Optimization Algorithms Imitating Nature

Frank Kursawe

Department of Computer Science
Chair of Systems Analysis
P.O. Box 50 05 00
4600 Dortmund 50, Germany

UNI DO

Contents

Abstract

This survey introduces optimization algorithms imitating certain principles of nature in order to find the global optimum of a given problem. Especially those domains are worth copying where nature has found 'stable islands' in the 'turbulent ocean' of possibilities, like annealing processes, central nerve systems and biological evolution. These approaches start to make their way today, although the underlying ideas already occurred relatively early, measured in scales of computer science. In a rather general and informal summary, this paper tries to convey similarities, differences and applicability of the algorithms derived from those ideas. An introduction to optimization has intentionally been left out. Instead, the reader's intuitive knowledge is appealed to.

1 Introduction

During the last years, people have started to pay more attention to algorithms that single out and imitate certain aspects of nature for problem solving purposes. On the one hand, this phenomenon goes back to the increasing computing power which allows the simulation of models becoming more and more complex. On the other hand, why not make use of those languages and strategies for modeling and solving real–world problems nature has been providing and successfully using for the last 4.5 billion years on our planet?

The correspondence between three–dimensional projections of the Mandelbrot set and rugged glaciers does not occur just by chance, but both express the universal law of 'nonlinearity' which rules on the border between order and chaos. This bears incisive consequences on the computability and predictability of systems in which nonlinear relations occur: Despite their strong determinism, the smallest change in starting conditions or the smallest interference with the ongoing process may be reinforced dramatically resulting in a seemingly open or 'chaotic' outcome. The metaphor of a butterfly's beat of wings — at least theoretically — causing a change of weather often illustrates this reaction. But additional feedback and selection processes may 'tame' incalculable systems. Thus, they get the opportunity of *self–organization*: The 'final' state of a system becomes the starting–point for new formations, potentially threatening disturbances may not only be weakened and dominated, but may even trigger further development.

By means of a more general descriptive language the different scientific communities grow together more closely: One common language may describe the behaviour of physical, chemical, biological, medical and even economical and sociological processes. The outbreak of a panic depends on the number headless people in the beginning just as well as the victory of the VHS video system over the Beta video system (which was said to be superior technically) depended on a small original majority of people preferring VHS [5]. Maybe, we are witnessing a change of paradigm: The Arts' and Sciences' traditional approach of classifying everything into drawers could be overcome by a 'systems science' trying to reveal the conditions being responsible for the formation of structures in the material world. The ideas of quantum physics have already shaken the mechanistic view of life, the knowledge gained by nonlinear dynamics may finally detach the old theories. For those, however, who have looked at the Sciences as a replacement for religion, this changeover may be painful. For we do *not* know what we are doing when interfering with natural systems comprising nonlinearities. Even the best intentions can lead to unbounded consequences.

Deterministic chaos in our brain may even be the key to explain ‘learning’ and ‘remembering’: A permanent, but locally restricted chaos enables the brain to leave this state immediately towards an attractor (something already learned) if one of the starting conditions (sense–stimulus) changes. If one assumed a permanent ‘white noise’ as an alternative explanation, one could only reach stable patterns (learn) by slowly cooling this noise down, thus losing the capability of learning. The attractor on the other hand may easily be left in order to become prepared for new stimuli.

One can also find this chaotic behaviour in other parts of the body: Too much regularity in the frequency of our heart beat rather indicates illness than health. The number of lymphocytes (white blood corpuscles) oscillates chaotically, so does the concentration of certain hormones.

Besides those models gleaning their ideas from nature which will be explained in the following chapters, I would like to briefly mention two more which are not suitable for optimization, though:

- *Cellular Automata* have originated from an idea John von Neumann had in the beginning of the 50ies. Originally looking for a formal description of self–reproduction, he defined the following cellular automaton [35]: A two–dimensional machine with 29 states and the so–called Von–Neumann–neighbourhood (the cell itself and the four orthogonal neighbours) proved to be as powerful as a Turing–machine and may construct any other given automaton including itself. Even the simple and well–known ‘Game of Life’ may be regarded as Turing–universal [11].

  In cellular automata, there is no separation between data and instructions, but only states or configurations. Driven by an imaginary clock, various complex and even global patterns may emerge from very simple, local rules and states. For example, one can simulate the spreading of an infection (see Figure 1). Just by changing the automaton’s semantics — ‘ill’ now becomes ‘white’ and ‘healthy’ changes to ‘black’ — one has found a model of oscillating chemical reactions (e.g. the Belousov–Zhabotinskii–reaction).

  In further applications, one may describe the global behaviour of a fluid or a gas by modeling the rules of local collisions between the particles. One can also simulate so–called dendritic, i.e. tree–like growth processes leading to snowflakes or corals in nature [9].

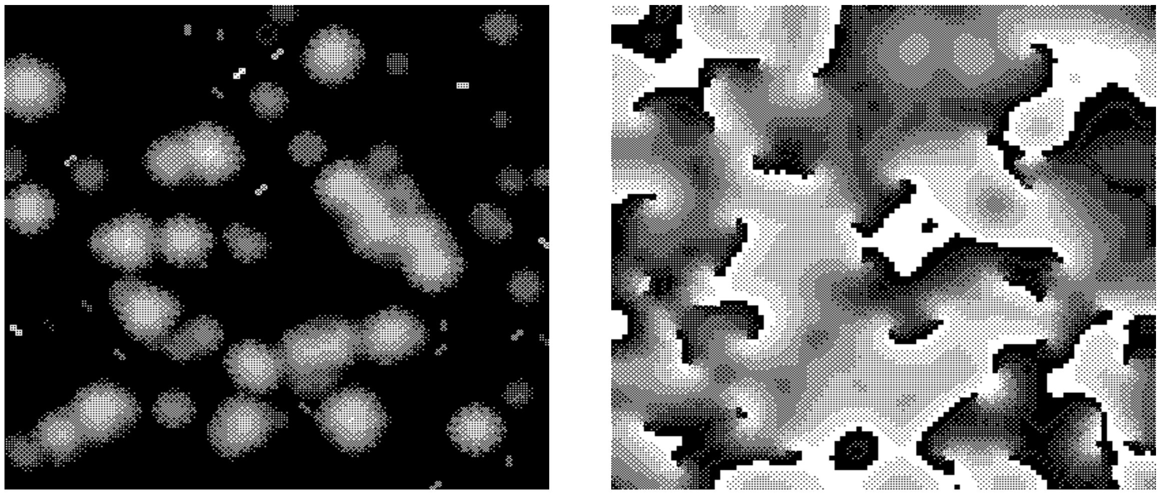

Figure 1: Spreading of an infection after 5 and 70 time steps, respectively

Because cellular automata cannot be adapted to perform any kind of optimization, they will be of no further importance in this article.

- Using models of the *immune system* for problem solving purposes seems obvious, because it proves to have a tremendous adaptability dealing with a constantly changing environment, not only recognizing situations that have already occurred (vaccination), but also coping with new, unknown intruders. In our immune system, a permanently varying network indiscriminately produces vast quantities of anti–bodies. The cell which has produced an anti–body 'fitting' on a pathogenic agent receives a signal and subsequently starts breeding rapidly, thus pouring out a large number of those anti–bodies necessary to overcome the intruder. In order to illustrate this process, one might think of an artificial neural network (see chapter 3) changing its structure over time ('idiotypic network model' [34]).

  How immune systems really work is still very much in doubt at present. Accordingly, computer simulations and their results have to be treated carefully. Today, it is difficult to estimate which further applications this model will have besides mere simulation. First results rather indicate a drifting search than an optimizing behaviour. Therefore, this onset, too, will not be explained in detail here.

