of staying in place has to exceed 0.96 to make the problem friendly.

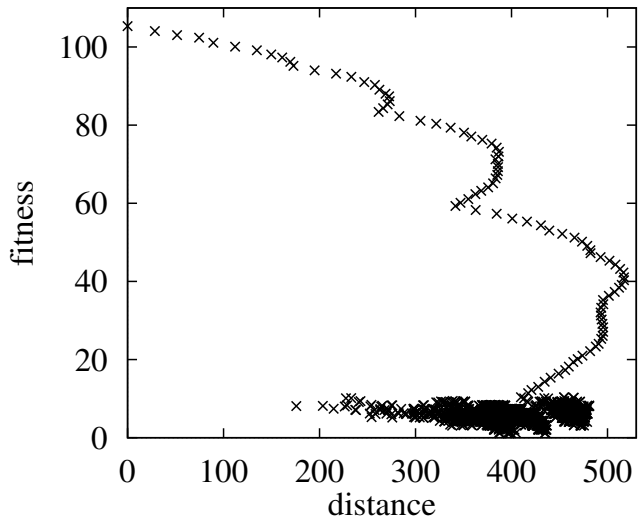


Figure 4.12: The iteration figure of the Long Path problem

Long Path

Long Path problem is introduced by Horn, Goldberg and Deb in (Horn et al., 1994). This problem was constructed to be difficult for the hill-climber. First of all, note that the problem size was set to 11 bits, because this function is defined as odd bits problem. In this case formula (4.5) converged after 1257 steps for all the points.

The iteration figure clearly reflects the structure of this problem, surely there is a long path. In the case when some points with low fitness are closer to the global optimum than points which are at the beginning of the path show that these points do not have to walk through the path, they are able to jump into it. It is interesting that the shape of the path is not linear. It means that inside the path bigger steps can be taken i.e. there are shortcuts. Recall that the interpretation of our plots is essentially different from the interpretation of the FDC plots. In our case the distance is the expected number of evaluations for reaching the optimum from the given point and this distance is not a direct function of the encoding alone as in the case of Hamming distance. This is why the structure of the plot indicates shortcuts why a similar structure in an FDC plot would simply indicate deceptiveness.

In figure 4.13 from bound 1 to 0.96 the situation is the same as with the ridge function. Under 0.96 the points of the path gradually become endpoints. The deceptiveness of the function in the area where the points of the long path become endpoints as the bound decreases is lower than in the case of the ridge function. This is due to the above mentioned shortcuts. When all the points of the path become endpoints there is no significant change anymore. Note that since in this case the path is longer than the path of the ridge function deceptiveness does not increase as fast. The other interesting result to note is that our figures clearly show the structural similarity between the long path and the ridge function.

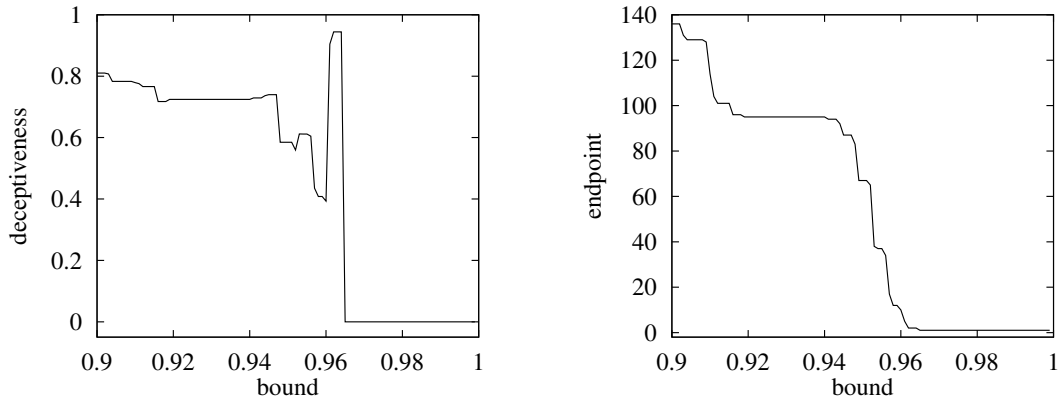


Figure 4.13: The deceptiveness figure and the number of endpoints of the Long Path problem

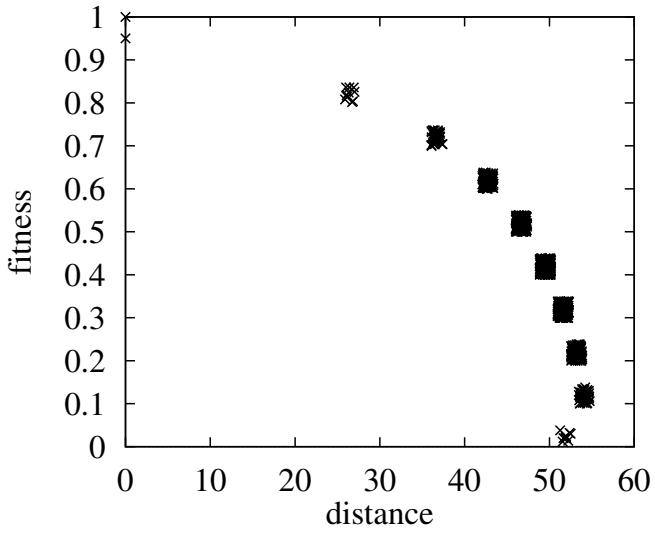


Figure 4.14: The generalized iteration figure of the Liepins and Vose function

Liepins and Vose fully deceptive function

This problem was introduced in (Liepins and Vose, 1991a). This function is fully deceptive. There is one global optimum and a local one. The global optimum is fairly independent of the whole structure of the function so no trajectories converge to it. The only possibility to find it is blind coin tossing.

If there is only one endpoint, the global maximum, then Eq. (4.5) converges for none of the points after 20000 iteration steps. It means that from all points of the search space the global optimum is unreachable under these conditions. That is why the iteration figure is meaningless though is clear that the points are very far from the optimum. For this reason Eqs. (4.7) and (4.8) were used.

The generalized iteration formula with bound 0.999 results in figure 4.14. There are two endpoints, the global and the local optimum. In this case all points convergent after 215 iteration

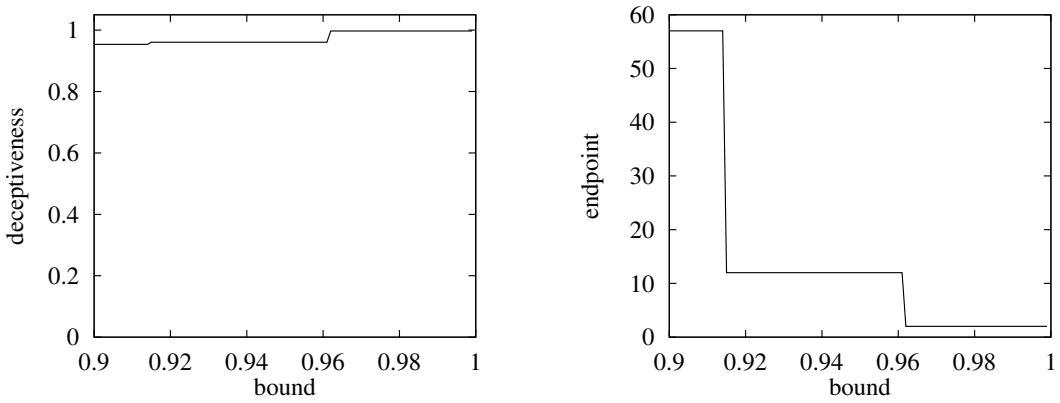


Figure 4.15: The deceptiveness figure and the number of endpoints of the Liepins and Vose function

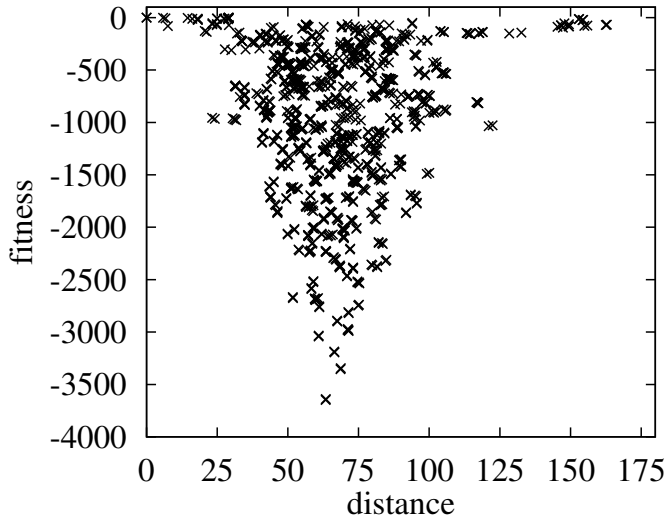


Figure 4.16: The iteration figure of the Subset Sum problem

steps, so it is clear that all points can easily reach the endpoints.

Examining the deceptiveness figure it is obvious that all points tend to the local optimum, even if the number of endpoints grows. It can be seen that this function is consistently deceptive independently of the stopping criterion (at least in the range we examined).

Subset Sum problem

The subset sum problem is a combinatorial optimization problem. In this problem we are given a set $W = \{w_1, \ldots, w_n\}$ of n integers and a large integer N. We would like to find a $V \subseteq W$ such that the sum of the elements in V is closest to N. For more details see (Khuri et al., 1993). We examined a 10 bit problem here.

Unlike the previous functions this is a highly multimodal function as it can be seen in the

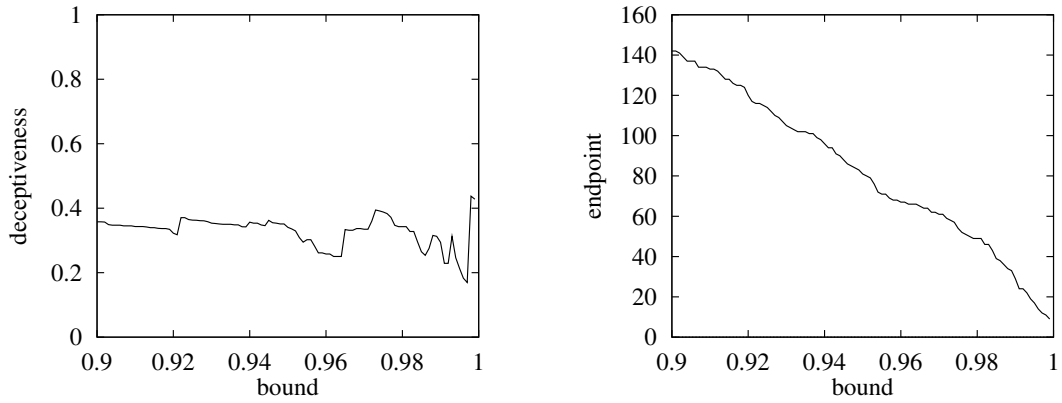


Figure 4.17: The deceptiveness figure and the number of endpoints of the Subset Sum problem

iteration figure. There are many points with high fitnesses which are far from the global optimum. Note that there are few points for which the formula had not converge, but the iteration figure correctly shows the structure of this function. Note that in this case the maximum iteration of formula (4.5) was 3000 and 1005 points converged. Let us note again that although the FDC plot of the subset sum problem is very similar in this case it does not mean that it means the same.

Let us examine the deceptiveness figure. Deceptiveness never goes above 0.5 so it shows that although this problem has many local optima most of the points tend to the endpoints with good fitness. Note that there are also large fallings and jumps because of the few endpoints, but the deviation of the deceptiveness is very little. So in this case the deceptiveness coefficient is very informative. An important observation about the figure of the number of endpoints is that in this case the number of endpoints grows smoothly. This is the reason why in the deceptiveness figure there is no plateau.

These results are in agreement with earlier investigations concerning the subset sum problem (e.g. Section 4.2 and (Khuri et al., 1993)) where it was found that the problem is easy (with the suggested encoding and with the applied test problem generation method) and according to our earlier results the subset sum problem indeed has an enormous number of relatively good local optima and these local optima can be very far from each other.

