Chapter 2
Evolutionary Algorithms for Chaos Researchers

Ivan Zelinka and Hendrik Richter

Abstract. Evolutionary algorithms are search methods that can be used for solving optimization problems. They mimic working principles from natural evolution by employing a population–based approach, labeling each individual of the population with a fitness and including elements of random, albeit the random is directed through a selection process. In this chapter, we review the basic principles of evolutionary algorithms and discuss their purpose, structure and behavior. In doing so, it is particularly shown how the fundamental understanding of natural evolution processes has cleared the ground for the origin of evolutionary algorithms. Major implementation variants and their structural as well as functional elements are discussed. We also give a brief overview on usability areas of the algorithm and end with some general remarks of the limits of computing.

2.1 Historical Facts from a Slightly Different Point of View

Evolutionary algorithms, or better evolutionary computational techniques (ECT), are based on principles of evolution which have been observed in nature long time before they were applied to and transformed into algorithms to be executed on computers. When next reviewing some historical facts that led to evolutionary computation as we know it now, we will mainly focus on the basic ideas, but will also

Ivan Zelinka
Tomas Bata University in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511,
Zlin 76001, Czech Republic
and
VSB-TUO, Faculty of Electrical Engineering and Computer Science, 17. listopadu 15,
708 33 Ostrava-Poruba, Czech Republic
e-mail: zelinka@fai.utb.cz

Hendrik Richter
HTWK Leipzig, Fakultät Elektrotechnik und Informationstechnik,
D–04251 Leipzig, Germany
e-mail: richter@fbeit.htwk-leipzig.de

allow to glimpse at the people who did the pioneering work and established the field. Maybe the two most significant persons whose research on evolution and genetics had the biggest impact on modern understanding of evolution and its use for computational purposes are Gregor Johann Mendel and Charles Darwin.

Gregor Johann Mendel (Fig. 2.1, July 20, 1822 - January 6, 1884) was an Augustinian priest and scientist, and is often called the father of genetics for his study of the inheritance of certain traits in pea plants. He was born in the family of farmers in Hyncice (Heinzendorf bei Odrau) in Bohemia (that time part of Austrian - Hungary empire, today Czech Republic). The most significant contribution of Mendel for science was his discovery of genetic laws which showed that the inheritance of these traits follows particular laws (published in [52]), which were later named after him. All his discoveries were done in Abbey of St. Thomas in Brno (Bohemia). Mendel published his research at two meetings of the Natural History Society of Brünn in Moravia (east part of Bohemia) in 1865 [52]. When Mendel's paper was published in 1866 in Proceedings of the Natural History Society of Brünn, it had little impact and was cited only about three times over the next thirty-five years. His paper was criticized at the time, but is now considered a seminal work. The significance of Mendel's work was not recognized until the turn of the 20th century. Its rediscovery (thanks to Hugo de Vries, Carl Correns and Erich von Tschermak) prompted the foundation of the discipline of genetics. Very peculiar historical fact about Mendel's research is also that his letters about his discovery, sent to many of scientific societies, had been found after many years in their libraries unopened. Mendel died on January 6, 1884, at age 61, soon after his death the succeeding abbot burned all papers in Mendel's collection, to mark an end to the disputes over taxation [10].

The other important (and much more well–known and therefore here only briefly introduced) researcher whose discoveries founded the theory of evolution was the British scientists Charles Darwin. Darwin (Fig. 2.2) published in his work [17] the

Fig. 2.1 Gregor Johann Mendel (July 20, 1822 - January 6, 1884)

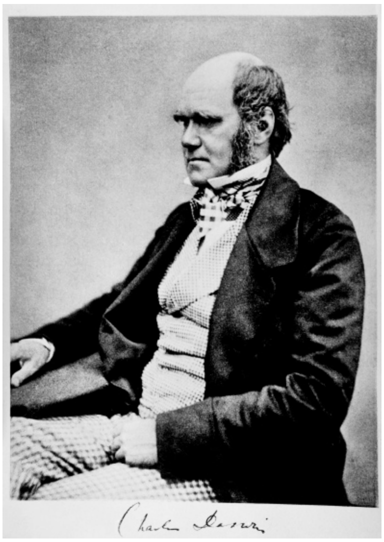


Fig. 2.2 Gregor Charles Darwin 12 February 1809 - 19 April 1882

main ideas of the evolutionary theory. The full and original title was *"On the Origin of Species by Means of Natural Selection, or the Preservation of Favored Races in the Struggle for Life"*. Word *"races"* refers here to biological varieties. The title has been changed to [17] for the 6th edition of 1872. In Darwin's book On the Origin of Species (1859) established evolutionary descent with modification as the dominant scientific explanation of diversification in the nature.

The above mentioned ideas of genetics and evolution have been formulated long before the first computer experiments with evolutionary principles had been done. The beginning of the ECT is officially dated to the 70s of the 20th century, when famous genetic algorithms were introduced by J. Holland [37, 38] or to the late 60s with evolutionary strategies, introduced by Schwefel [64] and Rechenberg [60] and evolutionary programming by L.J. Fogel [29]. However, when certain historical facts are taken into consideration, then one can see that the main principles and ideas of ECT as well as its computer simulations had been done earlier than mentioned above. Conceptionally, ECT can be traced back to the famous A.M. Turing, first numerical experiments to the (far less famous) N.A. Barricelli and others. Their understanding and formulation of basic ideas of ECT was remarkably clear, see e.g. Turing in his essay "Intelligent machinery" (1948) [67] where he say:

"...if the untrained infant's mind is to become an intelligent one, it must acquire both discipline and initiative... discipline is certainly not enough in itself to produce intelligence. That which is required in addition we call initiative...our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines...".

In other words he speaks about simulation of an intelligent creature. Turing continues in his text by

“...further research into intelligence of machinery will probably be very greatly concerned with ’searches’...”,

and suggested that *’searches’* will be done probably in the space of numbers and basically he describes the central idea of ECT by

“...there is the genetic or evolutionary search by which a combination of genes is looked for, the criterion being the survival value”.

Turing further improved this idea in his article “Computing Machinery and Intelligence” (1950) [67]:

“We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications”

- *structure of child machine = hereditary material*
- *changes of the child machine = mutations*
- *natural selection = judgement of the experimenter”.*

One of the first to transform Turing’s ideas into real computer numerical experiments was N. Barricelli (1954) [4, 5]. Results were published in the journal *”Methods”* with the title *”Esempi Numerici di processi di evoluzione”* and consequently repeated and improved in 1962 [6] when ECT numerical experiment with 500 of 8 bits strings had been successfully done. Based on this numerical simulations Barricelli reported that:

“we have created a class of numbers which are able to reproduce and to undergo hereditary changes...the constitution for an evolutionary process according to the principle of Darwins theory would appear to be present. The numbers which have the greatest survival in their environment will survive. The other numbers will be eliminated little by little. A process of adaptation to the environmental conditions, that is, a process of Darwinian evolution, will take place”.

Barricelli’s early computer-assisted experiments, which were focused on symbiogenesis and evolution (based on Darwin’s ideas), can be accepted like pioneering experiments in artificial life research. Barricelli has been working in Institute for Advanced Study in Princeton, New Jersey in 1953, 1954 and 1956. Later he worked at the University of California, Los Angeles, at Vanderbilt University, in the Department of Genetics of the University of Washington, Seattle and then at the Mathematics Institute of the University of Oslo. He is also author of a variety of articles in fields as different as theoretical physics and mathematical language, virus genetics, DNA, theoretical biology, space flight, etc. Barricelli’s experiments are probably some of the first historically recorded numerical ECT experiments. Another interesting tread of ECT's pre-history are the works of Box (1957) [7] and Friedberg (1958) [30]. Although the original papers are meanwhile hardly accessible, it is in particular thanks to D.B. Fogel (son of evolutionary programming pioneer L.J. Fogel), who edited some of these works [27] and recollected some technical details and implications [26, 28], that this early history of ECT can now be rediscovered. However, in some respect all these works were slightly ahead of time, as the results have clearly shown the potential of ECT methods, but the lack of computing power at that time prevented to solve "real problems" and hence to widespread the