2 Simulated Annealing

The freezing in and crystallization of a fluid or, in general, anything melted serves as the natural metaphor for the *simulated annealing* optimization algorithm. As long as the temperature is high, the molecules are still mobile and can move nearly independently of each other. But this capability vanishes more and more during the course of the annealing process. Given enough time, a crystal structure on a minimum energy level is reached. But if one anneals too quickly, this state cannot be reached and one obtains an inhomogeneous structure of a higher energy. For example, it takes several months to cool down the mirror of a classical, large telescope, the optical quality of which mainly depends on the homogeneity of the glass body. The slightest tension would deteriorate the observations.

Even if the comparison seems far–fetched, many traditional optimization algorithms 'cool down' their search space too quickly: Beginning at some starting–point, they move up– or downhill as long as they succeed in doing so. Obviously, one can only reach the local optimum located next to the starting–point.

The simulation of natural annealing processes on a computer is based on the so–called *Boltzmann probability distribution*

$$Prob(E) \sim \exp\left(\frac{-E}{k\,T}\right) .$$

In words: The system's energetic state in thermal equilibrium (at temperature T) is subject to a probability distribution over all possible energetic states E (k denotes the Boltzmann constant which relates energy to temperature). A system at low temperature can contain quite a lot of energy, although the probability of such an event decreases exponentially. Transferred to the artificial world of optimization, this implies that one has a chance to leave a local optimum again, find a better one and finally the global one [33, 1]. That is why the energy cannot decrease monotonically. Only by allowing intermediate deterioration, one gets the chance to find a better result than just the local minimum located next to the starting–point (see Figures 2 and 3).

The underlying idea of this method goes back as far as 1953. Metropolis et al. applied it to thermodynamic models: A system's energy changes from level E_1 to level E_2 with probability

$$p = \exp\left(\frac{-(E_2 - E_1)}{k\,T}\right) .$$

If $E_2 < E_1$ holds, the probability is 'cut off' at $p = 1$, meaning that any modification of the system resulting in a lower energy level will be accepted. In order to apply this algorithm without a thermodynamic context, one needs the following elements [19]:

- an appropriate description of the system's states,
- a mechanism generating random variations of these states,
- an objective function E to minimize and
- a control parameter T — the temperature's analogy — and a schedule decreasing T by a certain amount (usually a factor < 1) after a fixed number of successful changes.

As a more detailed example, we will now apply the simulated annealing approach to the *travelling salesman problem* (TSP). This problem consists in visiting n cities exactly once and returning to the starting–point on a tour (path) as short as possible. Although being of low practical relevance in the original form, it serves as the most intuitive representative for all sorts of scheduling problems which are highly relevant today in production planning. Furthermore, the TSP has become a standard measure relating the performance of optimization algorithms. It also represents the class of *NP–complete* problems, the members of which have one characteristic in common: The time necessary to solve such a problem *exactly* grows exponentially in n (the size of the problem). Trying out all possible tours returning to the starting city for only $n = 30$ cities on a computer being able to sequentially generate and evaluate 1 000 000 tours per second requires approximately

$$\frac{29!/2}{10^6 * 3600\ (secs./hr.) * 24\ (hrs./day) * 365\ (days/year)} = \frac{4.4209 * 10^{30}}{3.1536 * 10^{13}} \approx 1.4 * 10^{17}$$

years. The age of our universe 'only' comes to $\approx 20 * 10^9$ years. Therefore, even much faster computers will not help to solve this problem. So, we will always have to rely on robust heuristic strategies when dealing with NP–complete problems. Furthermore, the TSP implies a highly *multimodal* objective function, i.e. a function with more than one local optimum (see also Figure 8, right). But what do we have to do in order to apply the simulated annealing heuristic?

- The cities are numbered from 1 to n, and each of them is located at position $(x_i,\ y_i)$, $i = 1, \ldots, n$. The system's states comprise all possible permutations of the order in which a certain tour visits the cities.

- A tour can be modified in two ways:

  - A piece of the tour is taken out and inserted again at the same position in reverse order.
  - Again, a piece of the tour is cut out, but this time it is inserted at a randomly chosen position.

- Naturally, the length of the tour serves as the objective function:

$$E = \sum_{i=1}^{n} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \qquad \text{with } n+1 := 1$$

- Finding a good cooling schedule often is the hardest part and may require some experiments. One should choose the starting temperature with T_{Start} being significantly larger than the largest ΔE one has observed during the experiments. If — as in Figure 2 and 3 — the cities' locations are mapped to a unity square, e.g. 0.5 would represent a good initial value for T. The temperature remains unchanged on one level as long either $100 * n$ variations have been proposed regardless of their effect or $10 * n$ better configurations, i.e. shorter tours have been found. By then multiplying T with a constant factor < 1.0, the system cools down more or less slowly. One might also think of some rule varying this factor over time depending on the schedule's success so far. If no more improvements have occurred on one temperature level, the search terminates.

For different sorts of problems, finding an appropriate objective function is the easiest part. But defining sensible modifications of the states and a good cooling scheme will be more difficult in general, because only rules of thumb like the ones mentioned above exist for the simulated annealing algorithm.

The following two Figures display the final tour and the cooling process over time, respectively, for $n = 100$. Figure 2 presents the standard algorithm, Figure 3 shows the outcome if no tours worse than the best reached so far are accepted.