Discussion

This section gives a guide on how to read the figures in general. First notice that iteration figures are not correlation figures so the information shown by them is only indirectly related to our figures based in the approximated expected number of evaluation (referred to as iteration figures). If the iteration formulas converge for all points then the horizontal axis of the iteration figure shows the expected value of the steps from a given solution to the global optimum (or the set of endpoints). The structure of the plot is not necessarily important e.g. it is not necessarily good to have a linear form like in the case of FDC. However if there are many points with good fitness far from the global optimum then the given function has many good quality local optima.

The figure showing the deceptiveness coefficient as a function of the stopping criterion is

function	dec.coeff.	deviation
Ridge	0.63	0.46
Long Path	0.46	0.36
Liepins and Vose	0.973	0.018
Subset Sum	0.33	0.046

Table 4.3: The deceptiveness and its standard deviation for the studied functions

very interesting. First of all it should be noted that the coefficient is most informative if the deviation is small. If the number of endpoints is small then the deviation may be large since as additional endpoints jump in with the decreasing bound the value of the coefficient may be altered significantly. However most of the real problems behave like the subset sum problem i.e. they have many good local optima. On the other hand, regions with small deviation carry much information about the structure of the problem; the sudden change of the deception coefficient in the case of the ridge and long path problems is related to their special structure.

Table 4.3 shows the deceptiveness coefficient and its standard deviation for the studied functions. It can be observed that in the case of the fully deceptive problem the result is correct and the same can be said about the Subset Sum problem. However, in the case of the other two functions these values are not so informative because of the large deviation but here the deceptiveness as the function of stopping criterion still accurately shows the structure of the space.

Finally note that according to this measure the ideal function is the constant function. This is quite evident since in that case we can reach the global optimum with zero evaluations with any kind of parameter setting. Strangely enough, there is a tendency among the known measures to regard the constant function as hard. FDC and the measure suggested in (Naudts and Kallel, 1998) are good examples. This is related to the observation that plateaus are hard for hill-climbers and if a partial sample of the space contains similar elements then the problem can be expected to be hard. However if the plateaus have good fitness then why would it be a problem to get stuck in them?

Validation of the Model

In this subsection we have a look at that the expected value of fitness as predicted by the same heuristic used for calculating the deceptiveness coefficient as was mentioned in section 4.3.2. For every bound the averages of 500 hill-climber runs are shown.

In figure 4.18 it can be seen that the sudden performance growth predicted by the spanning forest over the transition probability graph is not followed by the hill-climber. Note that at the sudden growth of the prediction the standard deviation of the average results of the hill-climber is very large.

On the other hand in the case of the Subset Sum problem the prediction coincides with the real result. This is related to the fact that in this case the number of the endpoints decreases smoothly with the increasing bound. In the case of the Liepins and Vose function the predictions follow the observed behavior. These plots show that the heuristic used for determining the weights of the endpoints is valid since using the weights the weighted sum of the fitnesses of

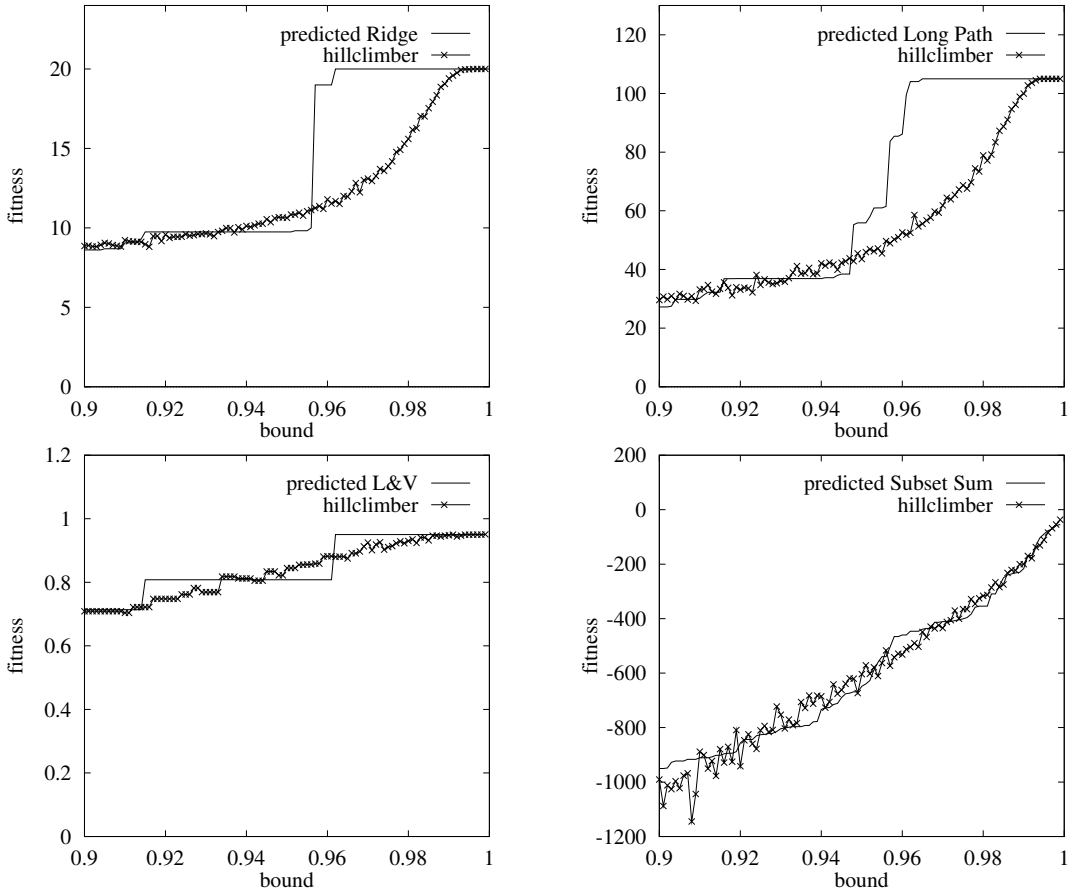


Figure 4.18: The prediction versus the results of the hill-climber

the endpoints predict the result of the hill-climber quite well.

4.3.4 Summary

In this section we suggested that the hardness measures of functions should take the local optima into account. The suggested measure is based on the fitnesses and weights of endpoints, the solutions to which the optimizer can be expected to converge. The weight is the probability that the optimizer will converge to the given point if started from a random solution. This measure depends on the stopping criterion of the optimizer since the set of endpoints depends on this parameter.

A heuristic was also suggested for calculating this coefficient. The time complexity of the method is $O(|S|^2)$ so only relatively small spaces can be considered. This method involves the calculation of the transition probabilities between solutions w.r.t. an operator. These transition probabilities were used also for approximating the expected number of evaluations needed to reach the global optimum and plots were presented showing these values for several spaces.

In general if the number of relatively good local optima is large then this measure seems to be reliable but if this is not true then the behavior of the coefficient as a function of the stopping criterion still carries a lot of information about the function at hand. A possible source of problems can be if the deviation of the fitnesses of the endpoints is too small since the coefficient is relative to the minimal and maximal fitness of the set of endpoints. This problem can be solved if the lowest acceptable fitness is given as a parameter.

Chapter 5

Human or Automated Design?

This chapter is organised around the problem of collecting and representing domain knowledge. This issue has two aspects that are strongly connected. One aspect is the practical way of extracting this knowledge. This is the subject of Section 5.1. The other aspect is the philosophical limitations of such knowledge extraction algorithms and also the relation of the extracted knowledge to our present scientific knowledge. Section 5.2 seeks answers to these problems, and furthermore attempts to place this issue in the broader context of adaptationism.

5.1 Automatic Generation of Representations

Domain knowledge is essential for successful problem solving and optimization. This section introduces a framework in which a form of automatic domain knowledge extraction can be implemented using concepts from the field of machine learning. The result is an encoding of the type used in most evolutionary computation (EC) algorithms. The approach focuses on whole problem domains instead of single problems. After the theoretical validation of the algorithm the main idea is given impetus by showing that on different subdomains of linear functions the method finds different encodings which result in different problem complexities.

5.1.1 Introduction and Motivation

Domain knowledge plays a key part in today's machine learning applications. Though in many cases relatively simple heuristics combined with the brute force of available fast processors and millions of test samples seems to be the best available solution — like hidden Markov models used in speech recognition systems rather than an expert knowledge of phonemes (Rudnicky et al., 1994), or simple Bayesian models employed in natural language processing applications instead of knowledge of grammar (Mitchell, 1997) — it is now widely accepted that for instance the performance of evolutionary heuristics depends heavily on the applied encoding and operators. In fact this is consistent with what the "no free lunch" theorems (Wolpert and Macready, 1997) tell us: there are no algorithms that are the best in each domain, so for the best performance in each domain one has to find the best algorithm for each separately. Hence the extraction of domain knowledge is essential.

Our research into the question also supports this view (see Chapter 4). We have shown that the usual practice of simply characterizing domains by giving them labels such as "NP-hard" or "subset-sum problem" is not necessarily useful or even misleading. The actual structure and complexity of a domain (i.e. a set of functions defined over a common search space) depends on the source of these functions. For example a domain containing subset sum problems (i.e. "NP-complete" combinatorial optimization problems) may turn out to be a trivial domain due to the *structure of the parameters* of the particular functions in the domain. So even when a characterization is available (the function is not a black-box) the extraction of domain knowledge is still essential. In Section 5.1.4 it will be shown that even when we know that a domain contains only linear functions the performance can still be significantly improved using domain analysis.

The problem is that extracting domain knowledge in general is quite a difficult problem; scientific researchers and engineers do this for a living and it is not one of the easiest jobs available. However, in systems where knowledge is explicitly and separately represented, it is possible to perform some kind of meta optimization over the domain of possible knowledge content. Evolutionary heuristics are good examples since knowledge is expressed in the encoding and operators while the basic algorithm remains the same. A lot of methods can be found in the literature that tackle the problem of dynamic problem structure analysis. A survey of methods using probabilistic models can be found in (Pelikan et al., 1999). Other approaches concentrate on linkage detection (Munetomo and Goldberg, 1999). A common feature of these methods is that they concentrate on single functions. Our goal here is different in that we would like to extract knowledge that characterizes a whole domain and is reusable and interchangeable. One area of research is relevant from this point of view, namely the work of Radcliffe (Radcliffe, 1994). Their basic ideas on the nature of knowledge to be extracted are not unlike those presented here (the differences being emphasized later on) but they did not tackle the problem of extracting knowledge automatically in a systematic way.

This section introduces a framework in which automatic domain knowledge extraction is possible. In our case domain knowledge means the representation or encoding of the search space. Here, binary representations will be generated that are optimal in a sense to be described later. A binary representation is a mapping of the search space to a set of the binary strings $\{0,1\}^n$. Though it is now widely accepted that an *arbitrary* binary representation is not necessarily better than an *arbitrary* non-binary encoding, our problem is a little different here. We are looking for the *optimal* binary representation in the space of all binary representations of a search space. Note that e.g. a search space of size 2^n has $2^n!$ different n-bit binary representations which is an enormous number. It is still possible that the *optimal* binary representation is not optimal in the space of all representations but here we do not tackle this problem.

In a binary representation every position of this string contains a 0 or a 1 which means that every position defines a *concept* over the search space. The term concept is used as in machine learning, i.e. a concept over a space is a subset of the space. Very briefly, our method is based on finding such concepts with the help of a measure over the concept space.

The outline is as follows. In Section 5.1.2 the basic concepts of the framework are defined. In Section 5.1.3 a useful property of the method is proved which supports applicability. Section 5.1.4 provides an illustrative but interesting example of the possible advantages of the

approach on the class of linear functions. In Section 5.1.5 it will be shown that this method is in fact a generalization of some probabilistic methods used to model the distribution of good solutions of a function (see (Pelikan et al., 1999)). Finally, Section 5.1.6 discusses the possibilities and limitations of implementation of the framework in real-world, large-scale problems.

5.1.2 Framework

It is important to emphasize that this section will define only a general framework which may have many implementations depending on problem size, available time and data, etc. These details will be discussed in the following sections.

Basic Terms

Let us first define a problem. A problem is given by its *search space*, and a real valued *objective function* defined over it. The notation of the search space is S. The notation of the function is $f : S \longrightarrow \mathbb{R}$. In other words $f \in \mathbb{R}^S$. In evolutionary computation the objective function is usually called the *fitness function*. Here I will adopt this convention.