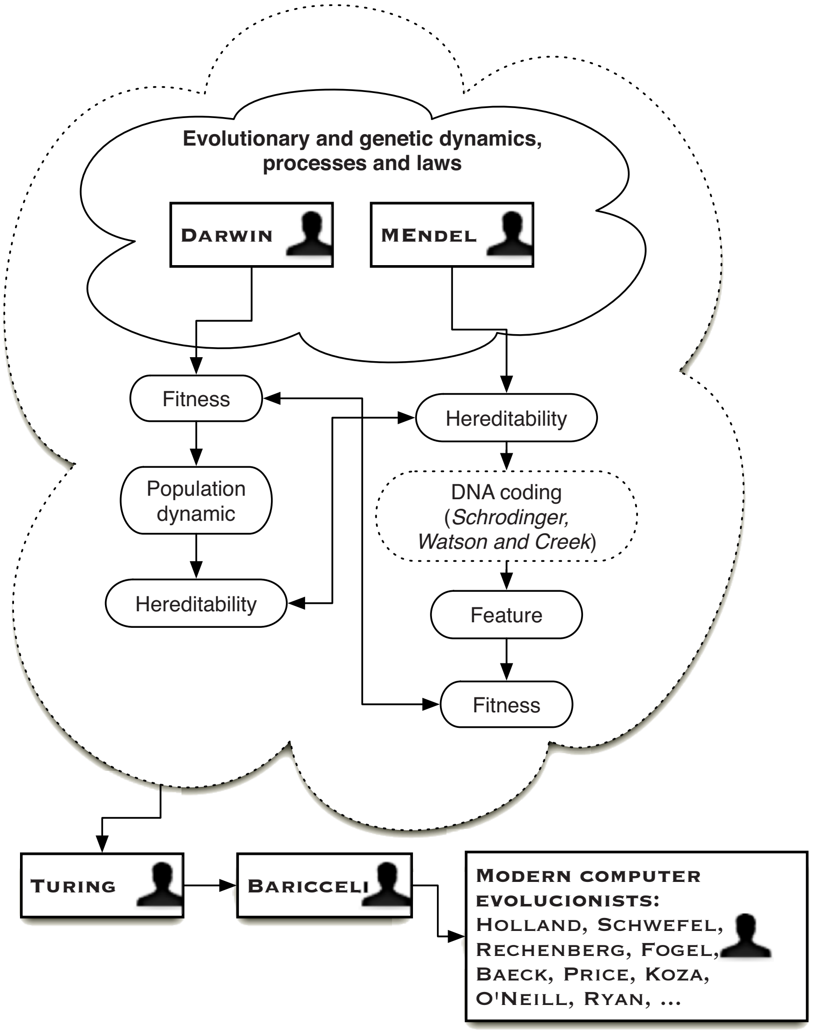

Evolutionary and genetic dynamics, processes and laws
DARWIN
MENDEL
Fitness
Hereditability
Population dynamic
DNA coding (Schrodinger, Watson and Creek)
Hereditability
Feature
Fitness
TURING
BARICCELI
MODERN COMPUTER EVOLUCIONISTS: HOLLAND, SCHWEFEL, RECHENBERG, FOGEL, BAECK, PRICE, KOZA, O'NEILL, RYAN, ...

Fig. 2.3 Evolutionary theory and its historical relations

methods. So, the “golden era” of ECT began, when genetic algorithms by J. Holland [37], evolutionary strategies, by Schwefel [64] and Rechenberg [60] and evolutionary programming by Fogel [29] had been introduced. All these designs were favored by the upcoming of more powerful and more easily programmable computers so that for the first time interesting problems could be tackled and ECT started to compete with and became a serious alternative to other optimization methods. Since that time other successful algorithms using ECT ideas have been developed, for instance scatter search, particle swarm, memetic algorithms, differential evolution, ant colony optimization, and many others. Before their brief description it is important to outline the main principle (lets call it *central dogma*) of evolutionary computation in general. History of ECT is of course more rich and complex, than described here. Main ideas and relations can be more clearly visible from Fig. 2.3.

2.2 Evolutionary Algorithms – Outline

In recent years, a broad class of algorithms has been developed for stochastic optimization, i.e. for optimizing systems where the functional relationship between the independent input variables and the output (objective function) of a system is not explicitly known. Using stochastic optimization algorithms such as Genetic Algorithms, Simulated Annealing and Differential Evolution, a system is confronted with a random input vector and its response is measured. This response is then used by the algorithm to tune the input vector in such a way that the system produces the desired output or target value in an iterative process. Most engineering problems can be defined as optimization problems, e.g. the finding of an optimal trajectory for a robot arm, the optimal thickness of steel in pressure vessels, the optimal set of parameters for controllers, optimal relations or fuzzy sets in fuzzy models, etc. Solutions to such problems are usually difficult to find as their parameters usually include variables of different types, such as floating point or integer variables. Evolutionary algorithms, such as the Genetic Algorithms, Particle Swarm, Ant Colony Optimization, Scatter Search, Differential Evolution etc., have been successfully used in the past for these engineering problems, because they can offer solutions to almost any problem in a simplified manner: they are able to handle optimizing tasks with mixed variables, including the appropriate constraints, and they do not rely on the existence of derivatives or auxiliary information about the system, e.g. its transfer function. This chapter is concerned with a brief introduction on so-called evolutionary computational techniques (ECT). Although the editors of this book assume that most of the readers will have at least basic knowledge of ECT, there might be the wish for clarification and broadening. For this reason, this chapter was also included to describe in simple terms what ECT actually means.

2.2.1 Central Dogma of Evolutionary Computational Techniques

The evolutionary computational techniques are numerical algorithms that are based on the basic principles of Darwin’s theory of evolution and Mendel’s foundation of genetics. The main idea is that every individual of a species can be characterized by its features and abilities that help it to cope with its environment in terms of survival and reproduction. These features and abilities can be termed its fitness and are inheritable via its genome. In the genome the features/abilities are encoded. The code in the genome can be viewed as a kind of "blue–print" that allows to store, process and transmit the information needed to build the individual. So, the fitness coded in the parent's genome can be handed over to new descendants and support the descendants in performing in the environment. Darwinian participation to this basic idea is the connection between fitness, population dynamics and inheritability while the Mendelian input is the relationship between inheritability, feature/ability and fitness. Both views have been brought together in molecular Darwinism with the idea of a genetic code (first uttered in its full information–theoretical meaning by Erwin Schrödinger in his 1944 book *What is life?*) and the discovery of the structure of the DNA and its implications to genetic coding by James Watson and Francis Crick in 1953.

By these principles and in connection with the occurrence of mutations that modify the genome and hence may produce inheritable traits that enhance fitness, a development of individuals and species towards best adaption to the environment takes place. Here, the multitude of individuals in a species serve two connected evolutionary aspects, (i.) provide the opportunity to "collect" mutations and pass traits that are or are not fitness enhancing to descendants on an individual level and (ii.) allow fitness enhancing genomes to spread in the species from one generation to the next if the traits bring advantages in survival and reproduction.

However, it should be noted that Darwin or, as the case may be, Mendel, were not the first. Already in the ancient era, there were thinkers who came with the same idea as Darwin and Mendel. An outstanding thinker, who supported the idea of evolution before Darwin, was Anaximander, a citizen from Miletus, an Ionian city of Asia Minor. Anaximander's philosophical ideas are summarized in his philosophical tract *"On nature"*, however, this name is of a later date, because this book was not preserved. According to Anaximander, the original principle of the world and the cause of all being is "without a limit" (*"apeiron"* in Greek), from which cold and warm and dry and wet is separated - essentially, one can imagine this principle in the sense of unlimited and undifferentiated wetness, from which all other natural substances and individual species of living creatures arise.

By his idea that the Earth, which he imagined as freely floating in space, was initially in a liquid state and later, when it was drying, gradually gave rise to animals, who at first lived in water and later migrated onto land, Anaximander in part anticipates the modern theory of evolution.

The ECT technology stands or falls with the existence of the so-called evolutionary algorithms (EA) that in principle form the majority of ECT. Besides evolutionary algorithms, there still exist other extensions, such as genetic programming, evolutionary hardware, etc. With respect to the fact that evolutionary algorithms are the backbone of ECT, attention will be paid in this chapter just to these algorithms, whose understanding is **absolutely necessary** for understanding the rest of this publication.

From the above mentioned main ideas of Darwin and Mendel theory of evolution, ECT uses some building blocks which the diagram in Fig. 2.4 illustrates. The evolutionary principles are transferred into computational methods in a simplified form that will be outlined now.