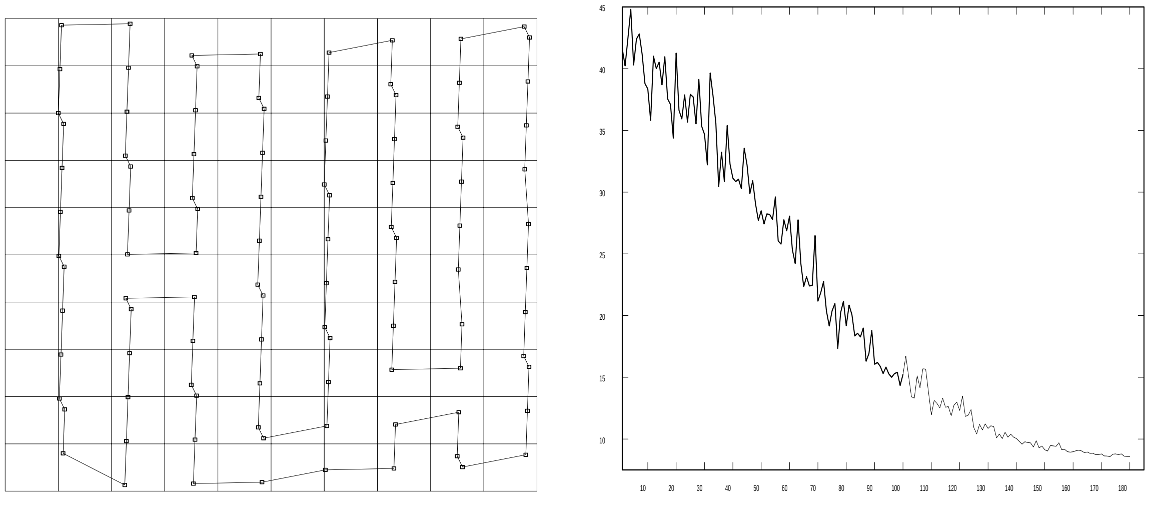

Figure 2: Length of tour: 8.5, Annealing with deterioration

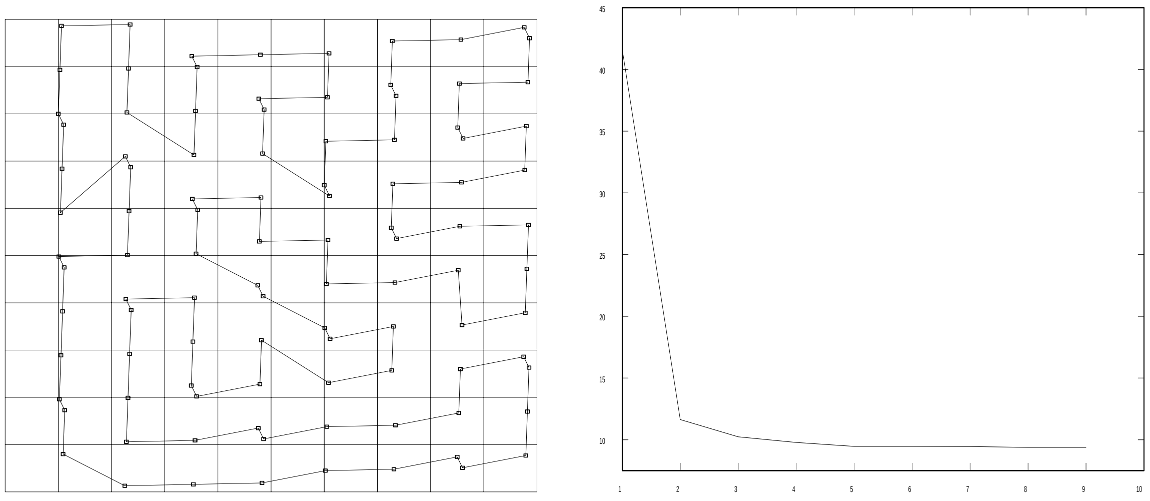

Figure 3: Length of tour: 9.4, Annealing without deterioration

3 (Artificial) Neural Networks

This notion comprises all kinds of information processing which imitates the way natural nerve systems operate: Many *neurons* work together in parallel being able to communicate with each other via synapses and axons. In a human brain, each neuron can receive the signals of up to 10 000 partners. And even neurons located far away may be reached via only 10 'relays'. Their working principle can be thought of as follows: A neuron registers the incoming signals, sums them up in a weighted sum and communicates this 'value' to those neurons connected to its outlet:

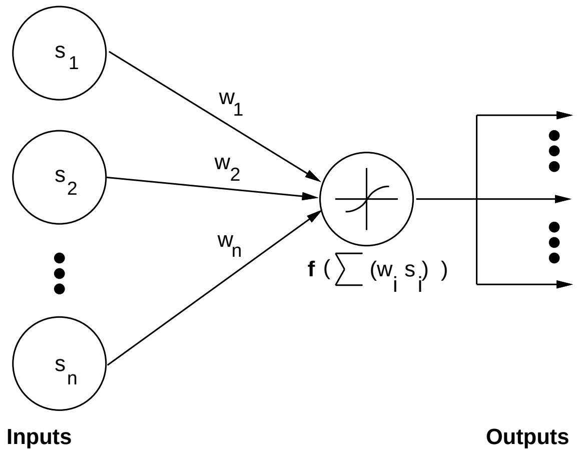

Figure 4: Working scheme of an artificial neuron

For a child, learning consists in a decay of many synapses and a reinforcement of those links activated by some pattern. This simple 'model' — proposed by Hebb [14] already in 1949 — may be sufficient for computer scientists to play around with, but it should be mentioned here that the processes really going on in a brain are still very much subject to speculation. We do know that the communication takes place with the help of at least 30 messenger substances and that not only one type of neuron exists, but many. By the way: If the human brain was simple enough for us to fully understand it, we would be too simple to do this job (E. Pugh).

Neural networks can build a framework for a learning controlled by successes and failures. Although sharing many tasks with the artificial intelligence (AI) community (pattern recognition or completion, associative memories, feature detection, fault–tolerant systems, language processing), they differ significantly from the traditional way of knowledge processing and representation: The knowledge is no longer located in e.g. the single rules of an expert system, but it is rather spread over the whole structure like in a hologram.

Already in 1958, Rosenblatt has developed the 'perceptron' [27]. Some of its properties were quite unique at that time: This simple model was capable of learning and tolerating faults. While classifying patterns, it displayed a certain ability to abstract by responding correctly to unknown inputs, i.e patterns it had not been trained with. Furthermore, it could be proved that this learning strategy is capable of learning all patterns it can possibly classify in a finite number of steps. But in 1969, there was a serious set–back, when Minsky and Papert discovered and proved the main weakness of a perceptron: It is necessary to adapt the synaptic weights with an accuracy growing exponentially with the problem's complexity. By mistake, they assumed this result (criticism) would hold for this type of network with more than two layers, too [22]. Until the beginning of the 80ies, this field of research could not recover. But then, the works of Hopfield [18] and Rumelhart, Hinton and Williams [29] have called it back to life and have given it an impetus it still thrives upon today.

The main parts of a neural network are

- the neurons (working elements) defining
  - a set of activating states,
  - an activation function,
  - an output function depending on the present state,
- a topology linking the neurons (the network) and
- a (learning) rule how to adapt the synaptic weights.

A certain choice of those parts mentioned above results in a certain type of neural network. Today, one can distinguish the following most important models [23, 25]:

- A *perceptron* consists of only one layer of a fixed number of neurons. Its task is to classify into categories the input patterns, each containing m characteristics. Whether a given network is capable of solving a certain problem, is not guaranteed a priori: The patterns have to be separable by a $m-1$ dimensional hyperplane. Spirals being intertwined (see Figure 5) cannot be separated by a straight line; therefore, the perceptron has no chance to succeed. By simply adding another feature to the data set (here: a third dimension), one may often overcome this situation.

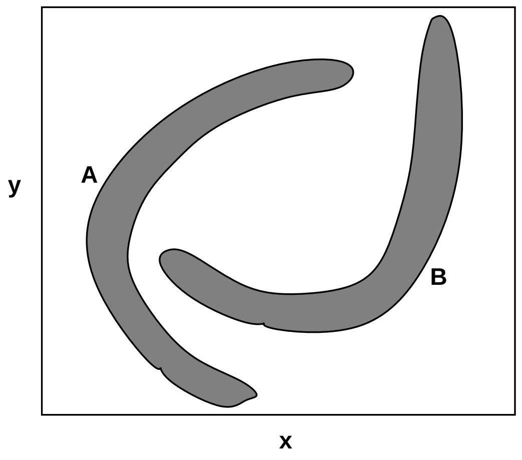

Figure 5: Patterns A, B not being separable linearly in the (x, y)–plane

Even such a simple problem like calculating the 'exclusive or' of two binary inputs lies beyond the capabilities of this network type. The four points $(0,0)$, $(1,1)$ — these constituting (output) class 0 — and $(0,1)$, $(1,0)$ — the members of class 1 — cannot be separated appropriately by a straight line.

But neural networks with more than one layer go beyond the perceptron's limited scope:

- The multi–layered *perceptron with back–propagation* ('backprop') probably represents the best known network model today. Training a net of this type has become possible since Rumelhart, Hinton and Williams [29] have managed to generalize the delta–rule, applicable only for nets with one input and one output layer, resulting in the recursive back–propagation learning algorithm. Now, neural networks can perform nearly arbitrary mappings between inputs and outputs, because the additional layers somehow encode and compress the information coming in. Figure 6 illustrates the topology (structure) of such a net:

  The following 'sigmoid' function often defines the output of one single neuron depending on all its synaptic inputs:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  Starting with randomly chosen initial values of the synaptic weights, the back–propagation algorithm tries to adjust those weights in such a way that the deviation between input and expected output becomes smaller and smaller. If one chooses