The *problem domain* is a subset of all possible fitness functions. The problem domain will be denoted by $\mathcal{D} = \{f_1, f_2, \ldots\} \subseteq \mathbb{R}^S$. This notion is crucial from our point of view. In practice the problem domain is given by the problem situation, e.g. a university which needs schedules for organizing its activity. The particular variables of the particular university — i.e. the number of employers, students, rooms, the sizes of rooms, the habits of each lecturer (who get up early/late, work at home/in their office etc.) and so on — will make the scheduling task special. The fitness functions in the domain will probably have a lot of features in common. At the same time, the scheduling task is an NP-complete combinatorial optimization problem in general. But this mathematical definition includes many more functions which are very diverse compared to the ones actually encountered at our university. To handle problem domains as an actual sample of functions and trying to describe them instead of using a given definition is therefore a main constituent of the philosophy of the present approach.

A *concept* over S is a subset of S. The notation will be $C \subseteq S$ while $\overline{C} = S \setminus C$. In other words, using a function notation $C \in \{0,1\}^S$. An *encoding of* S is given by an ordered list of concepts. The encoding will be denoted by $\mathcal{C} = (C_1, \ldots, C_n)$. Using this notation, the *code* of a solution $x \in S$ is given by $\mathcal{C}(x) = (C_1(x), \ldots, C_n(x)) \in \{0,1\}^n$. For the sake of simplicity these binary codes will be used noting that generalization is possible to other kinds of codes.

Next let us define two properties of encodings. The first is very important from a practical point of view: an encoding has to be *invertible*, i.e. given a code c of a solution, we should be able to effectively compute solutions $x \in S$ for which $\mathcal{C}(x) = c$. Note that it is possible that x is not unique. The second is related to the efficiency of the encoding. We want as few concepts as possible. To express this we call an encoding *independent* if its concepts are stochastically completely independent, i.e.

$$\forall k, i_1, \ldots, i_k \quad Pr(x \in C_{i_1}, \ldots, x \in C_{i_k}) = Pr(x \in C_{i_1}) \ldots Pr(x \in C_{i_k})$$

This seemingly contradicts other results from the GA literature, which report that non-coding segments (introns) may improve the search (e.g. (Mitchell et al., 1994)). This may be true

under the assumption that the encoding is not optimal and so the genetic drift introduced by the small population-size has a larger impact. With optimal encodings which will be defined later on the genetic drift is a smaller problem while with few concepts the search space size reduces significantly.

Automatic Generation of Codes

The task is to generate the optimal encoding for a problem domain. If we define the notion of optimality then the problem reduces to a search problem over the possible encodings.

First let us define the optimality of a concept over a domain. We need a concept that separates good and bad solutions as clearly as possible in all of the functions in the domain. Good (or bad) solutions are defined as being in the upper half (or the lower half) of the search space with respect to a given fitness function. Let us denote the concept representing exactly the good solutions for the fitness function f by G_f. Clearly every fitness function will define a different notion of good and bad solutions. First we define a measure of separation of a concept over a given fitness function. Clearly, the optimal concept over a given function f must be G_f. Then we apply the average of the values of this measure taken over all the fitness functions as a measure of separation quality over the entire domain.

For measuring the separation of good and bad solutions over a given fitness function *information gain* is an ideal choice. Information gain is a measure of "goodness" of cutting a space. Before cutting the space we calculate the number of bits that are needed on average to encode if a random solution is good or bad (i.e. a member of G_f or not). This measure depends only on the size of G_f. This value is called the entropy ($E(\mathrm{p})$) of the distribution of the probability variable that has two values: good and bad, where $p = P(\mathrm{good}) = |G_f|/|S|$. In the worst case one bit is needed (if the number of good solutions equals the number of bad ones) and in the best case no information is needed (0 bits) if all the solutions are good or bad. After cutting the space in two using C so that $S = C \cup \overline{C}$, the entropy restricted to the two resulting subspaces can be calculated. If the cut is good, these entropies will be smaller than the original entropy of the whole space. The difference of the average of the entropies of the two half spaces and the original entropy is the gain.

We use information gain as defined in the classical ID3 algorithm (Quinlan, 1986). For a given fitness function f from the domain the information gain of a concept C is defined as follows:

$$\mathrm{gain}(G_f, C) = E(\frac{|G_f|}{S}) - \frac{|C|}{|S|}E(\frac{|G_f \cap C|}{|C|}) - \frac{|\overline{C}|}{|S|}E(\frac{|G_f \cap \overline{C}|}{|\overline{C}|})$$

where function E is the entropy defined by $E(p) = -p \ln p - (1-p)\ln(1-p)$. Here p is the proportion of a given concept over the space under consideration. The natural logarithm was chosen because natural logarithm is equivalent to $\log_2$ as a measure of information according to information theory but our formulas will become simpler using $\ln$. $E(0) = E(1) = 0$ while $E(0.5)$ is maximal. This means that the information gain is maximal if $C = G_f$ or $\overline{C} = G_f$, and minimal (0) if C and G_f are independent. The measure we are seeking will be the average information gain of the concept over the functions in the domain. This measure is denoted by

$C_1 = \arg\max \text{gain}(C)$
$g = \text{gain}(C_1)$
$i = 1$
while($g >$ random gain)
    $C_i = \arg\max_C \text{gain}(C \mid C_1, \ldots, C_{i-1})$
    $g = \text{gain}(C_i \mid C_1, \ldots, C_{i-1})$
    $i \longleftarrow i + 1$

Figure 5.1: The algorithm for finding the optimal encoding.

$\text{gain}(C)$. This means that a concept is an *optimal concept of the domain* $\mathcal{D}$ if it maximizes the average information gain over $\mathcal{D}$.

Since a useful encoding contains several concepts we need a method for finding additional concepts while preserving the mutual independence between the concepts. A good heuristic for doing this is to find the concepts iteratively, one by one, and then applying the definition of optimality to the subdomains defined by the inverse sets of the possible codes determined so far. For example two concepts define four possible codes. The inverse sets of these codes yield a classification of the search space defining four subsets. These subsets define four subdomains by restricting the functions of the original domain. An optimal concept can be found in each of these subdomains. Now let the third concept of the encoding be the union of these four optimal concepts. The rationale behind this heuristic is that this recursive construction ensures independence if the optimal concepts divide the search space in two equal parts. Of course this will not be true in general.

This method provides us with a definition of the information gain of the concept C_{i+1} given that $C_1, \ldots, C_i$ are known (denoted by $\text{gain}(C_{i+1} \mid C_1, \ldots, C_i)$). For this let us take the information gain values of C_{i+1} restricted to each of the subspaces defined by the known concepts as described above and let the information gain of C_{i+1} be the weighted average of these gains where the weights are proportional to the sizes of the corresponding subspaces. The algorithm for finding the optimal encoding is given in Figure 5.1. The algorithm stops when the gain of the new concept is not greater than the optimal information gain over a domain containing only random functions.

5.1.3 Theoretical Foundations

In this section I will show that the algorithm described in Section 5.1.2 is optimal in an important sense: random domains are never divided by any concept if some assumptions hold. This means that the subdomains of the original domain defined by the inverses of the codes are either random or empty. We say that a subdomain is random if it contains only random functions i.e. the values of the functions are drawn from the same distribution. Besides this the random subdomains are maximal i.e. every larger domain becomes non-random. In other words it is impossible to gain more information from the space by refining the encoding and the information contained in the encoding cannot be expressed using fewer random classes.

According to our algorithm we have to find the optimal concept on a domain, the concept

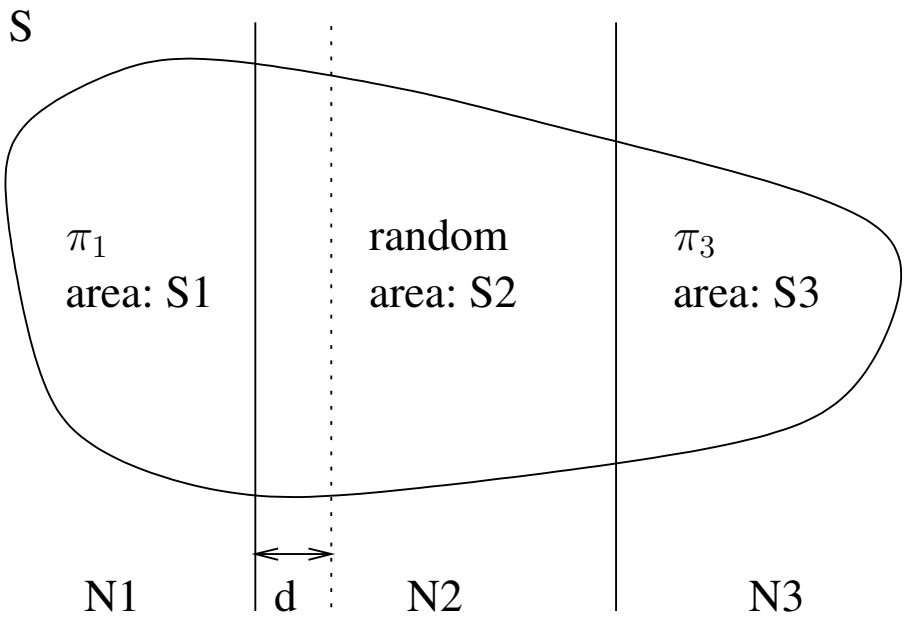


Figure 5.2: Illustration of the notations.

with the maximal information gain. Let as assume that our domain with search space S contains a random subdomain S_2 (see Figure 5.2 for an illustration). I will show that for every concept that splits this random space there exists another concept which has a larger gain and which does not split S_2.

Now let us choose an arbitrary concept C. The dotted line in Figure 5.2 is the boundary of the concept. The subspace $S_1 = C \setminus S_2$ is the non-random part of C and $S_3 = \overline{C} \setminus S_2$ is the non-random part of $\overline{C}$. The sizes of the classes are $|S_i| = N_i$, $i = 1, 2, 3$. π_1 is the value for which

$$E(\pi_1) = \frac{1}{|\mathcal{D}|} \sum_{f \in \mathcal{D}} E(\frac{|G_f \cap C|}{|C|})$$

There are two such values since $E(p) = E(1-p)$. Let π_1 be the smaller one. With a similar definition of π_3 the gain of C now can be written as

$$\text{gain}(C) = K - \frac{|C|}{|S|} E(\pi_1) - \frac{|\overline{C}|}{|S|} E(\pi_3)$$

where K is a constant independent of C. Using this value we introduce a simplification assumption:

$$\text{gain}(C_d) = K - \frac{|C_d|}{|S|} E(p_1(d)) - \frac{|\overline{C}_d|}{|S|} E(p_2(d)) \tag{5.1}$$

where

$$p_1(d) = \frac{\pi_1 N_1 + 0.5d}{N_1 + d}, \quad p_2(d) = \frac{0.5(N_2 - d) + \pi_3 N_3}{N_2 - d + N_3}$$

and C_d is any concept that was created from C by adding d elements from S_2. The point is that we replace the average of the entropies with the entropy of the average of $|G_f \cap C|/|C|$ over the domain. This introduces some error since it is possible only for linear functions of probability variables and entropy is not linear. This error depends on the actual distribution of $|G_f \cap C|/|C|$ over the domain. The lower the variance the higher the accuracy of the approximation. Furthermore, recall that we have chosen the smaller values for π_1 and π_3 so we use only the first half of the entropy function (over the interval $[0, 0.5]$) and here (apart from the neighborhood of the ends of the interval) it is not very far from linearity, so the accuracy depends also on the actual values of π_1 and π_2.

Now we can prove the following theorem:

Theorem 8. *Using the assumption in (5.1)*

$$C_i = \arg \max_{C \in \{C_0, \ldots, C_{N_2}\}} \text{gain}(C)$$

holds only for $i = 0$ *or* $i = N_2$.