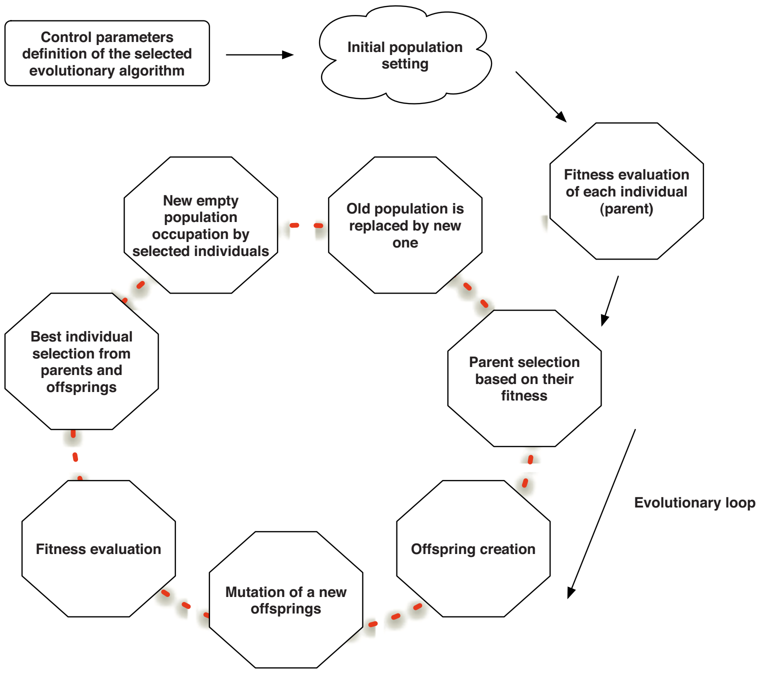

Control parameters definition of the selected evolutionary algorithm
Initial population setting
Fitness evaluation of each individual (parent)
New empty population occupation by selected individuals
Old population is replaced by new one
Best individual selection from parents and offsprings
Parent selection based on their fitness
Evolutionary loop
Fitness evaluation
Offspring creation
Mutation of a new offsprings

Fig. 2.4 General cycle of the evolutionary algorithm. The termination of the evolution after *n* generations and the selection of the best individual are not indicated in this figure - solution from the last population.

If the evolutionary principles are used for the purposes of complicated calculations (in accordance with Fig. 2.4), the following procedure is used:

1. Specification of the evolutionary parameters: For each algorithm, parameters must be defined that control the run of the algorithm or terminate it regularly, if the termination criterions defined in advance are fulfilled (for example, the number of cycles - generations). Part of this point is the definition of the cost function (objective function) or, as the case may be, what is called fitness - a modified return value of the objective function). The objective function is usually a mathematical model of the problem, whose minimization or maximization (generally therefore extremization) leads to the solution of the problem.

This function with possible limiting conditions is some kind of “environmental equivalent” in which the quality of current individuals is assessed.

2. Generation of the initial population (generally $N \times M$ matrix, where N is the number of parameters of an individual - D is used hereinafter in this publication - and M is the number of individuals in the population): Depending on the number of optimized arguments of the objective function and the user’s criterions, the initial population of individuals is generated. An individual is a vector of numbers having such a number of components as the number of optimized parameters of the objective function. These components are set randomly and each individual thus represents one possible specific solution of the problem. The set of individuals is called population.
3. All the individuals are evaluated through a defined objective function and to each of them is assigned a) Either a direct value of the return objective function, or b) A fitness value, which is a modified (usually normalized) value of the objective function.
4. Now parents are selected according to their quality (fitness, value of the objective function) or, as the case may be, also according to other criterions.
5. Descendants are created by crossbreeding the parents. The process of crossbreeding is different for each algorithm. Parts of parents are changed in classic genetic algorithms, in a differential evolution, crossbreeding is a certain vector operation, etc.
6. Every descendant is mutated. In other words, a new individual is changed by means of a suitable random process. This step is equivalent to the biological mutation of the genes of an individual.
7. Every new individual is evaluated in the same manner as in step 3.
8. The best individuals are selected.
9. The selected individuals fill a new population.
10. The old population is forgotten (eliminated, deleted, dies,..) and is replaced by a new population; step 4 represents further continuation.

Steps 4 - 10 are repeated until the number of evolution cycles specified before by the user is reached or if the required quality of the solution is not achieved. The principle of the evolutionary algorithm outlined above is general and may more or less differ in specific cases. So, methods that work by an algorithmic structure as outlined in the steps 1-10 share the following main evolutionary principles:

- **Biological inspiration:** The algorithms mimic and use in an abstracted way working mechanisms of biological systems.
- **Population–based calculations:** By structuring data in the algorithm by the individual–and–species model, individual search is coordinated to other individuals and so to the whole population. This has the effect of parallelism in the search which is assumed to be the main reason for success of evolutionary search.
- **Repeated calculation of fitness for all individuals:** This principles provides a spectrum of fitness to the population from which search can be guided by noticing and discriminating individuals of different fitness.

- **Generational search:** Repeated generational search guided by the fitness spectra allows to accumulate individuals with high fitness.
- **Stochastic and deterministic driving forces:** Random influences, for instance in form of mutations are balanced by the deterministic elements in the flow of the algorithm.
- **Coordination between individuals:** Some kind of communication on (or even in–between) the individuals of the population (e.g. in the selection (crossover) or recombination process) allows to recognize and exploit individual differences in fitness.

There are also exemptions that do not adhere to steps 1 - 10; in such a case, the corresponding algorithms are not denoted as evolutionary algorithms, but usually as algorithms that belong to ECT. Some evolutionists exclude them completely from the ECT class. The ACO algorithm (*Ant Colony Optimization*), see [21] and [56] may be an example - it simulates the behavior of an ant colony and can solve extremely complicated combinatory problems. It is based on the principles of cooperation of several individuals belonging to the same colony - in this case ants.

The evolution diagrams are not only popular because they are modern and differ from classical algorithms, but mainly because of the fact that they are able to replace a man in the event of a suitable application. This is illustrated in Fig. 2.5. There are two methods of the problem solution illustrated in this figure. The first one represents steps of a human investigator, the second one represents the procedure if ECT is used.

This publication thus deals with ECT's that in most cases adhere to the above indicated evolutionary scheme; nevertheless, exemptions are also indicated.

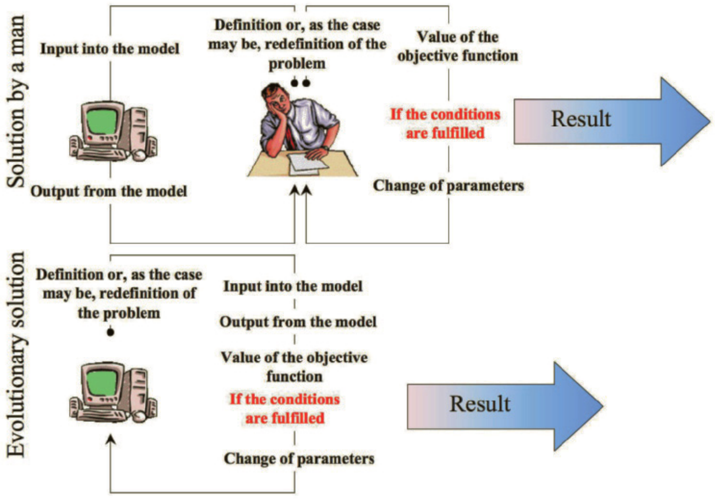

Solution by a man
Input into the model
Definition or, as the case may be, redefinition of the problem
Value of the objective function
If the conditions are fulfilled
Result
Output from the model
Change of parameters
Evolutionary solution
Definition or, as the case may be, redefinition of the problem
Input into the model
Output from the model
Value of the objective function
If the conditions are fulfilled
Result
Change of parameters

Fig. 2.5 Comparison of the problem solution by means of ECT and a man. Simplified illustration.

2.2.2 Evolutionary Algorithms and Importance of Their Use

Comparing to standard optimization techniques, evolutionary algorithms can be used on *almost* arbitrary optimization problem, however, it is important to remember that with different performance. As mentioned in the section 2.5.2, there are problems with different level of complexity, from the simplest (solvable by standard techniques) to the most complex, whose solution would take much more longer time, than our universe exist. Thus, some simple problems, that can be very easily and quickly solved by gradient based techniques, should not be solved by heuristic methods, because its use would be expensive, i.e. user would "pay" by big number of cost function evaluations. Another important fact, having impact on EA use is so called No Free Lunch Theorem (NFLT), see [70]. Main idea of this theorem is that there is no ideal algorithm which would be able to solve **any** problem. Simply, if there are for example two algorithms **A** and **B**, then for certain subset of possible problems is more suitable algorithms **A** and for another subset algorithm **B**. All those subsets can be of course totally disconnected, or/and overlapped.

Based on those facts it is important to remember that evolutionary algorithms are suitable for problems which are more complex rather simple, and also that their selection and setting depend on user experiences, expertise etc. More exact classification is mentioned in the section 2.4.

2.3 Selected Evolutionary Techniques

2.3.1 Overview

Optimization algorithms are a powerful tool for solving many problems of engineering applications. They are usually used where the solution of a given problem analytically is unsuitable or unrealistic. If implemented in a suitable manner, there is no need for frequent user intervention into the actions of equipment in which they are used.