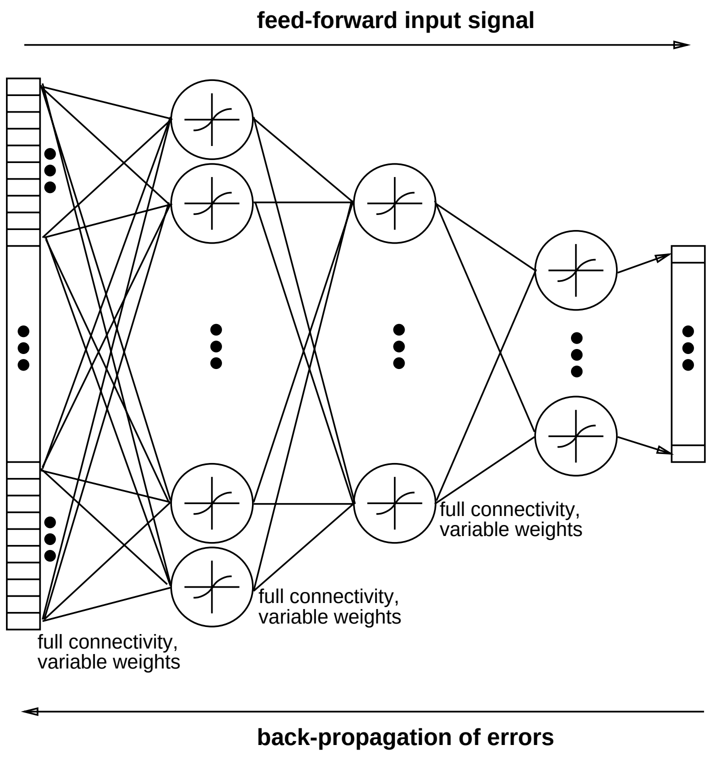
feed-forward input signal
full connectivity, variable weights
full connectivity, variable weights
full connectivity, variable weights
back-propagation of errors

Figure 6: Back–propagation network with three layers

the stepsize (learning rate) appropriately, the algorithm follows the path of steepest descent (gradient) of the error function. When using the sigmoid function mentioned above, the correction term for the weights of the output layer calculates to

$$\Delta w_{jk} = \eta \, o_j \, \delta_k,$$

with:

- Δw_{jk}: synaptic weight between neuron j of the previous layer and neuron k of the output layer,
- η: learning rate $\in [0, 1]$,
- o_j: output of neuron j in the previous layer,
- $\delta_k = o_k \, (1 - o_k) \, (t_k - o_k)$ with:
  - o_k: calculated output of neuron k in the output layer and
  - t_k: desired output of neuron k in the output layer.

The correction term for the hidden layer(-s) looks a little different, because one has to add up the error of all following layers. For a more detailed discussion, see e.g. [8].

The back–propagation algorithm is applicable to neural networks with an arbitrary number of hidden layers. But the time needed to train such a network grows enormously. Furthermore, since using a gradient algorithm, the training phase can get stuck in a local optimum differing from the global one. If one uses perfect, i.e. undisturbed patterns for training, the resulting net will have great difficulties in deciding about an input being slightly different. Moreover, it is still unknown in general, how many layers with how many neurons are needed to solve a give problem, and what will happen if the network is oversized.

- *Hopfield–nets* bear much resemblance to the idea of simulated annealing introduced in chapter 2. In 1982, Hopfield [18] could prove that the behaviour of a certain network type, which has later been named after him, corresponds to a physical system containing a large number of elementary magnets ('spin glasses'), thus leading to a common energy function. Hence, analytical methods from physics could be used to describe this sort of network. Accordingly, the maximum information capacity of a Hopfield net comprising n neurons calculates to $2 * n$. But this number decreases, if one wants the net to recognize patterns subjected to noise as well.

  Figuratively speaking, this model rolls a ball through the mountain–range formed by the 'energy' function (see for instance Figure 8) and searches for a valley with the lowest bottom. These bottoms correspond to the patterns learned by the network in the training phase. The size of their basins of attraction, the valleys' narrowness with respect to the size of the ball and the monotony of the path to get there determines how susceptible to disturbances of the inputs the course of the ball is.

  All network models presented so far rely on the same basic building blocks. A feedback loop constitutes the new characteristic of Hopfield–nets: The outputs of the one–layer net are fed into the inputs again. In this way, the net can restore the missing parts or parts disturbed by noise step by step. A program working in a traditional way could only compare a certain number of stored patterns to the present input and then respond with the least deviating pattern. A well trained *associative memory*, on the other hand, determines its answer according to an abstract and holistic similarity criterion being unknown beforehand. But nevertheless, this type of network does not perform very well as an associative memory because of its limited abstracting capabilities. But the power of natural brains especially relies on this feature, for our environment rarely confronts us with situations fitting exactly on ones we have experienced before. But again, in the ideal case, a model reflects all the knowledge about a certain area. With so much remaining to be explored by the cognitive sciences, nobody can really expect the present artificial networks to be perfect.

- *Kohonen's (self–organizing) feature maps* have been designed as two dimensional layers of neurons [20]. In contrast to back–propagation networks, any incoming signal only causes an amount of activity limited spatially in the following way: As in natural brains, similar signals only stimulate neighbouring (adjacent) neurons. Again using randomly chosen initial synaptic weights, an input vector – also selected randomly from the training set — represents the stimulus. The neuron feeling most 'responsible' for this input moves a little bit towards the vector and drags some of its neighbours along. The number of those neighbours decreases over time. Of course, the speed of this reduction strongly influences the quality of the results. On the one hand, if the neighbourhoods are made smaller too quickly, the network runs into 'stress' and 'freezes' in a state not being the global optimum. On the other hand, one gives away computing time. The analogy to chapter 2 is rather evident.

  As an example, we use the travelling salesman problem again: A network of neurons (for example: three times the number of cities) is supposed to learn a path as short as possible visiting all cities. In each iteration, a town $C_{x,y}$ chosen at random drags the nearest neuron $N_{x,y}^{closest}$ a little towards it, thus deforming the net which has started as a circle. A certain number of neighbours also moves towards the city. The further away these neurons are from the 'best match' neuron, the smaller their co–operation. During the course of the iterations, the co–operation vanishes more and more, and one can observe the net first shaping into a rough structure, but the neurons then get closer and closer to the cities until there is correspondence. Naturally, having started with more neurons than cities, several neurons will lie on one city in the end. Figure 7 (left) tries to convey an impression of what has gone on.

  The example in Figure 7 only contains 20 cities for clarity reasons. One can see 60 neurons, starting on a circle, then learning an approximate shape of the tour and finally converging to the global optimum in this case. This is not always guaranteed, though. But with one run only lasting for about 10 seconds on a workstation, one can afford several runs.

  In the field of robot control, this algorithm yields good results, too, without having to undergo any changes: The cities now represent positions where a robot is supposed to drill holes. So, visiting those positions on a short tour is desirable in order to minimize abrasion. In another impressive example, although being irrelevant for real world engineering applications, a net manages to balance a bar vertically.

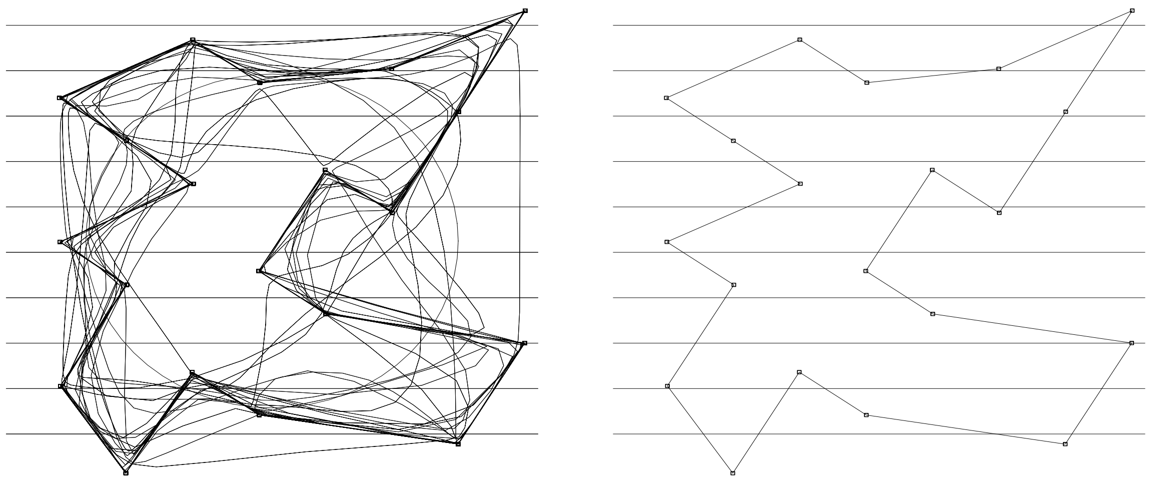

Figure 7: Development of a tour, best tour

- At the moment, *Carpenter / Grossberg–classifiers* are probably the models being closest to what is really going on in natural brains. They include feedback loops, a short–term and long–term memory and a function controlling the alertness, i.e. lowering it if the outside world does no longer provide any stimuli. For the first time, a model — like a brain — is capable of learning continuously without destroying (forgetting) the previous knowledge. So, in one area, you can observe islands of stability, whereas other regions produce seemingly chaotic activity. Grossberg's *adaptive resonance theory* (ART) laid the foundations of this model in the 70ies. But presenting further details would go beyond the scope of this paper.