Proof. We are looking for the maximum of the information gain

$$\text{gain}(C_d) = K - \frac{(N_1 + d)E(p_1(d)) + (N_2 - d + N_3)E(p_2(d))}{N_1 + N_2 + N_3}$$

The problem is equivalent to finding the minima of the counter of the fraction, the average entropy. I will show that this function is concave on the interval $[0, N_2]$ which directly proves the theorem. It is sufficient to show that the first term $(N_1 + d)E(p_1(d))$ is concave, the other term has a symmetrical structure and the sum of concave functions remains concave. The second derivative of the first term is

$$\frac{-1}{2(1 - \pi_1)N_1 + d} + \frac{-1}{\pi_1 N_1 + 0.5d}$$

which is negative so the proof is complete. □

This theorem means that either S_2 is included in the optimal concept or it is excluded completely. Applying this to every subdomain that arises during the running of the algorithm we get the result mentioned in the introduction of this section. It is very interesting to briefly relate this finding to Radcliffe's notion of a good encoding (Radcliffe, 1994). According to his model, the equivalence classes of the encoding should have a low fitness variance. This can be applied not only to single functions but to domains as well since it is possible to take e.g. the average of the variances of the functions of the domain. This is a special case of our approach since if the low variance property holds over a subdomain then according to our approach it will be a good candidate for being an optimal concept since all the solutions will tend to be good or bad due to low variance and therefore the entropy will tend to be low. However in the case of random domains the variance is not necessarily low.

5.1.4 Structure in the Linear Domain

For illustration of the potentials of the approach let us take a closer look at a domain of special significance: the linear functions over the binary search space $S = \{0,1\}^n$. A function $f : S \longrightarrow \mathbb{R}$ is linear with the coefficient vector $\mathbf{a} \in \mathbb{R}^n$ if

$$f(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} = \sum_{i=1}^{n} a_i x_i$$

In this section three subdomains of the general linear functions will be studied. Every subdomain will have prototypical functions, but the definitions are intended to be fuzzy. The closer examination of these subdomains is useful for two reasons. The first is that we will see that on certain subdomains the efficiency of the search can be significantly improved. The second is that this discussion will illustrate a major point of this work: different domains may have dramatically different structure and thus different optimal encodings even if the mathematical description of the functions of the domains have the same form.

Some claims of the section are based on experimental data. In all the experiments 8 bit domains were used and a concept was implemented as an explicit characteristic function (i.e. a list of 256 truth values). The concepts were optimized using a simple multistart hillclimber run until 10000 evaluations restarted when no one-bit change resulted in improvement. The other specific details are given in the subsections.

Orderable Problems

The coefficient vector of an *orderable* function contains numbers that differ in their order of magnitude significantly. In other words the coefficients (and thus the bits of a solution) can be ordered according to dominance. Prototypical examples are vectors with $|a_i| = 2^i$, $(i = 1, \ldots, n)$. It is easy to see that for orderable domains the optimal encoding will be the collection of schemata of length 1. As an additional benefit, the dominance order is also given by the algorithm.

Counting Problems

Here the coefficients do not differ in magnitude and they have the same sign. The coefficients of the prototypes of such problems are equal to a given constant: $a_i = c$, $(i = 1, \ldots, n)$. We have run experiments with domains containing 100 functions where the coefficients of a particular function were drawn from $[100, 120]$ (or $[-120, -100]$) to introduce some noise. Note that the value of solutions which have the same number of 1s is similar. Thus they generate random subdomains in the sense of Section 5.1.3.

The experiments confirmed our theoretical assumptions in that the concepts found during search never divided such a subdomain in any single run, only when the whole domain to divide was random. Surprisingly (to me), when trying to divide such a random domain the algorithm did find structure consistently. Closer analysis showed that this structure is due to the noise we introduced and can approximately be translated into an additional heuristic which says that in

a space of solutions containing the same number of 1s divide the space using the bit which has the smallest coefficient on average.

Note that the length of the optimal encoding is proportional to $\log n$ so a significant reduction of the search space can be achieved while the fitness variance of the inverse sets of the codes is low therefore this subdomain with the optimal encoding is much easier than the general linear domain.

Hamming Problems

Here the coefficients do not differ in magnitude but they may have different signs. A prototypical example could be $a_i = (-1)^n$, $(i = 1, \ldots, n)$. The value of a Hamming function depends on the Hamming distance from a given binary vector.

Experiments were run using a 100 function domain where the coefficients of a particular function were drawn from $[100, 120]$ and their sign was random. The maximal information gain of the first concept that was found by the hillclimber was 0.048 with a variance of 0.003 (from 10 experiments). This value is quite low given that on completely random domains the expected maximal gain is around 0.01 according to our simulations, and in the case of counting problems this value is 0.39 on average. Analyzing the optimal first concepts we can define the following heuristic: divide the space according to a one bit schema. Applying this heuristic explicitly we get a gain of 0.046 on average with a variance of 0.002 (10 experiments).

The conclusion is that the optimal encoding is the natural encoding as in the case of orderable problems but the information gain is significantly lower. This indicates that Hamming problems are harder then orderable problems since the fitness variance is much larger within the schemata and the problems are much more sensitive to sampling error and genetic drift.

5.1.5 Probabilistic Models

The approach presented here can be considered as a generalization of search techniques that use dynamic probabilistic models to generate good solutions (Pelikan et al., 1999). A probabilistic model of the good region of the space has a close relationship to our notion of concept. As mentioned earlier, a concept has to be invertible; we have to be able to generate solutions that satisfy a given concept. A probabilistic model is in fact a fuzzy concept which is of course invertible.

A practical implementation of the algorithm applied to a domain containing only a single function may be very similar to algorithms using probabilistic models since during the recursive building of the optimal encoding the gain of new concepts can be evaluated on solutions generated using the inverses of available codes. Furthermore — as a trivial extension — every subdomain can be labeled positive if the functions over it contain good solutions consistently (recall that large information gain requires only homogeneity), and emphasis can be moved to explore those regions further, even if the domain contains several functions.

5.1.6 Conclusions and Future Work

Our aim was to motivate, to theoretically ground and to illustrate an automatic encoding generation technique. We have seen that the method cuts search spaces along their "natural joints" in the sense that random domains are never cut in half. This also means — considering the structure of the algorithm as well — that the non-empty inverses of the optimal codes define either random domains or low fitness variance domains. It was demonstrated that even in the case of the linear functions three subdomains can be defined that have significantly different complexity. This also implies that similar or identical mathematical structure is not necessarily sufficient to characterize a domain: the distribution of the parameters of the functions is also essential.

Here I would like to touch on some problems of practical, real world applications and its limitations. One main problem to solve when implementing the system is to chose the actual representations of the abstract notion of concept. In the case of big spaces this representation is naturally a function class. The literature on machine learning provides us with an endless number of opportunities, the class of feedforward artificial neural networks (ANNs) is a good example. The only important constraint is the invertibility condition of the encoding.

Another important issue is the bias introduced by the chosen representation. When restricting ourselves to a specific function class we risk the possibility that we cannot describe the structure of the domain under consideration. For example if parity of bits plays an important role in a binary domain then there is practically no chance to capture this using feedforward ANNs. However this is not a specific problem of the present approach: it is the problem of machine learning in general.

Finally, for finding good concepts samples of the functions in the domain are needed. Two natural approaches seem to be reasonable. The first is to use the trajectories that were produced by search algorithms on the functions of the domain. The other is to recursively generate new solutions based on the available concepts and to evaluate them. Both methods assume a larger time-scale than ordinary optimization methods but the output, the interchangeable and reusable knowledge about important problem domains may pay off in the long term.

5.2 Human or Automated Design?

In this section the connections between the evolutionary paradigm called adaptationism and the field of evolutionary computation (EC) will be outlined. After giving an introduction to adaptationism we will try to show that the so called adaptational stance can be applied in EC as well as in biology and this application may have significant benefits. It will also be shown that this approach has serious, inherent limitations in both cases especially in the case of EC, because we lack the *language* which could be used to form the theories, but these representational limitations can be handled by devoting efforts to construct this language.

5.2.1 Adaptationism

This section introduces adaptationism, a strategy for understanding the products of evolution. We will discuss only biological evolution here; the discussion of the relationship with EC is given in Section 5.2.2. Adaptationism is a controversial question (Gould and Lewontin, 1979) and it seems that most of the misunderstandings can be originated from the insufficient and obscure definition of the paradigm.

Basic Notions

In this section we will try to make the assumptions of adaptationism clear and explicit and to show the role of the underlying language. First it seems that adaptationism requires the following principles in order to be applicable:

P_1 **(separation):** The separation of the organism under investigation and its environment is necessary. To give an example, it is not clear whether the ant colony or a single ant counts as an organism. To be more precise both approaches are valid; only the choice has to be fixed. If the organism is the ant then the ant colony is a part of the environment that has to be constant according to P_3. This assumption can be difficult to maintain though it might me meaningful if the time interval under investigation is relatively small. This is a very controversial question, see (Dawkins, 1989) and (Wilson and Sober, 1994) for two quite different viewpoints.

P_2 **(constraints):** It is necessary to determine the constraints that give the space of the biologically feasible genotypes. These constraints help us exclude caws with machine-guns, birds with jet engines, etc. This principle is needed to allow the optimality condition to be well-defined.

P_3 **(frozen environment):** Optimality can be given only w.r.t. a fixed environment; this makes it possible to block the circular definition of fitness[1]. Since the organisms cannot change the environment, the number of their offspring is solely the function of their properties. We must pay the price for this however; no dynamic properties of the population or its interaction with the environment can be examined. Of course, this limit diminishes if we choose a larger entity such as a population or an even larger subsystem such as a food chain to be our organism.

P_4 **(one niche):** Beside P_2 it is also necessary to restrict the possible individuals further only to those occupying a particular niche; roughly speaking the possible organisms cannot be *too* different. Optimality can be defined only inside of a species; this ensures that a pig will not be compared to a lion. Both may well be optimal in their own way but it would be a serious mistake to consider a pig a bad quality lion or vica versa. This constraint seems to be simple but in fact it causes major difficulties especially when fossils are analyzed since it is hard to tell with respect to what the organism in question should be optimal (is it a bad pig or a bad lion?). The definition of niche is itself a problem.

P_5 **(improving fitness):** We need to assume that evolution improves fitness. This will be one of the motivations of the optimality assumption to be introduced soon.

[1] If the number of offspring define fitness then this fitness cannot be used to predict the number of offspring since the claim "Individuals with high fitness will have high number of offspring" becomes a tautology. The problem is that this claim is in the center of Darwinism

P_6 **(single organism):** This is partly the consequence of P_1; at every time-step there is only one organism. It may seem to be counter-intuitive at first sight. But according to P_1 we have to decide what the definition of organism is. If we choose an *individual*[2] animal or plant then all the other members of its species become part of the environment and according to P_3 they are not allowed to change while our organism changes. It is clearly a plain contradiction though for *short* time intervals this approach could give fairly good approximations; in fact such discretization methods are quite common in mathematics for example in numeric approximations of differential equations.

Let us introduce a notation for the most important components. Let O be the set of organisms that are biologically feasible according to P_2 and live in a particular niche according to P_4. Let f be the fitness function, i.e. a function of type $O \to \mathbf{R}^+$. For an organism $o \in O$ the number $f(o)$ gives the expected number of offspring of o. f is also a function of the environment but since it is constant according to P_3, it is not indicated. Let $\mathcal{G}$ be the finite[3] set of features that make it possible to describe the organism. Its elements are functions of type $O \to \mathbf{R}$. Predicates are possible as well, in that case the function has only two values: 0 for false and 1 for true. An example could be the length of the neck or the degree of flying ability.

Characterization of Features

I would like to emphasize as early as possible that the characterization will be *independent* of the *function* of the given feature in the organism. At this level nothing is said about the roles or the causal relationships of the elements of $\mathcal{G}$. For instance we can say that the neck-length of a giraffe is optimal without mentioning its function (which could be for example reaching leaves at the top of the trees).

The aim of this characterization is to decide whether a given feature is relevant or irrelevant w.r.t. the fitness and if relevant then it is optimal or not. Therefore there are three categories: irrelevant, optimal and suboptimal. The interesting question is of course to find a method to somehow classify $\mathcal{G}$ into these three classes.

First let us examine the case when the fossil record is available that show the development of the feature under investigation. P_5 will be heavily exploited. P_6 is used too to ensure that there is no variance to be taken into account (only as another feature).

In the case of the *divergent* behavior we can conclude that at the moment the feature at hand is in a developing stage and therefore is *suboptimal*. In the case of *random* behavior we classify the feature as *irrelevant*. The situation is somewhat more difficult however. The random behavior can be caused by a lot of factors. The first possible explanation is that the given feature is neutral; it is independent of the fitness of the organism. The second explanation is that the given feature changes according to some kind of dynamics that is out of the scope of our analysis. For example if our organism was chosen to be a species then some features may

[2]This choice does not mean that we are interested in a particular individual. This only means that in every generation we are interested in one (hypothetical) individual which is subject to natural selection in its population.