The majority of the problems of engineering applications can be defined as optimization problems, for example, finding the optimum trajectory of a robot or the optimum thickness of the wall of a pressure tank or the optimum setting of the regulator's parameters. In other words, the problem solved can be transformed into a mathematical problem defined by a suitable prescription, whose optimization leads to finding the arguments of the objective function, which is its goal.

Countless examples can be found illustrating this problem. The solution of such problems usually requires working with the arguments of optimized functions, where the definition ranges of these arguments may be of a heterogeneous character, such as, for example, the range of integers, real or complex numbers, etc. Moreover, it may happen (depending on the case) that for certain subintervals from the permitted interval of values, the corresponding argument of the optimized function may assume values of various types (integers, real, complex,..). Besides this, various penalizations and restrictions can play a role within optimization, not only for given arguments, but also for the functional value of the optimized function. In many cases, the analytical solution of such an optimization problem is possible, nevertheless, considerably complicated and tedious.

A class of very efficient algorithms has been developed for the successful solution of such problems in the past two decades that make it possible to solve very complicated problems efficiently. The algorithms of this class have their specific name, namely "evolutionary algorithms". They solve problems in such an elegant manner that they became very popular and are used in many engineering fields.

From the point of view of the most general classification, the evolutionary algorithms belong to heuristic algorithms. Heuristic algorithms are either *deterministic* or *stochastic*. The algorithms of the second group differ in that their certain steps use random operations, which means that the results of the solutions obtained with their use may differ in the individual runs of the program. It is therefore meaningful to run the program several times and select the best solution obtained.

Stochastic heuristic methods are sometimes called *metaheuristics*, because they only provide a general framework and the algorithms of the operation itself must be chosen (for example, by the operation of crossbreeding and mutation in genetic algorithms, operation of neighborhood in simulated annealing, "tabu search", etc.) in dependence on the problem investigated. Because these methods are frequently inspired by natural processes, they are also called evolutionary algorithms. Depending on their strategy, they can be divided in two classes:

1. Methods based on point-based strategy such as, for example, *simulated annealing* ([41], [13], [66]), *hill-climbing algorithm* [63] and *tabu-search* [31]. These algorithms are based on the *neighborhood* operation of the current solution, in which we are looking for a better solution.
2. *Population-based strategy. Genetic algorithms* [19], [33], [53], [14] are based on *population strategy*.

These methods differ from classic gradient methods by admitting (with a certain probability) a worse solution into the next iteration; in this manner, they try to avoid local minima. For more details, see, for example, books [31], [33],[53],[54] and [61].

2.3.2 Current State

Evolutionary algorithms serve for finding the minima (or maxima) of a given objective function by looking for the optimum numerical combination of its arguments. These algorithms can be divided according to the principles of their action, complexity of the algorithm, etc. Of course, this classification is not the only possibility, nevertheless, because it fits the current state rather well, it can be considered as one of the possible views on the classical and modern optimization methods. There are slight differences in opinions on their classification. One can encounter statements that, for example, simulated annealing does not belong to evolutionary techniques, which is true to some extent. On the other side, other "evolutionists" state that simulated annealing does belong to evolutionary techniques, at least as their direct predecessor. It is true that if simulated annealing is taken into account with elitism, then one could consider this alternative as an evolutionary algorithm.

2.3.2.1 Classes Optimization Approaches

Figs. 2.6 - 2.8 illustrate various views on the classification of evolutionary algorithms that exhibit certain differences although they have a visible common line. These differences may be caused not only by the classification of the algorithm according to the principles by which it is controlled, but, for example, according to the classes of problems for which it is “predestined”. The individual classes of algorithms represent generally solutions of a given problem by the methods of various degrees of efficiency and complexity. Depending on their properties, we classify algorithms into the following categories:

Enumerative: The algorithm calculates all possible combinations of a given problem. This approach is suitable for problems where the arguments of the objective function have a discrete character and assume a small number of values. Should it be applied generally, it might need more time for its successful termination than is the time of the existence of the universe.

Deterministic: This group of algorithms is based only on the rigorous methods of classical mathematics. The algorithms of this character usually require limiting assumptions that enable these methods to provide efficient results. Usually, these assumptions are as follows:

- The problem is linear.
- The problem is convex.
- The space of the possible solutions is small.
- The space of the possible solutions is continuous.
- The objective function is unimodal (it has only one extreme).
- There are no nonlinear interactions between the parameters of the objective function.
- Information on the gradient is available, etc.
- The problem is defined in an analytical form.

The result of the deterministic algorithm is then only one solution.

Stochastic: The algorithms of this type are based on the use of chance. This is essentially a purely random search of the values of the objective function; the result is always the best solution that was found during the entire random search. The algorithms of this type are usually;

- slow.
- suitable only for small spaces of possible solutions (small range of arguments of the objective function),
- suitable for a rough estimation.

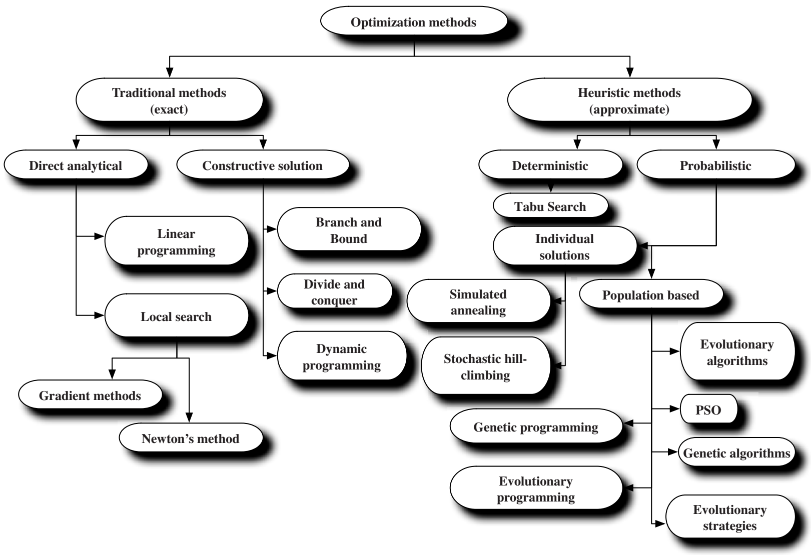


Fig. 2.6 Classification of the optimization methods according to [68]

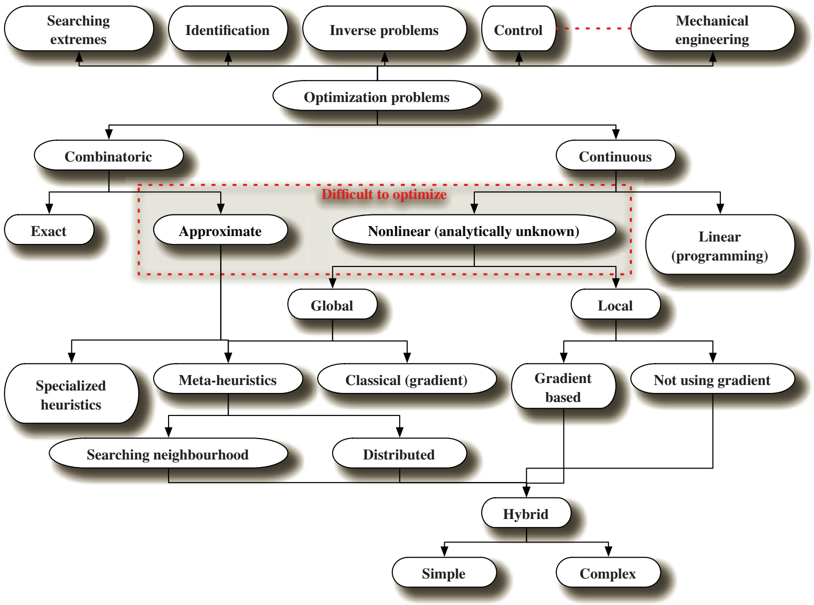


Fig. 2.7 Classification of the optimization methods according to [22]

Mixed: The algorithms of this class represent a “sophisticated” mixture of deterministic and stochastic methods that achieve surprisingly good results in mutual cooperation. The evolutionary algorithms mentioned above are a