Conclusively, one cannot reliably estimate the applicability of neural networks not until there is hardware available without a fixed wiring, i.e. optical computers which can generate and destroy links among computing elements dynamically. Then, there will be no more necessity to simulate a network with its intrinsic parallelism on a sequential computer. But attempting to increase the power just by using more and more neurons and/or layers misleads: With a sufficiently large number of neurons with respect to the size of the set of training data, the network memorizes all input patterns instead of learning an abstraction.

The following table summarizes the most important features of those types of neural networks presented in this paper [3]:

network model	*topology*	*computable values*	*learning*
perceptron	one layer feed–forward	binary and continuous	supervised
perceptron with back–propagation	several layers feed–forward	binary and continuous	supervised
Hopfield net	one layer with feedback	binary	supervised
Kohonen feature map	2–dimensional grid of neurons	continuous	unsupervised
Carpenter/Grossberg–classifier	several layers	binary and continuous	unsupervised

4 Evolutionary Algorithms

Because both algorithms presented in this chapter — *evolutionary strategies* (EA's) and genetic algorithms (GA's) — imitate certain principles of organic evolution, a common introduction seems appropriate.

Despite some of nature's astonishing results — from an engineering point of view — the idea to imitate certain principles of nature in order to create a new optimization algorithm has been ridiculed and rejected rigorously for the following reasons:

- Evolution proceeds very slowly compared to the lifetime of human beings, and therefore, it can only be observed very rarely.
- There is no global 'objective function'.
- A mutation–selection scheme with a stochastic description (modelling) of disturbances appears to waste resources compared to algorithms which specialize in a certain type of problem. Or in other words: Since Isaac Newton the deductive approach dominates the inductive one.
- When first trying to imitate evolution on a computer, the slogan 'Survival of the fittest' (singular!) was taken too literally. By the way, this slogan attributed nearly automatically with Charles Darwin was meant to be a provocation by Herbert Spencer.
- The element of chance inherent in evolution often leads to a misunderstanding: Simply 'rolling the dice' could indeed not have produced the diversity of complex living systems observable today. Accordingly, taking random samples from the high dimensional parameter space of an objective function in order to find the global optimum *must* fail (*Monte–Carlo search*). One can even prove that an algorithm enumerating the search space completely always performs better, because it avoids visiting a point more than once.

But by looking at evolution as a cumulative, highly parallel sieving process, the results of which pass on slightly modified into the next sieve, the amazing diversity and efficiency on earth no longer appears miraculous. When speaking about models of this process, the objective cannot be the re–invention of this diversity in quick–motion on a computer. But the point is to isolate the main mechanisms which have led to today's world and which have been subjected to evolution themselves. Inevitably, nature has 'invented' a mechanism allowing individuals of one species to exchange parts of their genetic information in order to meet changing environmental conditions in a better way (recombination or crossing–over). In the case of slipper animalcules this has got nothing to do with breeding. With this principle and others, nature managed to 'learn' even complex models of the environment: For instance, at branching points, the ratio of the diameters in a system of blood–vessels is close to the theoretical optimum given by the laws of fluid dynamics ($2^{-1/3}$). For other examples, see [26].

Genetic algorithms and evolution strategies originated independently from each other on both sides of the Atlantic Ocean in the middle of the 60ies competing with other (global) optimization algorithms. Later, their emergent properties became a topic of interest, too. In both methods, a population tries to 'learn' (find) the global optimum of the system to be optimized collectively, imitating the following principles:

- population
- inheritance (recombination or crossing–over)
- mutation
- selection

Mutation and recombination may be regarded as the 'engine' of the vehicle evolution, whereas selection would provide the 'steering wheel' in this metaphor. Although only being able to evaluate the phenotype, the degree of adaptation visible on the outside, the genotype which determines the outward appearance is evaluated indirectly, too. From the metaphor, one may also derive the importance of a permanent source of new variants, because a vehicle standing still does not need any steering. Of course, the degree of an individual's adaptation has to be re–evaluated all the time, for the environment is subjected to changes caused by the organisms living in and on it. For further fundamentals of genetics and biology, see [13].

4.1 Evolution Strategies

Rechenberg and Schwefel [24, 30] developed the *evolution strategies* when they wanted to optimize technical objects like a nozzle. No closed form analytical objective function was available, and hence, no applicable optimization method existed at that time. So, these strategies resemble a person experimenting, in the worst case not knowing anything about the object (black box situation). All there is to play around with is a finite number of regulators which have to be tuned in such a way that the output meets some kind of optimality criterion. Already for a black box with three knobs and a nonlinear output function proves to be too difficult for humans to optimize by hand.

The direction and the stepsize of the variations are crucial for all optimization methods. Figure 8 illustrates that these values have to be adapted all the time until the convergence has been reached. Starting far away from the optimum, large steps are advantageous in the beginning, but later, they have to become smaller as the algorithm approaches the extreme point. The right part of Figure 8 illustrates the situation of an optimization method using another metaphor: It resembles hillclimbing in foggy conditions where one is content having gone up one hill, although higher peaks might exist across the valley.

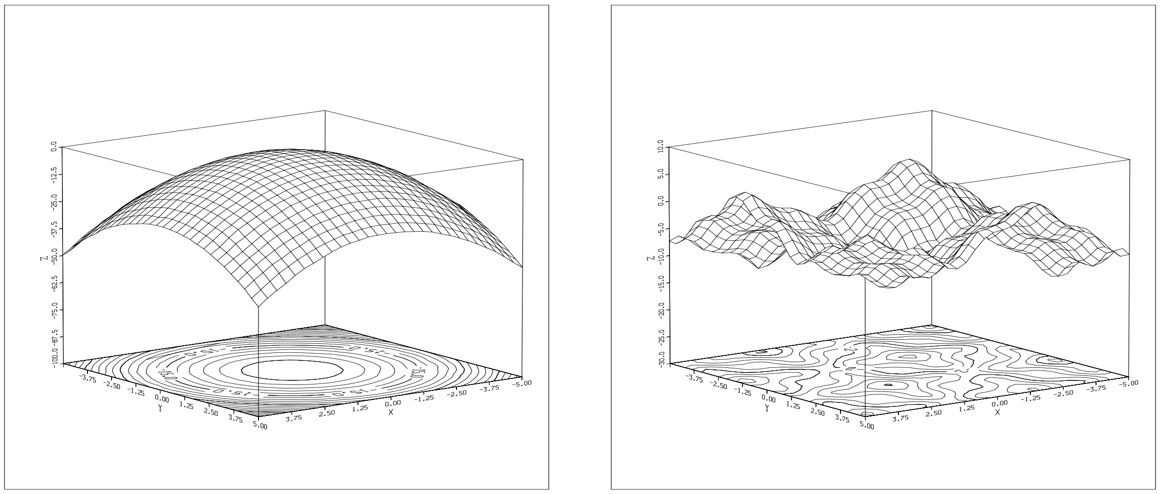

Figure 8: Topologies of objective functions — easy and difficult

The first attempt to imitate principles of organic evolution on a computer was still very similar to those iterative optimization methods known up to that time: In a two–membered or (1+1) evolution strategy, one 'parent' generates one offspring per generation by applying a normally distributed mutations (i.e. smaller steps are more likely than big jumps) until a 'child' performs better than its ancestor and takes its place. Because of this simple structure, theoretical results for stepsize control and convergence velocity could be derived: The ratio between successful and all mutations should come to 1/5. This first algorithm was then enhanced to a $(\mu+1)$ strategy which could incorporate recombination for the first time, because several parents are available. The mutation scheme and the exogenous stepsize control were taken across unchanged.