[3]Note that it is not the set of all possible features; it contains the actual features that are used (or will be used) by the biologists. It is awkward to emphasize that this set is finite since it is trivial. It is a habit however that some people might miss.

vary in accordance with evolutionary game theory (Maynard Smith, 1982). A third explanation can be that our assumption P_3 about the constant environment is false.

In the case of *convergence* we say that the feature is *optimal*. There are lot of error possibilities however. The most important is that an irrelevant feature may converge as a result of genetic drift. Note that on the other hand it is impossible that relevant features show random behavior.

The situation without fossile record is more interesting since we do not have any ground to classify the features. The question is very sensitive since one has to decide which features are relevant and among the relevant features which ones are optimal. The question is also important since a lot of interesting features of earlier generations such as behavioral patterns or brain structure disappear almost completely. This is the point where adaptational stance comes in with the optimality principle.

P_7 **(optimality):** In the absence of evidence for the contrary it will be assumed that every relevant feature is optimal. This is nothing else than a method which is suggested as a replacement for coin tossing. Its power lies in the fact that in real-life cases optimal features are believed to be in majority. Another problem remains however: how do we decide which feature is relevant and which is not. As Dennett says, the sum of the number of eyes and the number of legs does not seem to be a relevant feature, but not much more is said about this issue. In Section 5.2.3 a detailed discussion of this question is given.

Dependency

As it has been shown the relevant and irrelevant features can be separated without referring to their functional role in the organism. So far the organism under investigation was handled as a *black box* i.e. we have not examined the causal relationship between the features. In other words *structure* has to be given to the set $\mathcal{G}$ in order to give an explanation and a complete description of the organism.

The structure of $\mathcal{G}$ will have the form of dependency relations. The concept of dependency has been mentioned already in Section 5.2.1. To discuss reverse engineering, this notion has to be made more precise. We will not give a formal definition but will try to make this term as clear as possible. If g_1 and g_2 are features than we say that g_2 depends on g_1 (and denote this relationship by $g_1 \rightarrow g_2$) if for any organism $o \in O$ it is possible to predict $g_2(o)$ from the value of $g_1(o)$ with some accuracy greater than zero. We will call this accuracy the *importance* or *weight* of the given dependency. Note that it is not necessarily possible to do such a prediction in the other direction; this dependency works like a (fuzzy) implication operator on predicates of the form "g is known".

In some cases more difficult relationships can be described. Computing a feature g may depend on values of more than one other features $g_1, \ldots, g_n$. Again, we define this relationship as a fuzzy formula on predicates like "g is known" using conjunction and implication and denote it by $g_1 \wedge \ldots \wedge g_n \rightarrow g$. If there are no other relationships of other kinds then we say that the database containing the dependencies is in Horn normal form or simply we have a Horn dependency database.

We are not interested in the details of the actual realization of the dependency of the features,

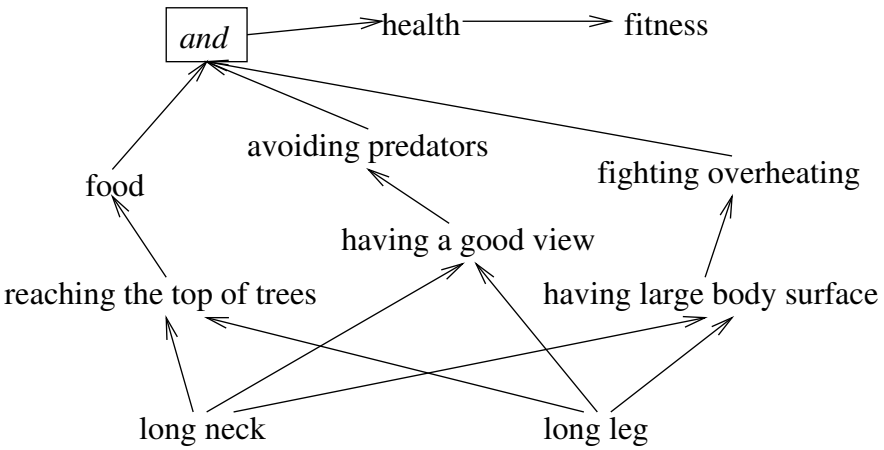


Figure 5.3: A small and ad hoc subset of the feature-dependency network of a giraffe for illustration only.

i.e. the underlying physical, biological or any kind of laws. Only the pure statistical fact of the relationship is important. We are not interested in the nature or the effective procedure of computing the features either; we only assume that such a procedure exists. At the abstraction level of our model, these are all unimportant.

It is enough to give the dependencies between the features to give an explanation of the fitness of the organism since mating ability itself is a feature, and the fitness can be directly derived from the mating ability of our organism and the competing members of the population (that are part of the environment) and some other factors of the environment like predators. Since we have accepted P_3, it can be said that fitness is a feature itself i.e. $f \in \mathcal{G}$. The functional role of a feature is nothing else but the way fitness depends on it. Irrelevant features have no functional role at all because fitness does not depend on them (this is the definition of irrelevance) though they can (and probably will) depend on other, maybe relevant, features.

An example of such a dependency network for the giraffe is shown in figure 5.3. The *network* structure of the dependencies must be emphasized. Taking a look at the diagram, it is hard to accept that for example the functional role of the long neck of the giraffe is to reach the leaves on the top of the trees. It is one of its functions but it may have several other roles as well. In biological organisms it is typical for the components to have several different functional roles; good examples are hormones, vitamins, muscles (motion and heat generation) and so on. Of course it is necessary to weight the dependencies according to importance. There are more important and less important roles but this does not change the fact that fitness depends on each role so they must be part of the explanation of the structure of the organism.

From a historical point of view it is possible that the effect of a less important feature results in a diminishing selective pressure related to a more important feature and can converge only *after* the more important feature has converged. But this historical perspective is completely irrelevant when discussing the dependency relations in the case of an organism. This argument about the independence of syncronic and diachronic analysis is not new: in connection with

the science of language Saussure (de Saussure, 1939) represents the same view which is now widely accepted.

Reverse Engineering

Roughly speaking, when trying to understand organisms via reverse engineering we try to reconstruct the dependency relations between the relevant features. The first difficulty is to find the relevant features. This problem will be discussed in detail later in many places from many directions. The second problem is that our dependency diagram is too simple; a classification of features along an additional direction needs to be introduced.

This dimension is approximately the level of abstraction. To understand this let us consider the typical way engineers solve a problem. In the first step they are given a functional description of the system to be designed. This includes its main purpose which is a high number of offspring in the case of biological organisms. This first specification is nothing else but requirements w.r.t. a set of high level, abstract features. The engineer then tries to reduce these features (or goals) into subgoals iteratively using more and more specific features while the design reaches a state where every subgoal can be implemented. This is the less abstract, physical state.

The actual procedure of finding a good design does not necessarily follow the above iteration; case based reasoning may play a major role for example but the result has the structure of features and the dependencies between the features. Some of the features represent low level physical properties some of them represent the highest level functional properties and there are features at intermediate levels of abstraction. Dennett distinguishes three levels: the physical, the organizational and the intentional levels (Dennett, 1990); we will need only the notion of physical features and the notion of top goal(s) which is essentially the first specification the engineer starts with. There are no clean boundaries, though: there is a practically continuous spectrum of abstraction along the implicational chain.

Now reverse engineering can be defined more clearly. If normal engineering proceeds from the top goals to the physical features then reverse engineering goes in the other direction: from the observable physical features it reconstructs the top and intermediate goals. There is one more kind of reverse engineering. In this case the to goals are also known, only the intermediate features have to be discovered. Good examples of the first kind are the mysterious tools of ancient cultures found by archaeologists. Here scientists have to find out the purpose of these tools and in most of the cases all they know is the actual physical properties of the objects.

An interesting question is that in the case of the problem of explaining physical features of animals such as long neck or big ears which kind of reverse engineering is involved. The top goal of different organisms is the subject of debate, for example what is the top goal of an ant? However in certain cases there are available top goals (which are actually subgoals in the strict sense) e.g. flying. Determining the dependency relationships between the physical features of birds and their observable flying ability is an example.

Optimality and dependency Dependency is a property of features as functions while optimality is a property of a particular feature *value*. However while reverse engineering an organ-

ism we usually have to rely only on particular feature values when we determine the dependency relations. The case when the optimality or irrelevance of features is known is the simpler. Here we ignore irrelevant features and try to guess (extrapolate) the optimal value of suboptimal features so we can rely on optimal relevant values. When trying to find the function of a physical feature we assume that this function is such that the physical feature is a good implementation of it. For example if we observe strong wings we assume that their function is flying. Note that without the optimality condition it would be possible that the wing is simply irrelevant and not used for anything or suboptimal and used for digging for example or even harmful (as the legs of snakes). When optimality is not known the optimality assumption P_7 has to be used (see Section 5.2.1) according to the adaptational stance.

5.2.2 Adaptation in EC

First let us take a look at the notions defined in Section 5.2.1. The easiest is P_2 which requires that the constraints of the possible organisms should be taken into account. In EC this is not a problem since one of the first steps of any application is the exact definition of the search space i.e. O. The particular methods of constraint handling and coding is not relevant here only the fact that the problem is handled properly.

P_1 and P_3 are trickier but still easier than in the case of biological evolution. In EC it is typical to have a population of solutions in every time step (generation) and usually an objective function is defined over the possible solutions. The expected number of offspring (i.e. fitness) of the members of the population is given by the objective function values of the other members of the population thereby fitness depends on the objective function *and* the actual population. Therefore we face the same problem of determining the boundaries of the organism and keeping the environment constant.

The later goal is easier since the environment outside the population is constant in most of the applications though nowadays the applications in dynamic environments are becoming more and more important. This means that the earlier goal reduces to deciding whether the individual or the population should be the organism to study. The later choice seems more reasonable though the arguments in connection with P_6 given in Section 5.2.1 apply here as well. Anyway, the usefulness of having large populations is not proven; there are cases where one-element populations perform best. A typical example is (Eiben et al., 1998b).

The notion of the fitness function f is also cleaner in EC. It is very interesting that in EC there is a tradition of calling the objective function the fitness function. At first sight this results in a dissonance between biological and computational terminology but in the light of the restrictions expressed by P_3 we saw that even in biology fitness is taken as a kind of objective function; adaptationists emphasize the objective nature of fitness. In EC to be an adaptationist all we have to declare that the good old tradition should be continued.

$\mathcal{G}$ also has traditions in EC especially in genetic algorithms (GAs). In GAs the solutions need to be encoded so that genetic operators which are defined problem independently could be applied to them. This encoding is analogous to the DNA sequence in which mutation is very similar in the case of every living organism. The views expressed in this question usually take the form of problems of encoding which is essentially nothing else but defining some

atomic elements of $\mathcal{G}$ (see e.g. (Radcliffe, 1994)). These are the physical features as introduced in Section 5.2.1. The other features called building blocks are some simple combinations of these primitives (Goldberg, 1989). Though these building blocks can be regarded as functional properties this approach has a number of well known difficulties and limitations as will be discussed in Section 5.2.3.

Principle P_4 which requires that only a single niche should be studied is maybe the most problematic. There is a significant amount of work in the field of niche and species formation in the field of EC, Chapter 2 and (Deb and Goldberg, 1989) are two examples. The common problem is that all these methods operate with a distance measure defined over O and this distance measure typically depends on the atomic elements of $\mathcal{G}$. This makes the whole procedure ad hoc in the sense that the actual encoding of a given problem is not necessarily optimal. In fact it is always possible to find a distance measure such that fitness has only one optimum i.e. it is unimodal. For example the difference between the fitnesses is such a distance measure. This makes $\mathcal{G}$ even more interesting.

Finally P_5 and P_6 has to be mentioned. P_5 usually holds in EC if the applied selection mechanism is elitist which means that the best member of every generation will be the member of the next generation. The elitist strategy usually performs quite well so it is typically applied. P_6 can be interpreted similarly as in the case of biological evolution.