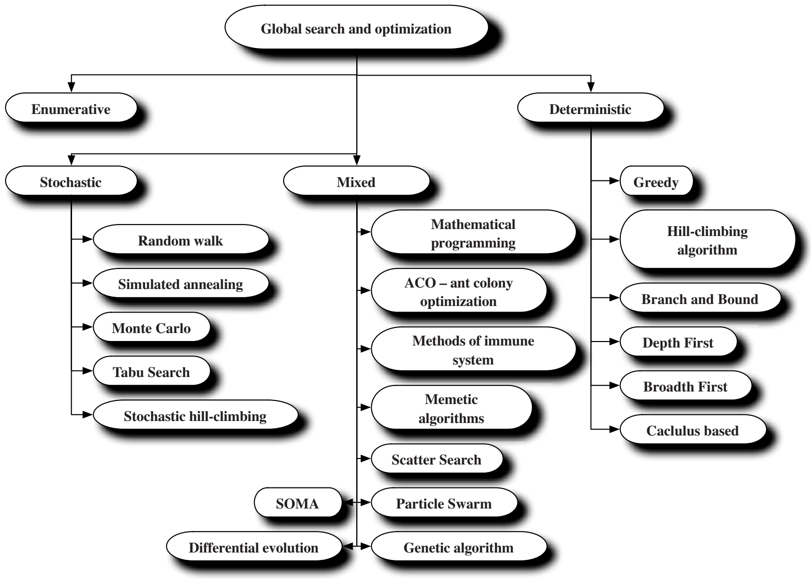


Fig. 2.8 Other possible organization of optimization algorithms [72]

relatively strong sub-set of these algorithms. The algorithms of the mixed character are:

- Robust, which means that they very frequently find a quality solution independently of the initial conditions; this solution is usually represented by one or several global extremes.
- Efficient and powerful. The terms "efficient and powerful" here mean that they are able to find a quality solution during a relatively small number of evaluations of the objective function.
- Differ from purely stochastic methods (thanks to the presence of deterministic approaches).
- Have minimum or no requirements for preliminary information.
- They are able to work with problems of the "black box" type, i.e., they do not need an analytical description of the problem for their activity.
- Are able to find several solutions during one run.

We can briefly summarize these features as follows:

- The enumerative and stochastic optimization is not suitable for problems where an extensive space of possible solutions must be searched.
- The deterministic optimization works well with problems where the space of possible solutions is not too extensive.
- Mixed optimization is suitable for problems without limitations to the size of the space of possible solutions.

In this book, selected algorithms are described with emphasis put on their explanation and testing. The evolutionary algorithms are now very popular thanks to properties that are characteristic for the entire class. However, before we start to discuss their details, it is suitable to mention both the generally known algorithms and the newer algorithms in this field, which will be developed in more detail later in this publication. From the well known algorithms, we have selected a few random–driven algorithms:

- **Stochastic Hill Climbing**
- **Tabu search**

the "classic" evolutionary algorithm

- **Genetic algorithms**
- **Genetic programming**
- **Evolutionary strategy**
- **Evolutionary programming**

and from the newest ones, we will describe

- **Learning classifier systems**
- **Population-based incremental learning**
- **Ant Colony Optimization**
- **Immunology System Method**
- **Memetic Algorithms**
- **Scatter Search**
- **Particle Swarm**
- **Differential Evolution**
- **SOMA**

2.3.2.2 The Outline of the Principles of Action of Random–Driven Search Algorithms

At the present time, there is a broad spectrum of publications dealing with optimization algorithms, for example, [2]. The purpose of this chapter is to outline only the principles of some selected algorithms for better information for the reader. The discussed algorithms are:

Stochastic Hill-Climbing: (SHC) is a version of the hill-climbing mechanism enriched by the stochastic component [40], [63]. It belongs among the gradient methods, which means that it searches the space of possible solutions in the direction of the steepest gradient. Thanks to its gradient nature, it frequently gets stuck in a local extreme. The basic version functions so that it always starts from the random point in the space of possible solutions. For the momentary proposed solution, the certain neighborhood is proposed by means of a final set of transformations (Figure 2.9) and the given function is minimized only in this neighborhood. The local solution obtained is then used as a start point for the calculation of the new neighborhood. The entire process is then repeated iteratively. The best found

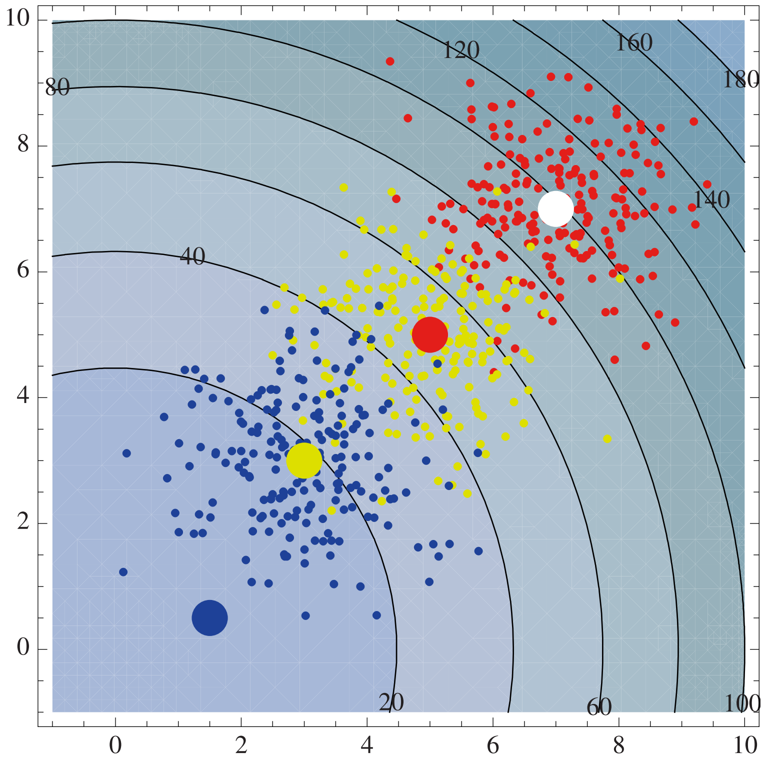


Fig. 2.9 Principle of the hill-climbing algorithm. The white dot is the starting point of the algorithm. The set of red solutions is generated for this dot. The best solution is denoted by the bold red dot that serves also as a new position for the generation of a new (yellow) set of solutions. The best solution (bold yellow dot) is then used for the generation of the following set of solutions (blue).

solution is recorded during the process that serves as the optimum found after termination. The stochastic hill-climbing algorithm is basically only the multiple repetition of the standard hill-climbing algorithm, each time from another randomly selected position. The disadvantage of this algorithm is that runaway may occur in it under certain conditions and the solution gets stuck in a local extreme [40].

Tabu Search: (TS) (prohibited search,[31]) is the improved version of the hill-climbing algorithm. This algorithm was created by Professor Fred Glover from the University of Colorado. The improvement consists in introducing a "short-term memory" into the hill-climbing algorithm, whose task is to remember those transformations by means of which the current median was calculated. The final consequence is that runaway cannot occur due to the forbidden use of these transformations. The name "Tabu Search" originates from this feature. This method was improved by a "long term memory" that includes transformations, which are not short term in the memory, but have been used frequently. Their use is then penalized, which decreases the frequency of their use. Contrary to the hill-climbing algorithm, Tabu Search does not get stuck so easily in local extremes.

2.3.2.3 The Outline of the Principles of Action of Evolutionary Algorithms

Genetic algorithm: (GA) This algorithm is one of the first successful applied ECT methods [37, 33]. In GAs the main principles of ECT are applied in their purest form. The individuals are encoded as binary strings (mostly over the alphabet $[0,1]$), which can be understood as a model of the biological counterpart, the genome,[1] and represent possible solutions to the optimization problem under study. After initially a population of binary strings is created randomly, the circle as given in Figure 2.4 is carried out with the steps fitness evaluation, selection, offspring generation (crossover) and mutation until the algorithm terminates. The application area of these algorithms are wide and it seem particularly sensible to use them if the problem description allows a straightforward coding of the objects to optimize as binary string over a finite alphabet, for instance in combinatorial optimization problem timetabling and scheduling.

Evolutionary strategy: (ES) This algorithm also belongs to the first successful stochastic algorithms in history. It was proposed at the beginning of the sixties by Rechenberg [60] and Schwefel [64]. It is based on the principles of natural selection similarly as the genetic algorithms. Contrary to genetic algorithms, the evolutionary strategy works directly with individuals described by vectors of real values. Its core is to use candidate solutions in the form of vectors of real numbers, which are recombined and then mutated with the help of a vector of random numbers. The problem of accepting a new solution is strictly deterministic. Another distinctive feature is that ES use self-adaptation, that is the mutation strength for each individual is variable over the generational run and subject to an own evolutionary adaption and optimization process.

Evolutionary programming: (EP) EP algorithms [29] have much similarity to ES in using vectors of real numbers as representation. The main operator in the generational circle is mutation, in most (particularly early) implementations no recombination is carried out. In recent years, by adopting elements of their algorithmic structure EP more and more tends to become similar to ES.