Schwefel [30, 31] generalized these strategies to become the multimembered or $(\mu \overset{+}{,} \lambda)$ evolution strategy which will now be explained in more detail. An *individual* on the computer consists of the following 'genes':

- Real–valued *object variables* x_i have to be tuned by recombination and mutation in such a way that an objective function reaches its global optimum. Referring to a metaphor mentioned previously, these are the regulators of a black box.
- Real–valued *strategy variables* or stepsizes σ_i determine the mutability of the x_i. They represent the standard deviation of a $(0, \sigma_i)$ Gaussian distribution being added to each x_i as an undirected mutation. With an expectancy value of 0, the parents will produce offsprings similar to them on average. In order to make a doubling and a halving of a stepsize equally probable, the σ_i themselves mutate log–normally distributed from generation to generation. Inside those stepsizes, the population hides the internal 'model' it has made of its environment so far. So, a self–adaptation of the stepsizes has taken over from the exogenous control within the $(1+1)$ strategy. This works because the selection will sooner or later prefer those individuals having learned a good model of the objective function, thus producing better offsprings. Hence, learning takes place on two levels.

  Because this kind of algorithm prefers a search along the coordinate axes, one can optionally include inclination angles which also undergo mutation and recombination. With these additional strategy variables, the mutation ellipsoids can move (turn) freely in the search space $\mathbb{R}^n$:

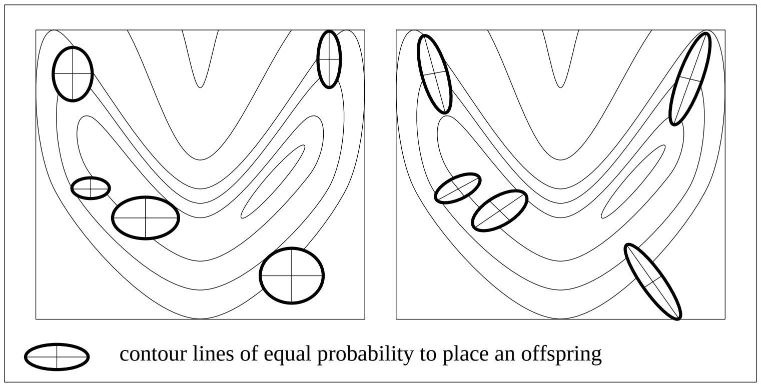

Figure 9: Mutations without and with inclination angles (taken from [15])

Mathematically, those angles represent the covariances of the mutations. In nature, a second level also exists, e.g. repair enzymes. Furthermore, biological phenomena like polygeny (several genes making up one feature) or pleiotropy (one gene influencing several features) support the hypothesis.

- Depending on an individual's x_i, the resulting objective function value $f(\underline{\mathbf{x}})$ serves as the 'phenotype' (fitness) in the selection step.

The μ parents produce λ offsprings per generation in three steps:

1) Object and strategy variables of an offspring are put together according to the type of recombination chosen. Each parent has an equal opportunity to reproduce. One may choose from the following kinds of recombination, independently for object and strategy variables:

   - no recombination,
   - discrete recombination taking the genetic material either from only two randomly chosen parents or taking it from all,
   - intermediate recombination between two or among all parents using the mean at each locus,
   - crossing–over between two parents with one or two intersection points.

   In a test series with 50 objective functions, the best results have been achieved with a discrete recombination of the object variables and an intermediate recombination of the strategy variables.

2) First, the strategy variables mutate, and their new values are used for mutating the object variables.

3) According to the strategy chosen, the selection then takes place: In a plus strategy, the μ best of all $\mu + \lambda$ individuals survive to become parents in the next generation. Using the comma variant, the selection takes place only among the λ offsprings. The second scheme is more realistic and therefore, more successful, because no individual may survive forever which could at least theoretically occur using the first variant. Again (see chapter 2), a strategy allowing intermediate deterioration, performs better. Only by 'forgetting' individuals with a good phenotype, which may have been achieved with an internal model that is no longer appropriate for further progress, a permanent adaptation of the stepsizes (and inclination angles) can take place. This becomes particularly evident if the location of the optimum changes over time (see Figure 10). The plus variant sticks to good phenotypes, which of course

become worse and worse as time passes on (left), whereas the comma strategy manages to follow the optimum which moves on periodically (right).

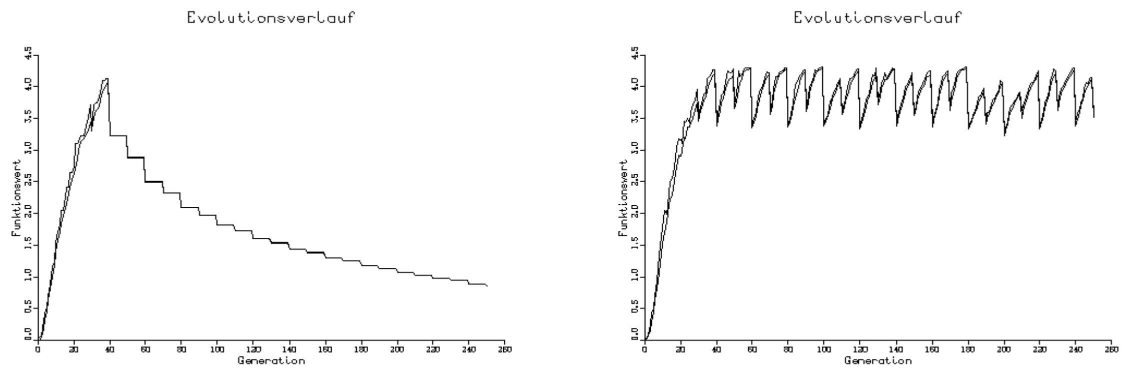

Figure 10: Plus– and comma–strategy applied to a moving optimum

By choosing a certain ratio μ/λ, one can determine the convergence property of the evolution strategy: If one wants a fast local convergence, one should choose a small ratio, but looking for the global optimum, one should decide in favour of a 'softer' selection.

Evolution strategies are adaptable to nearly all sorts of problems in optimization, because they need very little information about the problem — especially no derivatives of the objective function. So, they are capable of coping with high dimensional, multimodal, nonlinear, discrete / continuous objective function subject to linear and / or nonlinear constraints. The objective can also 'hide' the result of a simulation, it does not have to be given in a closed form. This also holds for the constraints which may represent the outcome of a finite elements method (FEM) [6]. Evolution strategies have been adapted to vector optimization problems [21], and they can also solve NP–complete problems like the TSP which has been introduced in chapter 2 [2, 28]. One can even think of applications where the strategy only provides propositions and a human being selects according to her / his preferences that cannot be formalized like aesthetic criteria. With an underlying GA (see below), such a system already exists (BUGS — Better to use genetic systems). It has been written in the spirit of Richard Dawkins' *blind watchmaker* [7]. The user can play God or fitness function, rather, trying to evolve organisms looking like e.g. (butter-)flies, or some sort of insect in general. The parallel hardware becoming more and more available also offers new prospects, because the strategies' inherent parallelism can then be exploited [16].

But besides their problem solving capability, evolution strategies can also serve as a simple model of the underlying natural processes, thus understanding evolution — although in an ideal and artificial world — a little better. In this context, 'model' is intended in a descriptive way, not explanatory.