It should be clear by now that there are no basic incompatibilities between EC and the adaptationist stance. All that remains is to take a look at the possible applications of the *methods* of the adaptationist stance such as optimality analysis and reverse engineering.

Characterization of Features

In this area EC has a major advantage namely that it is possible to perform virtually any number of experiments and thereby collecting as many "fossils" as necessary. It is possible to predict with a much greater degree of confidence if a feature is optimal or not by performing statistical experiments since for optimal features convergence to the same value should occur in the majority of the experiments. Of course premature convergence and other well known effects can alter the results. The most difficult problem is to ensure that the algorithm should stay in the same niche every time the experiment is run.

There is another advantage: a hypothesis about the characterization of a feature can be tested experimentally. For example if a feature is suggested to be irrelevant it is easier to vary its value while leaving the other features unchanged and calculate the fitness of the resulting solutions.

Reverse Engineering

The dependency relations between features have the same status as their optimality since they were also defined in statistical terms. Any number of experiments can be performed to test and refine the hypothesis. In spite of the fact that there are much greater possibilities in testing the dependency relations between any kind of features (which are the result of reverse engineering) there is very little work in the literature that would describe such reverse engineering results. The reason probably is that the success of reverse engineering which usually results in a deeper

understanding of the engineering problem at hand and so also faster and better algorithms than plain EC algorithms is usually considered a *failure* of EC for some reason. People seem to forget that the way the faster and better algorithm are developed is largely dependent on the performance of EC algorithms; in fact it is impossible without them.

In this section it is attempted to show that it may be useful to at least try to understand the outcome of the optimization process, the solutions suggested by EC algorithms since these solutions form a good starting point to a reverse engineering process that makes it possible to translate the information accumulated by evolution into engineering knowledge. If this process results in better heuristics then *it is the success of EC* and the engineers of course.

A simple example of the successful applications of reverse engineering was presented in Section 4.2. There we developed a heuristic for solving subset sum problem instances that were generated in a special way. This heuristic outperformed all other methods we tried including the GA that originally lead us to the heuristic. There are more sophisticated examples as well such as network design (Cox et al., 1996).

5.2.3 Representational Bottleneck

The language of biology is particularly rich and in a way it is very close to natural language. The terms (features) used to describe organisms such as color, shape, organs and body-parts like heart, lungs and arms and behavioral patterns like aggression are typically understandable by anyone. Biology inherited a large, detailed terminology of describing the living world. This may be a result of our own evolution and the evolution of our culture; animals and plants have been around us since language and culture emerged and have been playing a crucial role in our lives as food, enemy, building material and so on ever since. Our sensory and cognitive system is likely to be specialized in describing living organisms, among other things.

This makes the job of finding problems in biology relatively easy since the features to be explained *are there*. The situation is radically different in EC. The description of solutions lacks even the simplest terms and usually reduced to the encoding of the solution. This means that we talk only about *genes* as if biologists could describe a monkey only with the help of its DNA sequence. The insufficiency of this description may be evident to some of the readers but actually in EC the practice is accepting that the encoding provides us with a language sufficient for describing the solutions. This is motivated by the need of developing a domain independent theory of EC, and this need may be originated from the analogy with other optimization methods like the method of steepest descent or the different Newtonian iterations. The problem is that the application area of EC is *much* larger than any of these restricted methods. This is why domain specific information plays a more important role.

The elephant picture function When it comes to reverse engineering and giving an explanation of the solutions developed we need to find features to create a model of the particular problem class. To see this let us give an example: the elephant picture function. In this problem the organisms are two-dimensional bitmaps where the physical features are the pixels of the picture. The fitness of a given picture is the degree of resemblance to an elephant; any kind of elephant in any position as illustrated by Figure 5.4.

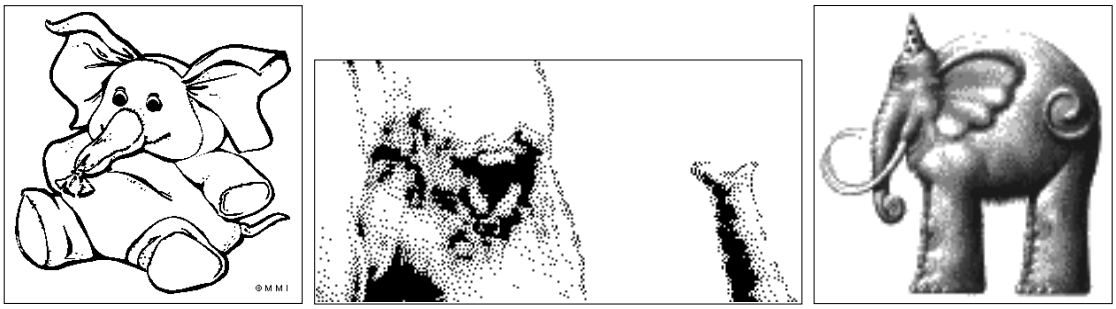


Figure 5.4: Examples of bitmaps with a good fitness.

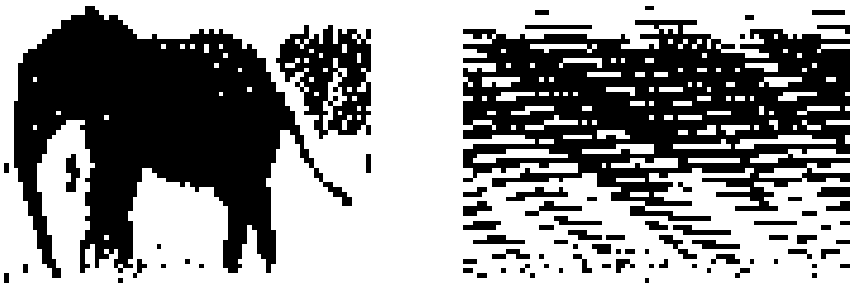


Figure 5.5: An elephant picture and one of its permutations.

The fitness is independent from a particular pixel since the degree of resemblance of a negative picture is the same as of the original. The elephant picture function does depend on lower-level features such as lines, curves and similar basic components but these are still too abstract to depend on particular pixels; we need another abstraction level closer to the physical features. To convince those who are still in doubt imagine that the positions of the pixels of the picture are mixed by a *deterministic* and *invertible* algorithm so the elephant cannot be recognized anymore by looking at the picture (see Figure 5.5). Let the fitness function be the original elephant picture function applied to the result of the inverse of the above transformation on the given bitmap. The most predictive high-level feature is still resemblance to elephants *before* mixing the pixels. If one cannot "decode" the image then it is almost impossible to predict fitness.

On the other hand an EC algorithm could probably find a good solution of the elephant picture problem provided that somehow this function was available. Interactive applications i.e. applications that use the human user as a fitness function indeed exist; one example is the iterative evolution of textures where the fitness is a kind of artistic or aesthetic value (Ibrahim, 1998).

Finally let us note that the features used in biology to describe organisms are practically independent of the level of DNA as well since it would be extremely hard to predict the shape or behavior of an *unknown* animal from its DNA only. Genes connected to higher level features

are often described and known species with similar DNA can be used but this process proceeds in a top-down fashion: the features are described first and then the corresponding genes are looked for.

Similar problems Self-organizing maps (SOMs) are devices that are capable of finding structure or a clustering in their input data-set. One good example is Kohonen's phoneme recognizer (Kohonen, 1988). SOMs are usually based on the physical features of these vectors only so in many cases it has serious problems finding clusters that are similar in some important sense but are not in the physical level (e.g. elephant pictures). In the case of instance based learning and case based reasoning (see e.g. (Mitchell, 1997)) it is well known that one of the main problems is the selection of the distance measure. To find a good distance measure, one has to *understand* the problem domain to be able to tell the difference between important and irrelevant features or at least to find features to start with. Just like in SOMs for the interesting problems it is not sufficient to rely only on the physical, lowest level features like pixels in an image or elements of a vector as usually done.

The spaceship picture function To illustrate that the elephant picture function is indeed a very serious problem let us give another example: the spaceship picture function. The situation is like with the elephant pictures but the difference is that e.g. 500 years ago nobody had an idea about spaceships. The interesting thing is that the function did exist nevertheless though no one could predict its values. Engineers of that time would have been in great trouble when trying to understand a spaceship function computing machine. Note that no low-level "decoding" would have helped them in this case as we have seen in Section 5.2.3. The problem is that spaceships can be very different and some of them are very similar to aircrafts, saucers etc.

What the engineers lacked 500 years ago is the knowledge about spaceships, space, artificial flying, science fiction movies etc. To be short the term spaceship was not part of their language. I assume that there are more unknown than known terms: this is what I call the representational bottleneck. The situation in the abstract domains such as flow control or combinatorial problems that are likely to have a rich and complex yet undiscovered structure is even worse.

5.2.4 Conclusions

It was demonstrated that the adaptational stance and reverse engineering is strongly connected. Adaptational stance is a strategy of reverse engineering that might fail in certain kinds of circumstances but it seems to be useful in many cases. It was shown that reverse engineering is applicable and in fact it has been applied in EC. The bottleneck of this method however is the language that is available for describing the abstract problem domains. As we have seen, biology has a much larger and effective vocabulary partly inherited from natural language though this vocabulary is constantly growing as other sciences develop so adaptationist explanations in biology also have the representational bottleneck.

It is also important to emphasize that the performance of an EC algorithm may well be very good even if reverse engineering is not successful and domain specific knowledge is not available. This is the main power of evolution: it only works with the physical features and

does not care about any explanations because explanations or dependency models of features of different abstraction levels are simply tools for us human beings to handle predictions with our limited cognitive capacity. It is very much like the relationship between axioms and theorems in mathematics. Axioms have the status of physical features; knowing the axioms means knowing everything[4]. Theorems are needed only because it exceeds our abilities to tell the truth value of a formula directly from the axioms. This is why the representational bottleneck is a problem though the physical features (encoding of a solution in EC) are known; explanations need higher level features because of our cognitive limits.

While EC algorithms often provide good solutions without much domain knowledge they can be exploited also as a ladder leading to an understanding of the problem domain making reverse engineering possible and thus leading to faster and better specific algorithms. The representational bottleneck problem should be solved by scientific research in each domain. Since experiments can be repeated any time with any settings this research is much easier than in biology though the vocabulary to start with may be poor. The contribution of EC to the new algorithms is essential since without known good solutions reverse engineering is impossible therefore the success of such results is definitely a success of EC as well.

[4]It is not always true because of Gödel's theorem but practically this is the case in traditional maths. Theorems that may be true or false in different models are very hard to find.

Summary

Here the main points of the thesis are briefly summarized touching on the open problems and possible future research directions.

GAS is a GA with niching via maintaining a subpopulation structure. The motivation of developing GAS was that other known niching techniques like fitness sharing cannot handle situations where the distribution of the local optima are not even, in other words they do not solve the *niche radius problem*. GAS solves the niche radius problem via maintaining not only a set of subpopulations but also a recursive tree structure which allows subpopulations to adapt to certain regions in the search space while avoiding the low quality regions. The exploration is concentrated recursively in subspaces occupied by subpopulations and it is more and more focused as the search proceeds. Thus we achieve a sort of cooling effect which is very similar to the technique applied in simulated annealing.

The bias of GAS follows from the above motivation: if the search problem at hand contains a huge number of local optima of similar quality that are evenly spread then the application of the algorithm is not justified. On the other hand, the performance of the algorithm depends heavily on the distance function that is used to determine the difference between solutions. An advantage is however that the general framework does not make any assumptions about the distance function so it can be replaced without a problem.

UEGO is a generalization of GAS. The motivation for developing UEGO was mainly to convert the ideas behind GAS into a true general paradigm in which any global optimizer can be plugged in. The structure was simplified and modifications were made to make parallel implementation easier. Extensive empirical tests have also been performed that allowed us to understand the parameters better. An open question that is related to both algorithms is to sufficiently characterize their application domain. In this thesis ideas are presented but obviously more empirical and theoretical work is necessary to pin down the key characteristics.

In the second part of the thesis models of the genetic search are presented. These include a technique that is useful for measuring problem difficulty and it also provides a way of visualization of the structure of the space. The advantage of the method is that it is based on transition probabilities between solutions that are calculated using the actual operators. The disadvantage of the method is that it is computationally more expensive than e.g. fitness distance correlation.