Learning classifier systems: (LCS) LCS [9] are machine learning algorithms which are based on GAs and reinforcement learning techniques. Interestingly, LCS were introduced by Holland[2] [37] and for a certain time

[1] The genome is coded over the alphabet $[A,C,G,T]$, which stand for the amino acids adenine A, cytosine C, guanine G, thymine T.

[2] Holland is also know as the father of GAs.

regarded as a generalization of GAs. LCS optimize over a set of rules that are intended to best–fit inputs to outputs. The rules are coded binary and undergo an adaption using GA–like optimization that modifies and selects the best rules. The fitting of the rules is determined by reinforcement learning methods.

Population-based incremental learning: (PBIL) PBIL was proposed by Baluja [3] and combines ideas from evolutionary computation with methods from statistical learning [49]. It uses a real valued representation that is usually restricted to the interval $[0,1]$ and can be interpreted as the probability to have a "1" - bit at a certain place in a binary string. From these probabilities, a collection of binary strings is created. These strings are subjected to a standard evolutionary circle with fitness evaluation, selection and discarding of inferior samples. In addition, based on the evaluation of the fitness, a deterministic statistical-learning-like updating of the probability vector takes place, which afterwards is also altered by random mutation.

Ant Colony Optimization: (ACO), [21] This is an algorithm whose action simulates the behavior of ants in a colony. It is based on the following principle. Let there be a source of ants (colony) and the goal of their activity (food), see Fig. 2.10. When they are released, all the ants move after some time along the shorter (optimum) route between the source and goal. The effect of finding the optimum route is given by the fact that the ants mark the route with pheromones. If an ant arrives to the crossroads of two routes that lead to the same goal, his decision along which route to go is random. Those ants that found food start marking the route and when returning, their decision is influenced thanks to these marks in favor of this route. When returning, they mark it for the second time, which increases the probability of the decision of further ants in its favor. These principles are used in the ACO algorithm. Pheromone is here represented by the weight that is assigned to a given route leading to the goal. This weight is additive, which makes it possible to add further "pheromones" from other ants. The evaporation of pheromones is also taken into account in the ACO algorithm in such a way that the weights fade away with time at individual joints. This increases the robust character of the algorithm from the point of view of finding the global extreme. ACO was successfully used to solve optimization problems such as the travelling salesman problem or the design of telecommunication networks, see [56].

Immunology System Method: (ISM) This algorithm is unusual by its algorithm based on the principles of functioning of the immunology system in living organisms. As indicated in [56], there are several principles based on this model. In this work, the immunology system is considered as a multivalent system, where individual agents have their specific tasks. These agents have various competencies and ability to communicate with other agents. On the basis of this communication and a certain "freedom" in making decisions of individual agents, a hierarchic structure is formed able to solve complicated problems. As an example of using this method,

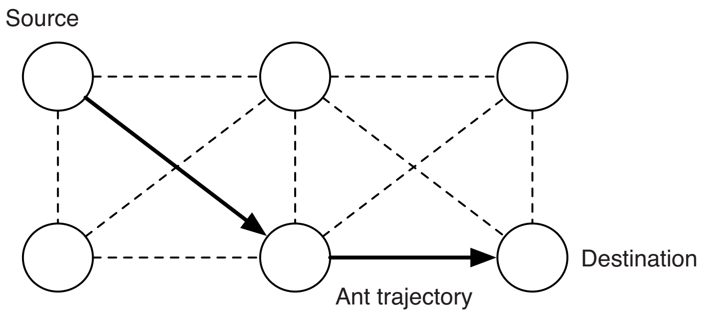


Fig. 2.10 Principle of the ACO algorithm

antivirus protection can be mentioned in large and extensive computer systems [18], [12].

Memetic Algorithms: (MA) This term represents a broad class of metaheuristic algorithms [56], [35], [32], [65]. The key characteristics of these algorithms are the use of various approximation algorithms, local search techniques, special recombination operators, etc. These metaheuristic algorithms can be basically characterized as competitive-cooperative strategies featuring attributes of synergy. As an example of memetic algorithms, hybrid combinations of genetic algorithms and simulated annealing or a parallel local search can be indicated. Memetic algorithms were successfully used for solving such problems as the traveling salesman problem, learning of a neural multilayer network, maintenance planning, nonlinear integer number programming and others (references see [56].

Scatter Search: (SS) This optimization algorithm differs by its nature from the standard evolutionary diagrams. It is a vector oriented algorithm that generates new vectors (solutions) on the basis of auxiliary heuristic techniques. It starts from the solutions obtained by means of a suitable heuristic technique. New solutions are then generated on the basis of a subset of the best solutions obtained from the start. A set of the best solutions is then selected from these newly found solutions and the entire process is repeated. This algorithm was used for the solution of traffic problems, such as traffic control, learning neural network, optimization without limits and many other problems [56], [45].

Particle Swarm: (PSO) The "particle swarm" algorithm is based on work with the population of individuals, whose position in the space of possible solutions is changed by means of the so-called velocity vector. According to the description in [56], [71] and [15], there is no mutual interaction between individuals in the basic version. This is removed in the version with the so-called neighborhood. In the framework of this neighborhood, mutual interaction occurs in such a manner that individuals belonging to one neighborhood migrate to the deepest extreme that was found in this neighborhood.

Differential Evolution: (DE) Differential Evolution [57] is a population-based optimization method that works on real-number coded individuals. For each individual $\overrightarrow{x}_{i,G}$ in the current generation G, DE generates a new trial individual $\overrightarrow{x'}_{i,G}$ by adding the weighted difference between two randomly selected individuals $\overrightarrow{x}_{r1,G}$ and $\overrightarrow{x}_{r2,G}$ to a third randomly selected individual $\overrightarrow{x}_{r3,G}$. The resulting individual $\overrightarrow{x'}_{i,G}$ is crossed-over with the original individual $\overrightarrow{x}_{i,G}$. The fitness of the resulting individual, referred to as perturbated vector $\overrightarrow{u}_{i,G+1}$, is then compared with the fitness of $\overrightarrow{x}_{i,G}$. If the fitness of $\overrightarrow{u}_{i,G+1}$ is greater than the fitness of $\overrightarrow{x}_{i,G}$, $\overrightarrow{x}_{i,G}$ is replaced with $\overrightarrow{u}_{i,G+1}$, otherwise $\overrightarrow{x}_{i,G}$ remains in the population as $\overrightarrow{x}_{i,G+1}$. Differential Evolution is robust, fast, and effective with global optimization ability. It does not require that the objective function is differentiable , and it works with noisy, epistatic and time-dependent objective functions.

SOMA: (Self-Organizing Migrating Algorithm) is a stochastic optimization algorithm that is modeled on the social behavior of cooperating individuals [73]. It was chosen because it has been proven that the algorithm has the ability to converge towards the global optimum [73]. SOMA works on a population of candidate solutions in loops called *migration loops*. The population is initialized randomly distributed over the search space at the beginning of the search. In each loop, the population is evaluated and the solution with the highest fitness becomes the leader L. Apart from the leader, in one migration loop, all individuals will traverse the input space in the direction of the leader. Mutation, the random perturbation of individuals, is an important operation for evolutionary strategies (ES). It ensures the diversity amongst the individuals and it also provides the means to restore lost information in a population. Mutation is different in SOMA compared with other ES strategies. SOMA uses a parameter called PRT to achieve perturbation. This parameter has the same effect for SOMA as mutation has for GA. The novelty of this approach is that the PRT Vector is created before an individual starts its journey over the search space. The PRT Vector defines the final movement of an active individual in search space. The randomly generated binary perturbation vector controls the allowed dimensions for an individual. If an element of the perturbation vector is set to zero, then the individual is not allowed to change its position in the corresponding dimension. An individual will travel a certain distance (called the path length) towards the leader in n steps of defined length. If the path length is chosen to be greater than one, then the individual will overshoot the leader. This path is perturbed randomly.

The evolutionary algorithms can be essentially used for the solution of very heterogeneous problems. Of course, for the solution of the optimization problems, there are many more algorithms than were indicated here. Because their description would exceed the framework of this text, we can only refer to the corresponding literature, where the algorithms indicated above are described in more details.

2.4 Selected Basic Terms from the Evolutionary Algorithms

For work with evolutionary diagrams (Figure 2.4), it is necessary to know the meaning of certain terms that occur in the terminology of evolutionary algorithms and optimization. Some of them will be explained in this section.