Using

$$f(\underline{x}) := \sum_{i=1}^{30}(i * x_i^2)$$

as the objective function, $n = 30$ stepsizes have to be tuned properly with respect to each other in order to achieve maximum progress. Figure 11 displays the performance of different strategies:

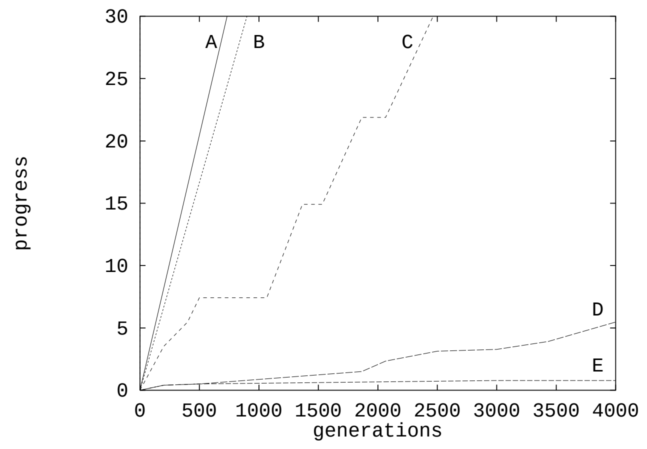

Figure 11: Learning of a scaling

Curve E represents a $(1, 10)$ strategy which fails because the convergence velocity decreases reciprocally to n. If only one offspring survives, an individual operating in a subspace of $\mathbb{R}^n$ will prevail over others. Increasing λ does not help, but the simultaneous increase of μ and λ leads the way. Curves D ($(3, 6)$) and C ($(6, 30)$) display the impact of recombination on convergence velocity. But only the comparison with the best variant can show its true capabilities. Curve A already starts with optimum stepsize relations ($\sigma_i = c/\sqrt{i}$) and can therefore abandon recombination. A $(15, 100)$ evolution strategy starting 'dull' and having to gain this knowledge on the fly nearly proceeds at the same speed (B). But how many offsprings should survive to become the parents of the next generation? Varying μ with a constant $\lambda = 100$, Figure 12 answers this question.

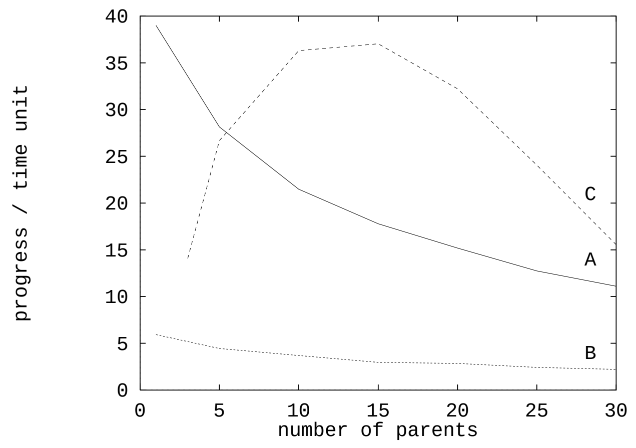

Figure 12: Optimum selection pressure

Curve A again represents a strategy with perfect knowledge calculated analytically, whereas in B, randomly chosen relations of the σ_i were fixed. In curve C, the stepsizes could self–adapt. Not surprisingly, $\mu = 1$ proved to be the best choice for A and B where only one stepsize had to adapt, because the relations had been fixed. On the other hand, one should choose μ between 12 and 20 if one wants learning to take place (C). Two observations from Figure 12 are remarkable: The $(15, 100)$ strategy converges nearly as fast as variant A, and it performs better than a $(15, 100)$ strategy with perfect knowledge. One can regard this phenomenon as a *synergetic* effect: 15 'fools' perform better *collectively* than the same number of 'specialists'.

So far, the Figures 11 and 12 displayed the progress on the phenotypic level. But by looking at one of the x_i alone, more details reveal (see Figure 13): Because the objective function depends on 30 parameters, the development of one variable does not necessarily correspond to the overall progress. In a nonlinear environment, a big step of one variable towards the optimum allows others to even move away from the optimum. Under high selective pressure the evolution no longer runs continuously, but takes place unsteadily between stagnation phases — a phenomenon biologists call 'punctuated equilibria' [10].

To sum up, the self–adaptation capability of evolution strategies depends on the following factors [32]:

- The population has to be sufficiently large. Not only the temporary best should be allowed to reproduce, but a set of better individuals. Biologists have coined the

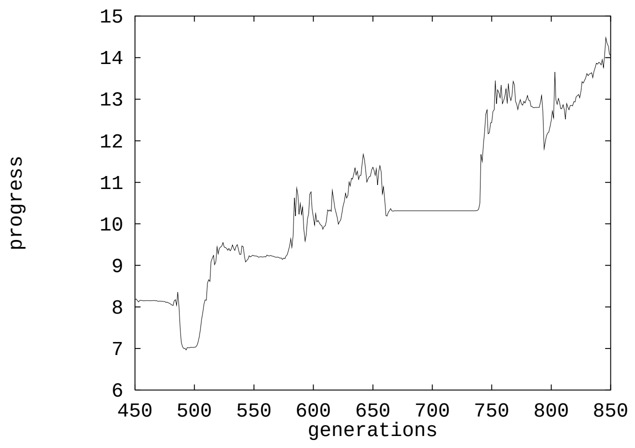

Figure 13: Development of one decision (object) variable

term 'requisite variety' being necessary to prevent a species from becoming poorer and poorer genetically and eventually dying out.

- Within a population, the individuals should recombine their knowledge with that of others (co–operate).
- In order to allow better internal models (stepsizes) to provide better progress in the future, one should accept deterioration from one generation to the next. A limited life–span in nature is not a sign of failure, but an important means of preventing a species from 'freezing' genetically.

4.2 Genetic Algorithms

In principal, the scope of *genetic algorithms*, which have been developed by John Holland [17] taking up Bremermann's earlier idea of a binary coding [4], is identical to that of evolution strategies. They imitate the same natural principles which have already been mentioned in the introduction of this chapter. Despite the common roots, differences exist and will be mentioned in the following [15]:

- The coding (usually binary) of the object variables represents the biggest difference. The knobs of the black box have turned to switches [12]. In order to calculate an individual's fitness the bitstring has to be decoded first, thus also determining the interval where one suspects the optimum. Constraints being violated reduce the fitness value by means of a penalty function.

- Mutation and recombination or crossing–over, respectively, form the genetic operators, but with a different importance than in ES: Crossing–over which typically occurs with probability 0.6 represents the main search operator, whereas mutation occurring only with probability 0.0001 prevents the total loss of an allele (0 or 1) in the population at a certain position. Both probabilities are determined exogenously, and they do not adapt during the search, although a decrease of the mutation rate seems appropriate if many bits have already been set correctly. Furthermore, a self–adaptation does not work using the standard GA selection schemes.
- When using binary values, an intermediate selection becomes impossible.
- Genetic algorithms do not generate surplus offsprings like ES (and nature) do.
- The selection step constitutes another main difference. Reproduction probabilities proportional to the individual's contribution to the overall fitness of the population can lead to the dominance of a good individual in the following generation. Such a behaviour may already occur in simple, unimodal topologies and slow down the convergence velocity. Of course, other selection schemes have been developed, e.g. ranking where a certain number of offsprings is assigned to an individual depending on its relative fitness (rank) in the population. But for self–adaptation on a second level to work, one has to introduce an extinctive selection like the one in ES.

An overview over the range of applications can be found in David Goldberg's book [12], as well as an introduction to *classifier systems* in which a GA works on binary codings of rules which have to last by means of correct predictions about their environment.