Problems with the schema-based approaches are also tackled. The motivation behind introducing implicit formae is that models operating only with sets defined using the applied encoding overlook the fact that what actually counts is only the disruptive effects of the algorithm components like operators and selection. Therefore in a model of this type it is necessary to consider predicates over the space which are not defined by the encoding but which are not dis-

rupted by the algorithm even if the operators were originally designed to respect the encoding. Several examples are given for such implicit formae for different problems. The wave model extends these ideas. The predicates it operates with are decomposed further so it is possible that some predicate is important only at the beginning of the search while another only at the end. The search is modeled as a wave that spreads via paths formed of such predicates.

An open problem related to this research is that since the predicates we use are not related to the encoding we must face the problem of finding them. Using only predicates defined easily by the encoding (e.g. schemata) is comfortable and easy but insufficient. Using more general predicates is sufficient but very hard in general. In fact finding such predicates for our models is a serious research task by itself. Furthermore in abstract domains it is not clear whether we can hope to find useful predicates. One way of solving this problem is to do it automatically. A method for automatic knowledge extraction is also presented in the form of an automatic encoding generation framework that can apply any appropriate machine learning algorithm.

This later framework raises many interesting questions. The main problem is its implementation. To implement it one has to chose a machine learning algorithm. But every machine learning algorithm has a bias that automatically becomes the bias of the code generation. Thus if the domain at hand is not suitable for the algorithm we use then no good result can be expected. To find the appropriate learning algorithm experimenting is necessary. The consequence is that we only push this uncertainty one level higher. I think that this problem is not the problem of our approach. it is an inherent limit of every kind of learning: every learning algorithm has its bias except random search. So the highest level of problem solving must always be random search.

This highest level is not the engineer's or the scientist's mind however. It is the product of evolution and culture so it is full of different biases. Of course culture is a result of a sort of evolution. The highest level is not even evolution itself since it is clearly not random search: every solution is a modification of an already existing one. This question would deserve a separate book so let us leave it open for a while. . .

The last section of the thesis elaborates on a part of the above mentioned philosophical problem, namely the possibility of creating a vocabulary for describing evolutionary algorithms. It also draws a parallel between *adaptational stance* which is a way of thinking about evolution and EC.

Bibliography

Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*. Wiley and Sons.

Altenberg, L. (1994). Evolving better representations trough selective genome growth. In *First IEEE conference on Evolutionary Computation*, pages 182–187, Piscataway, NJ. IEEE.

Altenberg, L. (1997a). Fitness distance correlation analysis: an instructive counterexample. In (Bäck, 1997), pages 57–64.

Altenberg, L. (1997b). NK landscapes. In (Bäck et al., 1997), pages B2.7:5–10.

Bäck, T. (1993). Optimal mutation rates in genetic search. In (Forrest, 1993), pages 2–9.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford University Press, New York.

Bäck, T., editor (1997). *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, California. Morgan Kaufmann.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York.

Bäck, T., Hoffmeister, F., and Schwefel, H.-P. (1991). A survey of evolution strategies. In (Belew and Booker, 1991).

Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors (1999). *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA. Morgan Kaufmann.

Battle, D. L. and Vose, M. D. (1993). Isomorphisms of genetic algorithms. *Artificial Intelligence*, 60(1):155–165.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125.

Belew, R. K. and Booker, L. B., editors (1991). *The Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann.

Bertoni, A. and Dorigo, M. (1993). Implicit parallelism in genetic algorithms. *Artificial Intelligence*, 61:307–314.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical report, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich.

Boettcher, S. and Percus, A. (2000). Nature's way of optimizing. *Artificial Intelligence*, 119:275–286.

Caruana, R. A. and Schaffer, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In Laird, J., editor, *Proceedings of the fifth international conference on machine learning*, pages 153–161.

Cox, Jr, L. A., Davis, L., Lu, L. L., Orvosh, D., Sun, X., and Sirovica, D. (1996). Reducing the costs of backhaul networks for PCS companies using genetic algorithms. *Journal of Heuristics*, 2(1):1–16.

Csányi, V. (1982). *General Theory of Evolution*. Publ. House of the Hung. Acad. Sci. Budapest.

Davidor, Y. (1991a). Epistatsis variance: A viewpoint on GA-hardness. In (Rawlins, 1991), pages 23–35.

Davidor, Y. (1991b). A naturally occuring niche and species phenomeon: the model and first results. In (Belew and Booker, 1991).

Davidor, Y., Schwefel, H.-P., and Männer, R., editors (1994). *Parallel Problem Solving from Nature 3*, volume 866 of *Lecture Notes in Computational Science*. Springer-Verlag.

Dawkins, R. (1989). *The Selfish Gene*. Oxford University Press, Oxford, 2nd edition.

De Jong, K. A. (1993). Genetic algorithms are NOT function optimizers. In Whitley, L. D., editor, *Foundations of Genetic Algorithms II*, pages 5–18. Morgan Kaufmann.

de Saussure, F. (1939). *Cours de linguistique générale*. Payot, Paris.

Deb, K. (1989). Genetic algorithms in multimodal function optimization. TCGA report no. 89002, The University of Alabama, Dept. of Engineering mechanics.

Deb, K. and Goldberg, D. E. (1989). An investegation of niche and species formation in genetic function optimization. In (Schaffer, 1989).

Dennett, D. C. (1990). *The Intentional Stance*. MIT Press.

Eiben, A. E., Aarts, E. H. L., van Hee, K. M., and Nuijten, W. P. M. (1995). A unifying approach to heuristic search. *Annals of Operations Research*, 55:81–99.

Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors (1998a). *Parallel Problem Solving from Nature - PPSN V*, volume 1498 of *Lecture Notes in Computational Science*. Springer-Verlag.

Eiben, A. E., Raué, P.-E., and Ruttkay, Zs. (1994). Genetic algorithms with multi-parent recombination. In (Davidor et al., 1994), pages 78–87.

Eiben, A. E., van der Hauw, J. K., and van Hemert, J. I. (1998b). Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46.

Eshelman, L. J. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In (Rawlins, 1991).

Eshelman, L. J., editor (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann.

Eshelman, L. J., Caruana, R. A., and Schaffer, J. D. (1989). Biases in the crossover landscape. In (Schaffer, 1989).

Falkenauer, E. (1994a). New representation and operators for genetic algorithms applied for grouping problems. *Evolutionary Computation*, 2(2):123–144.

Falkenauer, E. (1994b). Setting new limits in bin packing with a grouping GA using reduction. Technical Report RO108, Research Centre for Belgian Metalworking Industry, Department of Industrial Automation.

Fogel, D. B., editor (1998). *Evolutionary Computation: the Fossile Record*. IEEE Press.

Fogel, L. J., Owens, A. J., and Whals, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley.

Forrest, S., editor (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann.

Glover, F. W. (1996). Tabu search and adaptive memory programming — advances, applications, and challenges. In Barr, R. S., Helgason, R. V., and Kennington, J. L., editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Kluwer Academic Publishers, Norwell, MA.

Glover, F. W. and Laguna, M. (1998). *Tabu Search*. Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Goldberg, D. E. (1991). A comparative analisys of selection schemes used in genetic algorithms. In (Rawlins, 1991).

Gould, S. J. and Lewontin, R. (1979). The spandrels of San Marco and the Panglossian paradigm: A critique of the adaptationist programme. In *Proceedings of the Royal Society*, volume B205, pages 581–598.

Grefenstette, J. J. (1984). Genesis: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165.

Grefenstette, J. J. (1991). Conditions for implicit parallelism. In (Rawlins, 1991).

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

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42.

Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. In (Davidor et al., 1994), pages 149–158.

Ibrahim, A. (1998). *GenShade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation*. PhD thesis, Texas A&M University.

IEE/IEEE (1995). *The proceedings of Genetic Agorithms in Engeneering Systems: Innovations and Applications*, Sheffield, UK. IEE/IEEE.

Ishibuchi, H., Murata, T., and Tomioka, S. (1997). Effectiveness of genetic local search algorithms. In (Bäck, 1997), pages 505–512.

Jelasity, M. (1997). A wave analysis of the subset sum problem. In (Bäck, 1997), pages 89–96.

Jelasity, M. (1999). The adaptationist stance and evolutionary computation. In (Banzhaf et al., 1999), pages 1859–1864.

Jelasity, M. (2000). Towards automatic domain knowledge extraction for evolutionary heuristics. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VI*, volume 1917 of *Lecture Notes in Computational Science*, pages 755–764. Springer-Verlag.

Jelasity, M. and Dombi, J. (1996). Implicit formae in genetic algorithms. In Ebeling, W., Rechenberg, I., Schwefel, H.-P., and Voigt, H.-M., editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computational Science*, pages 154–163. Springer-Verlag.

Jelasity, M. and Dombi, J. (1998). GAS, a concept on modeling species in genetic algorithms. *Artificial Intelligence*, 99(1):1–19.

Jelasity, M., Ortigosa, P. M., and García, I. (2001). UEGO, an abstract clustering technique for multimodal global optimization. *Journal of Heuristics*, 7(3).

Jelasity, M., Tóth, B., and Vinkó, T. (1999). Characterizations of trajectory structure of fitness landscapes based on pairwise transition probabilities of solutions. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, pages 623–630. IEEE, IEEE Press.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In (Eshelman, 1995), pages 184–192.

Juels, A. and Wattenberg, M. (1996). Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 430–436. The MIT Press.

Juliany, J. J. and Vose, M. D. (1994). The genetic algorithm fractal. *Evolutionary Computation*, 2(2):165–180.

Kallel, L. (1998). Inside GA dynamics: Ground basis for comparison. In (Eiben et al., 1998a), pages 57–66.

Kargupta, H. (1995). Signal-to-noise, croostalk and long range difficulty in genetic algorithms. In (Eshelman, 1995).

Kargupta, H., Deb, K., and Goldberg, D. E. (1992). Ordering genetic algorithms and deception. In (Männer and Manderick, 1992).

Kauffman, S. A. and Levin, S. (1987). Towards a general theory of adaptive walks on rugged fitness landscapes. *Journal of Theoretical Biology*, 128:11–45.

Khuri, S. (1994). Walsh and haar functions in genetic algorithms. In *Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94)*, Phoenix, AZ. ACM Press.

Khuri, S., Bäck, T., and Heitkötter, J. (1993). An evolutionary approach to combinatorial optimization problems. In *The Proceedings of CSC'94*.

Kingdon, J. and Dekker, L. (1995). The shape of space. In (IEE/IEEE, 1995).

Kohonen, T. (1988). The "neural" phonetic typewriter. *IEEE Computer*, pages 11–22.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Liepins, G. and Vose, M. D. (1991a). Deceptiveness and genetic algorithm dynamics. In (Rawlins, 1991), pages 36–50.

Liepins, G. E. and Vose, M. D. (1991b). Deceptiveness and genetic algorithm dynamics. In (Rawlins, 1991).

Lucas, S. M. (1995). Towards the open ended evolution of neural networks. In (IEE/IEEE, 1995).

MacWilliams, F. J. and Sloan, N. J. A. (1977). *The Theory of Error-correcting Codes*. North Holland.

Manderick, B. (1997). Correlation analysis. In (Bäck et al., 1997), pages B2.7:10–14.

Manderick, B., de Weger, M., and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. In (Belew and Booker, 1991), pages 143–150.

Mandischer, M. (1993). Representation and evolution of neural networks. In Albrecht, R. F., Reeves, C. R., and Steele, N., editors, *Artificial Neural Nets and Genetic Algorithm Proceedings of the International Conference at Innsbruck*, pages 643–649, Austria. Springer.

Männer, R. and Manderick, B., editors (1992). *Parallel Problem Solving from Nature 2*. Elsevier Science Publishers/North Holland (Amsterdam).

Maynard Smith, J. (1982). *Evolution and the Theory of Games*. Cambridge University Press, Cambridge.

McEliece, R. J. (1977). *The Theory of Information and Coding*. Addison-Wesley.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition.

Mitchell, M. (1996). *An Intruduction to Genetic Algorithms*. MIT Press.

Mitchell, M., Holland, J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hillclimbing? In Cowan, J. D. et al., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Munetomo, M. and Goldberg, D. E. (1999). Identifying linkage groups by nonlinearity/non-monotonicity detection. In (Banzhaf et al., 1999), pages 433–440.