2.4.1 The Usability Areas of Evolutionary Algorithms

Until the present date, there are many algorithms that belong to the class of evolutionary diagrams or can be included into this class under certain conditions. Typical examples are the already mentioned Ant Colony Optimization algorithms, Immunology System Method, Scatter Search or Particle Swarm. These algorithms, like many others, are not universal, but from the principle of their action, they are always suitable for solving certain classes of problems. The class of problem may be of various "size" for each algorithm. Genetic algorithms, for example, can be used for a wide class of problems, while the ACO algorithm, acting on the principle of the behavior of ants, is essentially predetermined for combinatoric problems of the type of a traveling salesman, where its performance is excellent.

It is therefore obvious that it is not only sufficient to have a good algorithm, but it is frequently of vital importance to know with what class of problems a given algorithm can work. This means that it is therefore necessary to determine the usability range of a given algorithm. In the case of evolution algorithms, we will understand by this term the class of problems for which a given algorithm provides at least satisfactory results.

Most optimization problem can be viewed as a geometrical problem, whose goal is to the find the lowest (minimum) or highest point (maximum) on the N dimensional surface. Such surfaces, defined usually by some functional prescription, may suffer from various pathologies from the mathematical point of view. With respect to the tests carried out on the functions tested, whose algorithms are described below, one can state that the evolutionary algorithms are very efficient and usually suitable for global optimization (see Fig. 2.11 and Fig.2.12). This set of test function can be viewed as the usability range of the evolutionary diagrams. It holds for the test functions mentioned above:

1. The graph of the function does not have a fractal character;
2. They are defined on real, integer or discrete arguments;
3. They are multimodal (one or several extremes);
4. They have various limits (imposed on the arguments or the value of the objective function);
5. They are strongly nonlinear;
6. They represent problems of the type "needle in a haystack";
7. Finding the global extreme with evolutionary algorithms is less or more complicated;

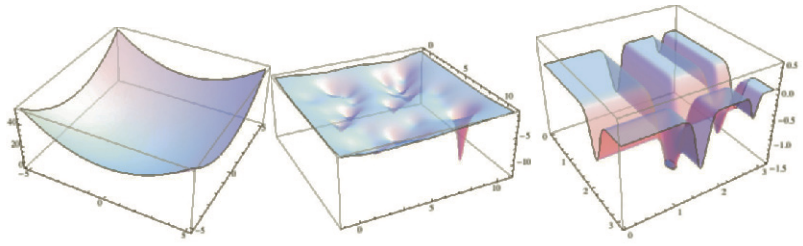


Fig. 2.11 Examples of functions that exhibit certain combinations of properties 1-7, a-c

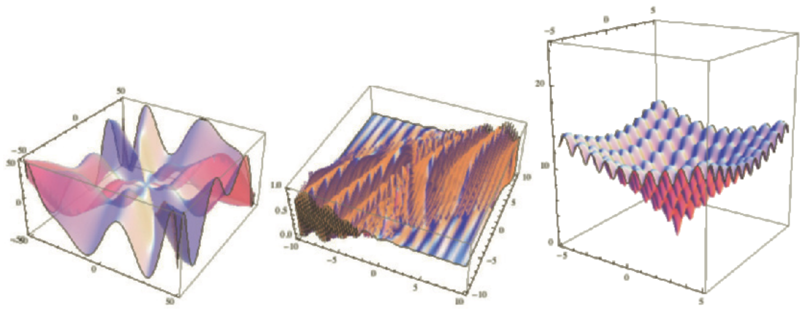


Fig. 2.12 Examples of functions that exhibit certain combinations of properties 1-7, a-c

moreover, it may hold true that:

(a) The function is separable (non-separable), which means that it can be (cannot be) decomposed into several simpler functions that can be optimized separately;
(b) The number of variables is high;
(c) The space of possible solutions may be large and discontinuous.

Generally speaking, evolutionary algorithms can be used to find the optima of functions from a very large class. Other information such as gradient, etc., are usually not necessary.

2.4.2 Common Features

Evolutionary algorithms have certain common features.

1. **Simplicity**, because algorithms can usually be programmed in a simple manner.
2. **Hybridity** of numbers with which the algorithms work. Numbers of the *integer, real* type, or, as the case may be, only selected sets of numbers (usually denoted as discrete), such as, for example, -5, 2, 8, 55, 3, 100, can be combined without any problem.

3. **Use of decimal numbers** - The individual need not be converted into the binary code that is commonly used in genetic algorithms. By the conversion into a binary code, a given number is distorted (the binary string has a limited length). When binary recording is used, mutations may cause a sudden change of the number, which may not have a good impact on the course of evolution. For example, numbers 15, 16 and 17 are represented as 01111, 10000 and 10001. The transition from 15 to 16 means the inversion of all five bits, i.e. a 100% mutation. However, transition from 16 to 17 requires the mutation of only one bit. Although this "unevenness" can be removed with what is called Gray coding (see Section 4.5), work with real numbers is still more convenient.
4. **Speed** - Thanks to its relative simplicity, particularly in comparison with classical methods, one can say that the required solutions are found much faster.
5. **The ability to find an extreme also for functions that are flat** from the graphical point of view and the extreme is just a "hole" in this plane. With some exaggeration, searching the extreme in such a function can be denoted as "looking for a needle in a haystack". Unfortunately, the efficiency of any algorithms, including evolutionary, is very low in these problems. If the surface around the extreme is a plane, than finding the extreme is usually a matter of chance.
6. **Ability to provide manifold solutions** - The best individual is the result of evolution - one solution. However, if, for example, the three best individuals are selected from the last population, they represent three different solutions of the problem. Of course, with graduated quality. If there are more global extremes in a given problem, one can expect that they will also be found by the evolutionary process. Therefore, there will be several solutions of the same quality available. As an example, the design of the optimum geared transmission can be used [47], [73], for which the method of differential evolution obtained four alternative solutions with the same value of objective function or a test function that has two global extremes at different points. The majority of good evolutionary techniques are able to localize these extremes.

In other words, the evolutionary algorithms are suitable for looking for extremes of functions suffering from such pathologies, such as, for example, noise, a high number of dimensions, "multi-modality" (several local extremes).

2.4.3 Population

A typical feature of the evolutionary algorithms is that they are based on work with the population of individuals. Population can be represented as matrix *N*x*M* (Fig. 2.13), where the columns represent individuals. Each individual represents the current solution of the problem. Essentially this is a set of arguments of the objective function, whose optimum numerical combination is searched for gradually. Moreover, a value of the objective function is connected with each individual (sometimes "fitness") that tells how the individual is suitable for further evolution of the population. This value does not participate in the evolution process itself. It only carries information on the quality of the corresponding individual.

	I_1	I_2	I_3	I_4	..	..	..	I_M
Fitness	**55.2**	**68.3**	**5.36**	**9.5**	..	..	..	**0.89**
P_1	2.55	549.3	-55.36	896.5	..	..	..	1.89
P_2	0.25	66.2	2	-10	..	..	..	-2.2
P_3	-66.3	56	4	15.001	..	..	..	-83.66
..	..	..	..	..	..	..	..	..
P_N	259.3	-10	22.22	536.22	..	..	..	-42.22

Fig. 2.13 Population (of the *NM* size), J_x is the x-th individual, P_i is the y-th individual Fitness - individual quality measured by means of the objective function

For the generation of population, it is necessary to define a specimen, see (2.1), according to which the entire initial population is generated. This sample individual is also used for correcting the parameters of individuals, who exceed the boundaries of the space searched.

$$\text{Specimen} = \{\{\text{Real}, \{Lo, Hi\}\}, \{\text{Integer}, \{Lo, Hi\}\}, \ldots \ , \{\text{Real}, \{Lo, Hi\}\}\} \quad (2.1)$$

In the sample, three constants are defined for each parameter of a specific individual from the population: The type of variable (i.e. integer, real, discrete, etc.) and the boundaries of the interval in which the value of the parameter may be. For example, {Integer, {*Lo*, *Hi*}} defines an integer parameter with the bottom limit *Lo* and upper limit *Hi*. The choice of limits is a very important step, because if they are chosen in an unsuitable manner, it may happen that solutions will be found that are not physically real (for example, a negative thickness of the pressure vessel), or will not be substantiated (for example, an airplane without wings, a pressure vessel whose wall thickness equals its radius, etc.).

Another equally important meaning of the boundaries is related to the evolutionary process itself. It may happen that a given optimization problem will be represented by a surface, on which the local extremes will assume greater values with the increasing distance from the origin (Fig. 2.14). This will cause that the evolution will be finding new solutions until infinity. Of course, if termination is not specified in dependence on the number of evolutionary cycles (generations, migrations, annealing,). This is caused by the fact that the evolutionary process always proceeds to deeper and more distant extremes on this (Schwefel's) function.