The following table sums up the main differences between GA's and ES's:

genetic algorithms	*evolution strategies*
binary coding	real–valued variables
limited search space because of decoding	(in principle) unlimited search space
recombination most important operator	mutation most important operator, recombination for the self–adaptation of the stepsizes
no collective learning of strategy parameters	collective learning of strategy parameters
learning on one level	simultaneous learning on two levels

5 Summary / Outlook

Especially for problems which cannot be solved (optimized) analytically, the new methods presented in this paper can provide results not obtainable otherwise. Convergence proofs exist for simulated annealing, genetic algorithms and the $(1+1)$ evolution strategy, but they are not of much practical relevance due to the unlimited time they assume. But in engineering applications, there often is no need to know for sure about the global optimum for two reasons:

- Doing better than before is sufficient.
- One cannot adjust the equipment as accurately as computed. If a simulation model has to be optimized, the underlying data are usually subjected to errors.

Finally, the algorithms presented here should be looked upon from a more abstract level in order to stress the aspects they have in common:

- All these methods can be written down mathematically in very few lines. One has to provide only a few (strategy) parameters as the framework in which learning can take place.
- All algorithms strongly rely on chance modifying states and 'proposing' them to the selection component. The emergent order is achieved in the long run by selectively stabilizing 'good' states.
- In all cases, the knowledge is distributed over the respective structure. Intelligence is a collective phenomenon.
- Therefore, learning can only take place collectively.

It seems as if the notions 'self–organization' and 'evolution' only describe the same phenomena either from a physicist's or a biologist's point of view. They both describe those reactions in which a system with many attractors steers towards a stable one, successfully defending it against disturbances — within certain limits of course.

The aim of this overview was the introduction of algorithms still being capable of optimizing those models which do no longer fulfill the traditional mathematical prerequisites like smoothness and differentiability. The freedom thus gained in the modelling phase has to be paid for by a loss of convergence certainty and velocity. Of course, the simplex algorithm or one of its variants is the most efficient method if the objective function and the constraints are linear. But do linear models always reflect the complex relations in the real world correctly? In economics, does the adaptability and learning capability of markets not hint at the existence of stable islands in chaos, and that therefore those systems are governed by nonlinear rules with feedback loops as well? And is the chaos surrounding those islands not the key to adaptability? Does the last stock market crash (being without any economic background) not imply that highly ordered systems with their rigid rules and deceptive security respond a lot more sensitively to disturbances?

References

[1] **E. H. L. Aarts and J. H. M. Korst**, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley & Sons, Chichester, 1989.

[2] **P. Ablay**, *Optimieren mit Evolutionsstrategien*, Spektrum der Wissenschaft, 1987, pages 104–115.

[3] **T. Barr**, *Netze im Aufwind*, c't, 1991.

[4] **H. J. Bremermann**, *Optimization through Evolution and Recombination*, in: Self–Organizing Systems, M. C. Yovits, G. T. Jacobi and D. G. Goldstein (Eds.), Spartan Books, Washington, D.C., 1962, pages 93–106.

[5] **J. Briggs and F. D. Peat**, *Turbulent Mirror — An Illustrated Guide to Chaos Theory and the Science of Wholeness*, Harper & Row Publishers, New York, 1989.

[6] **I. Campos Pinto**, *Wissensbasierte Unterstützung bei der Lösung von Optimierungsaufgaben*, PhD thesis, Universität Dortmund, Fachbereich Informatik, 1989.

[7] **R. Dawkins**, *The Evolution of Evolvability*, in: Artificial Life, C. Langton (Ed.), Redwood City, California, 1989, Addison–Wesley.

[8] **J. E. Dayhoff**, *Neural Network Architectures: An Introduction*, Van Nostrand Reinhold, New York, 1990.

[9] **A. K. Dewdney**, *Computer–Kurzweil*, Spektrum der Wissenschaft, 1988.

[10] **N. Eldredge and S. J. Gould**, *Punctuated Equilibria: The Tempo and Mode of Evolution Reconsidered*, Paleobiology, 3 1977, pages 115–151.

[11] **M. Gardner**, *The Fantastic Combinations of John Horton Conway's New Solitaire Game of 'Life'*, Scientific American, 1970.

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

[13] **L. Gonick and M. Wheels**, *The Cartoon Guide to Genetics*, Barnes & Noble Books, New York, 1983.

[14] **D. O. Hebb**, *The Organization of Behaviour*, Wiley & Sons, New York, 1949.

[15] **F. Hoffmeister and T. Bäck**, *Genetic Algorithms and Evolution Strategies: Similarities and Differences*, in: Parallel Problem Solving from Nature, Proceedings of the 1st PPSN–Workshop, Dortmund, 1990, H.-P. Schwefel and R. Männer (Eds.), Volume 496 of Lecture Notes in Computer Science, Berlin, 1991, Springer, pages 445–469.

[16] **F. Hoffmeister and H.-P. Schwefel**, *A Taxonomy of Parallel Evolutionary Algorithms*, in: PARCELLA '90, G. Wolf, T. Legendi and U. Schendel (Eds.), Berlin, 1990, Akademie–Verlag, pages 97–107.

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

[18] **J. J. Hopfield**, *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, in: Proc. Nat. Acad. Sci. USA, 1982, pages 2554–2558.

[19] **S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi**, *Optimization by Simulated Annealing*, Science, 220 1983, pages 671–680.

[20] **T. Kohonen**, *Self–Organization and Associative Memory*, Springer, Berlin, 1984.

[21] **F. Kursawe**, *A variant of evolution strategies for vector optimization*, in: Parallel Problem Solving from Nature, Proceedings of the 1^{st} PPSN–Workshop, Dortmund, 1990, H.-P. Schwefel and R. Männer (Eds.), Volume 496 of Lecture Notes in Computer Science, Berlin, 1991, Springer, pages 193–197.

[22] **M. Minsky and S. Papert**, *Perceptrons*, MIT Press, Cambridge, Massachusetts, 1969.

[23] **Y. H. Pao**, *Adaptive Pattern Recognition and Neural Networks*, Addison–Wesley, Reading, Massachusetts, 1989.

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

[25] **H. Ritter, T. Martinetz and K. Schulten**, *Neuronale Netze*, Addison–Wesley, Bonn, 1990.

[26] **R. Rosen**, *Optimality Principles in Biology*, Butterworths, London, 1967.

[27] **F. Rosenblatt**, *Principle of Neurodynamics*, Spartan, New York, 1962.

[28] **G. Rudolph**, *Global Optimization by means of Distributed Evolution Strategies*, in: Parallel Problem Solving from Nature, Proceedings of the 1^{st} PPSN–Workshop, Dortmund, 1990, H.-P. Schwefel and R. Männer (Eds.), Volume 496 of Lecture Notes in Computer Science, Berlin, 1991, Springer, pages 209–213.

[29] **D. E. Rumelhart, G. E. Hinton and R. J. Williams**, *Learning Internal Representations by Error Propagation*, in: Parallel Distributed Processing I: Foundations, D. E. Rumelhart and J. L. McClelland (Eds.), MIT Press, Cambridge, Massachusetts, 1986, Chapter 8.

[30] **H.-P. Schwefel**, *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, Birkhäuser, Basel, 1977.

[31] ——, *Numerical Optimization of Computer Models*, Wiley & Sons, Chichester, 1981.

[32] ——, *Collective Phenomena in Evolutionary Systems*, in: 31^{st} Annual Meeting of the International Society for General System Research, Budapest, 1987, pages 1025–1033.

[33] **P. J. M. van Laarhoven and E. H. L. Aarts**, *Simulated Annealing: Theory and Applications*, D. Reidel, Dordrecht, 1987.

[34] **F. J. Varela and H. Bersini**, *Hints for Adaptive Problem Solving Gleaned from Immune Networks*, in: Parallel Problem Solving from Nature, Proceedings of the 1st PPSN–Workshop, Dortmund, 1990, H.-P. Schwefel and R. Männer (Eds.), Volume 496 of Lecture Notes in Computer Science, Berlin, 1991, Springer, pages 343–354.

[35] **J. von Neumann**, *Theory of Self–Reproducing Automata*, University of Illinois Press, 1966.