Naudts, B. and Kallel, L. (1998). Comparison of summary statistics of fitness landscapes. Submitted for publication.

Newell, A., Shaw, J. C., and Simon, H. A. (1960). A variety of intelligent learning in a general problem solver. In Yovits, M. C. and Cameron, S., editors, *Self-organizing systems: Proceedins of an interdisciplinary conference*, pages 153–189, New York, NY. Pergamon Press.

Ortigosa, P. M. (1999). *Métodos estocásticos de Optimización Global. Procesamiento paralelo*. PhD thesis, Department of Computer Architecture and Electronics, University of Almería, Almería, Spain. Available as http://dali.ualm.es/papers/99/tesispilar.ps.gz.

Ortigosa, P. M., García, I., and Jelasity, M. (2001). Two approaches for parallelizing the UEGO algorithm. Accepted for publication in the Volume bearing the title of Mátraháza Optimization Days.

Pelikan, M., Goldberg, D. E., and Lobo, F. (1999). A survey of optimization by building and using probablistic models. Technical Report 99018, Illinois Genetic Algorithms Laboratory.

Quick, R. J., Rayward-Schmith, V. J., and Smith, G. D. (1998). Fitness distance correlation and ridge functions. In (Eiben et al., 1998a), pages 77–86.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Radcliffe, N. J. (1991a). Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205.

Radcliffe, N. J. (1991b). Genetic set recombination and its applications to neural network topology optimization. Technical Report EPCC-TR-91-21, Edinburgh Parallel Computing Centre.

Radcliffe, N. J. (1992). Non-linear genetic representations. In (Männer and Manderick, 1992).

Radcliffe, N. J. (1994). The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384.

Radcliffe, N. J. (1997). Schema processing. In (Bäck et al., 1997), pages B2.5:1–10.

Radcliffe, N. J. and George, F. A. W. (1993). A study in set recombination. In (Forrest, 1993).

Radcliffe, N. J. and Surry, P. D. (1995). Fitness variance of formae and performance prediction. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms III*, pages 51–72. Morgan Kaufmann.

Rawlins, G. J. E., editor (1991). *Foundations of Genetic Algorithms*. Morgan Kaufmann.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog, Stuttgart.

Rudnicky, A. I., Hauptmann, A. G., and Lee, K. (1994). Survey of current speech technology. *Communications of the ACM*, 37(3):52–57.

Rudolph, G. (1997). *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač, Hamburg.

Schaffer, J. D., editor (1989). *The Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Wiley.

Sokal, R. R. and Rohlf, F. J. (1981). *Biometry*. W. H. Freeman and company, New York.

Solis, F. J. and Wets, R. J.-B. (1981). Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30.

Suzuki, J. (1993). A Markov chain analysis on a genetic algorithm. In (Forrest, 1993), pages 146–153.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In (Schaffer, 1989).

Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithms. In (Rawlins, 1991), pages 94–101.

Tang, K. S., Chan, C. Y., Man, K. F., and Kwong, S. (1995). Genetic structure for NN topology and weights optimization. In (IEE/IEEE, 1995).

Thierens, D. (1998). Non-redundant genetic coding of neural networks. Technical Report UU-CS-1998-46, Utrecht University, Department of Computer Science.

Törn, A. and Žilinskas, A. (1989). *Global Optimization*, volume 350 of *Lecture Notes in Computational Science*. Springer-Verlag.

Tóth, G. J., Kovács, S., and Lőrinc, A. (1995). Genetic algorithm with alphabet optimization. *Biological Cybernetics*, 73:61–68.

van Lint, J. H. (1992). *Introduction to Coding Theory*. Springer-Verlag.

Venturini, G. (1995). GA consistently deceptive functions are not challenging problems. In (IEE/IEEE, 1995).

Vose, M. D. (1991). Generalizing the notion of schemata in genetic algorithms. *Artificial Intelligence*, 50(3):385–396.

Vose, M. D. (1992). Models of genetic algorithms. Technical Report CS-92-148, University of Tennessee, Knoxville, TN.

Vose, M. D. (1999). *The simple genetic algorithm: foundations and theory*. MIT Press.

White, M. S. and Flockton, S. J. (1995). Modelling the behaviour of the genetic algorithm. In (IEE/IEEE, 1995).

Whitely, L. D. (1991). Fundamental principles of deception in genetic algorithms. In (Rawlins, 1991), pages 221–241.

Wilson, D. S. and Sober, E. (1994). Reintroducing group selection to the human behavioral sciences. *Behavioral and Brain Sciences*, 17(4):585–654.

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

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In (Rawlins, 1991), pages 205–218.

Yagiura, M. and Ibaraki, T. (1996). Genetic and local search algorithms as robust and simple optimization tools. In Osman, I. H. and Kelly, J. P., editors, *Meta-Heuristics: Theory and Application*, pages 63–82. Kluwer Academic Publishers.

Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4):539–567.

Samenvatting

Het centrale thema van dit proefschrift is de structuur van de zoekruimtes van globale optimalisatieproblemen. Er zijn veel vragen die op een antwoord wachten op dit gebied, bij voorbeeld, hoe deze ruimtes te karakteriseren? Hoe ze te beschrijven, hoe ze te classificeren? Hoe kan men het mogelijk maken dat een algoritme zich aanpast bij een speciale ruimte structuur? Ik ben vooral geïnteresseerd in die ruimtes die geen voor de hand liggende structuur hebben die in gangbare termen, zoals bijvoorbeeld discreet, stochastisch, of niet differentiabel, te karakteriseren valt. Zulke problemen vormen het voornaamste applicatiegebied van menig (meta)heuristiek, bijvoorbeeld evolutionaire rekentechnieken, simulated annealing, tabu-search, ant colony optimization, enzovoort. Combinatorische problemen zijn voorbelden bij uitstek: beschikbare effectieve algoritmen gebruiken heel weinig informatie over de zoekruimte, er uithalen van deze structuur kan dus een manier zijn om aanzienlijke verbeteringen te bereiken.

Het eerste hoofdstuk bevat een introductie tot de basisbegrippen in dit proefschrift. De rest van het proefschrift wordt in vier verdere hoofdstukken ingedeeld. Hofdstuk 2 (Jelasity and Dombi, 1998) en 3 (Jelasity et al., 2001) beschrijfen globale optimalisatietechnieken die de structuur van de verzameling van locale optima verkennen. De eerste methode is GAS genaamd. Dit algoritme is in staat om structurele informatie op een adaptieve manier te benutten, door een subpopulatie-benadering te gebruiken die het vormen van soorten simuleert. Dit betekent dat de zoekruimte is opgedeeld in clusters (die we soorten noemen) en deze clusterstructuur wordt voortdurend aangepast volgens de resultaten van globaal zoeken beperkt tot deze clusters. De cluster-structuur weerspiegelt de verdeling van de locale optima. De bias van het algoritme kan expliciet worden gecontroleerd door een willekeurige afstandsfunctie over de ruimte te gebruiken. Het automatisch instellen van sommige belangrijke parameters is mogelijk op basis van deze afstandfunctie en andere door de gebruiker in te stellen parameters. De uitvoer van het algoritme bevat structurele informatie en de eindoplossing. De tweede methode is genaamd UEGO. Het verschil met GAS is dat de optimalisatie techniek die binnen een soort werkt kan willekeurig zijn. Dit maakt de hele aanpak tot een algemeen paradigma dat in combinatie met elke willekeurige optimalisatie techniek gebruikt kan worden als de structuur van de verzameling van locale optima belangrijk is, of als er stabiele oplossingen nodig zijn. Een ander verschil is dat de structuur van het algoritme vereenvoudigd is om parallelle implementaties te vergemakkelijken. Voor resultaten van parallelle implementaties zie (Ortigosa et al., 2001).

De secties binnen Hoofdstuk 4 zijn gerelateerd aan de manier waarop de werking van evolutionaire rekentechnieken beschreven is. Sectie 4.1 (Jelasity and Dombi, 1996) toont aan dat het gebruiken van een beperkt woordenboek – zoals gedaan bij sommige traditionele theoretische aanpakken – is onvoldoende en stelt een manier voor om het raamwerk uit te breiden. Dit

werk is nauw verbonden aan de epistemologische problemen van de computerwetenschappen laat zien dat theoretische studies die gebaseerd zijn op een vooraf vastgestelde collectie van eigenschappen van de zoekruimte tot falen gedoemd zijn. Sectie 4.2 (Jelasity, 1997) presenteert een alternatief model met nadruk op onbekende doch mogelijk belangrijke informatie over de structuur van de zoekruimte. Deze aanpak is gebaseerd op de voorafgaande sectie. Het verschil is dat alle aspecten van de oude manier van problemen bekijken verworpen worden en het wordt voorgesteld om het zoekproces te beschrijven als een pad door een ruimte van eigenschappen die relevant zijn voor de zoekruimte in kwestie. Sectie 4.3 (Jelasity et al., 1999) stelt een manier voor om de structuur van zoekruimtes te visualiseren en geeft tevens een maat voor de moeilijkheidsgraad van problemen. In dit geval is de visualisatie gebaseerd op de evaluatie functie *en* het gebruikte algoritme, dus het is een middel om vanuit het perspectief van het zoekalgoritme naar de door de evaluatie functie gedefinieerde structuur te kijken.

Hoofdstuk 5 introduceert een filosofisch probleem omtrent de grenzen van menselijk ontwerp. In sectie 5.2 (Jelasity, 1999) is het aangetoond dat de manier waarop binnen het gebied van evolutionaire rekentechnieken naar evolutie gekeken wordt, veel lijkt op de zogenaamde *adaptational stance*. Het wordt ook gesteld dat naast de overeenkomsten er significante verschillen zijn, en deze verband houden met de epistemologische crisis van computerwetenschappen. Waar het om gaat is, dat terwijl biologen een natuurlijk woordenschat hebben om over eigenschappen van dingen te praten (bijvoorbeeld, vleugel, oog, hart, enz.), de situatie in de computerwetenschappen is radicaal anders. Om dit te verduidelijken, stel eens voor om over een olifant te praten door alleen naar zijn DNA sequentie te kunnen refereren. In evolutionaire rekentechnieken heeft men geen andere informatie (terminologie) dan wat de meest elementaire technische taal toelaat. Het wordt ook aangetoond dat dit een diepliggend probleem is en er geen reden is om te veronderstellen dat het ooit door mensen opgelost zou worden. Menselijke intelligentie evolueerde in een specifieke omgeving voor specifieke taken. Hoewel het ontstaan van taal het mogelijk maakte om hoge abstractie niveau's te bereiken, het goed beschrijven van zulke abstracte ruimtes en het benutten van hun structuur is wellicht alleen mogelijk door kunstmatige intelligentie en machine learning technieken die in staat zijn om voor mensen totaal onnatuurlijke concepten te ontwikkelen en te gebruiken. Sectie 5.1 (Jelasity, 2000) oppert een manier om dit proces te automatisch te laten verlopen. Er wordt een raamwerk gepresenteerd waarin binaire coderingen van een probleem domein geleerd kunnen worden door ideeën uit machine learning te gebruiken. De werkelijke leerprocedure kan willekeurig zijn. Door deze aanpak wordt het misschien mogelijk om het reverse engineering probleem in de toekomst op te lossen op een "natuurlijke" manier. De evolutie van taal resulteerde in een vocabulaire dat voor mensen belangrijke dingen omvat. In abstracte domeinen wordt een soortgelijke ontwikkeling steeds meer mogelijk.

Curriculum Vitae

Márk Jelasity was born in Székesfehérvár, Hungary, in 1973 April 6th. He finished secondary school in 1991 in Székesfehérvár. Afterwards he went to the József Attila University of Szeged to study mathematics and computer science. He received his masters degree in 1996. During the university years he started to work in evolutionary computation under the direction of József Dombi. In 1997 he won the 1st Price at the National Scientific Conference of Students, and he got the Pro Scientia Medal of the Hungarian Scientific Association of Students. In 1996 he started studying general linguistics at the József Attila University of Szeged, and in 1998 he became a phd student in the cognitive science program of Eötvös Loránd University, Budapest. In the meantime he worked in the Research Group of Artificial Intelligence in Szeged when he was developing a speech recognition system. In 2000 he moved to the Netherlands to Leiden University, where he finished this thesis.