Population is generated on the basis of the sample individual by means of 2.2, see also [47]. $P^{(0)}$ represents the initial population, x_{ij} is the j-th parameter of the i-th individual.

$$\begin{array}{l} P_{i,j}^{(0)} = x_{i,j}^{(0)} = rnd[0.1] \cdot (x_{i,j}^{(Hi)} - x_{i,j}^{(Lo)}) + x_{i,j}^{(Lo)} \\ i = 1, \ \ldots \ , \ M \quad , \quad j = 1, \ \ldots \ , \ N \end{array} \quad (2.2)$$

In Fig. 2.14 and Fig. 2.15, two randomly generated populations of ten individuals are represented. The individuals were two-dimensional in this case. It is obvious

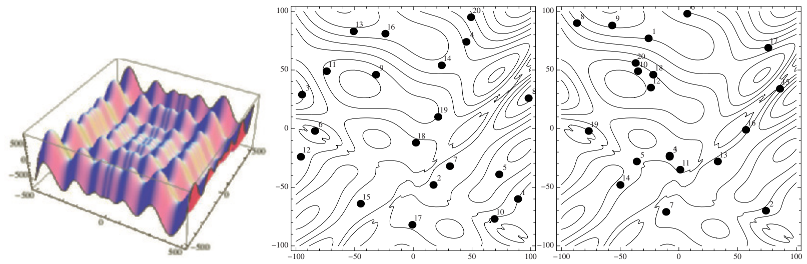


Fig. 2.14 Schwefel's function [64] with the growing extremes in the direction from the origin (a) and two randomly generated populations (b-c) on another function

I_1	I_2	I_3	I_4	..	..	..	I_{10}
424	104	53.3	942.9	..	..	..	178.008
-1.8	-1	0.7	-1.25	..	..	..	-1.19
1.2	2	1.2	-1.5	..	..	..	0.1

Fig. 2.15 Numerical representation of two randomly generated populations from Fig. 2.14 b)

from both figures that the generation of population is essentially a random distribution of individuals in the space of possible solutions. It occurs during the run of a given evolutionary algorithm that the individuals gather around one (usually global) or more extremes, which is graphically illustrated in Fig. 2.17. Mapping of information on how the evolution proceeded qualitatively is carried out by means of the evolution history of the objective function value in form of a simple graph. The dependence of the evolution of the value of the objective function on the evolution cycle is illustrated in this figure (Fig. 2.16). This is the sequence of the worst (upper curve) and best (bottom curve) solutions from individual populations. A mapping more convenient than that described just now is plotting the dependence of the value of the objective function on the current number of objective function evaluations. This approach is suitable because during evolutionary cycles (generation, annealing cycles, migration cycles, etc.), various numbers of evaluations of the profit function are carried out in individual algorithms. In the first method of graphical mapping, the slower convergence of the values of the objective function may be displayed as the faster one and vice versa. The true information on the quality of evolution may then be distorted. However, if we use the second method, it is then possible to compare various types of algorithms irrespective of their inner structure.

Besides the evolution of the best individual (or the best individuals when repeating the simulation), it is also suitable to display the evolution of the worst individual from the population in one graph. This reveals the overall convergence of the population as such. In a case where the courses of the best and worst individuals meet soon in the same extreme, it is possible that this is the case of a local extreme. If the

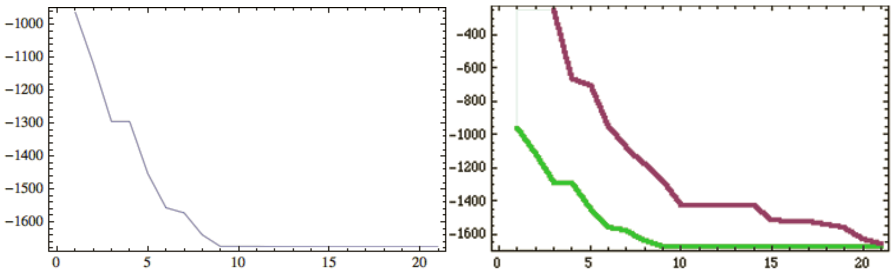
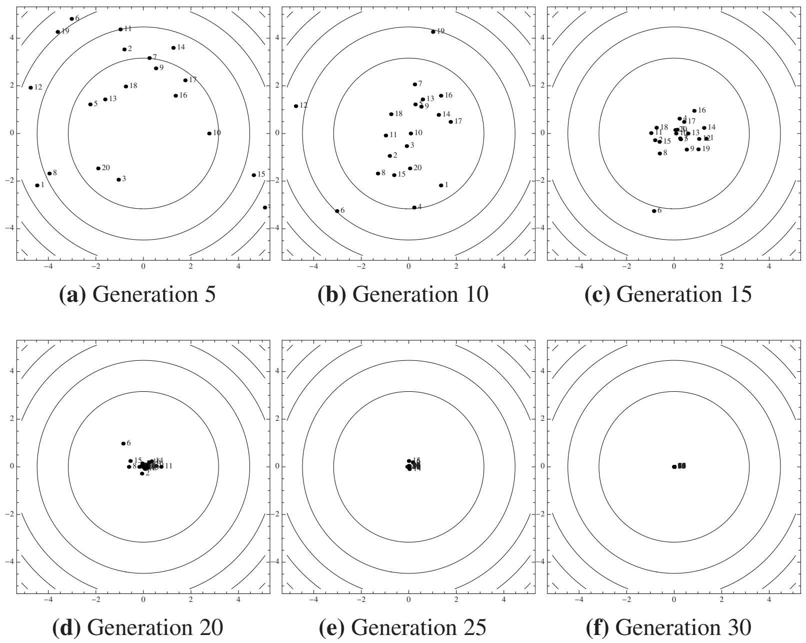


Fig. 2.16 Evolution of the value of the objective functions during evolution. This is a sequence of the best solutions from individual populations in dependence on the evolution cycle (generation, migration cycle, etc.).



(a) Generation 5 **(b)** Generation 10 **(c)** Generation 15

(d) Generation 20 **(e)** Generation 25 **(f)** Generation 30

Fig. 2.17 Convergence of the population to the global extreme in individual evolutionary cycles. The start display was omitted; individuals are uniformly distributed on the entire surface during the start.

best and worst individuals have the same value of the objective function, then only two explanations are possible:

1. The population is distributed in several extremes with the same value of the objective function or
2. The entire population is in one extreme, which is more probable, because a lot of problems are represented by a function with one global extreme.

In case No. 1, there is a chance that the evolution will proceed further, while in case No. 2, the same value of both individuals shows that further evolution is useless. The evolution of the population **must** always converge to better values, which means that it may never show divergence. If a minimum (maximum) is sought, the evolution must converge to lower (higher) values. If this is not the case, then in a given algorithm, "elitism" is somehow disrupted (elitism serves as some kind of a one way filter, which transmits only *such solutions that are better or equally good as those from the old population*). In the event of its dysfunction, the given algorithm would degrade to merely a random search.

2.4.4 Individuals and Their Representation

Several methods are being used when representing individuals in the evolutionary algorithms. The binary representation is historically the oldest one. In this case, the individual is formed by a 0 and 1 sequence called a chromosome [53], [2]. This representation of individuals has its historic roots and is being used in genetic algorithms up to date. In spite of its extension, it has its disadvantages. The basic disadvantage is the step change of the structure of chromosomes, the corresponding real values during a continuous change. In order to prevent these undesirable changes in the behavior of the binary code, the so-called Gray code is used. This is again a binary code, however, completely without the step changes mentioned above. The transformation of the standard binary code into the Gray binary code and the inverse transformation are illustrated in Fig. 2.18. The accuracy with which individuals are able to occur in the space of possible solutions is related to the binary representation of individuals. Basically, if the binary individual is short and represents a number with a few digits behind the decimal point, then it will occur only in certain positions of the real space of possible solutions. If its length and thus also accuracy grow, then the density of the positions of possible occurrences will grow. Nevertheless, one should note that the growing length of the individual causes considerable problems during further operations in the corresponding evolutionary algorithm (mutations, crossbreeding ...).

Furthermore, individuals can be represented in the form of real or integer numbers [57] or, as the case may be, their combinations, depending on the type of the algorithm [73]. The individual that also contains non-numerical values is a special representation. With the use of special techniques, it is possible to work numerically also with this individual. The last special form of representation is the so-called "tree". This form of representation makes it possible to visualize the tree structure, nevertheless, in the computer sense, the string of suitable symbols of a certain character is still the individual. This kind of visualization is is usually used in so called genetic programming [48], which is advanced evolutionary technique, used to manipulate with symbolic structures and create in this way more complex structures. Another similar approach is grammatical evolution [55], [20].

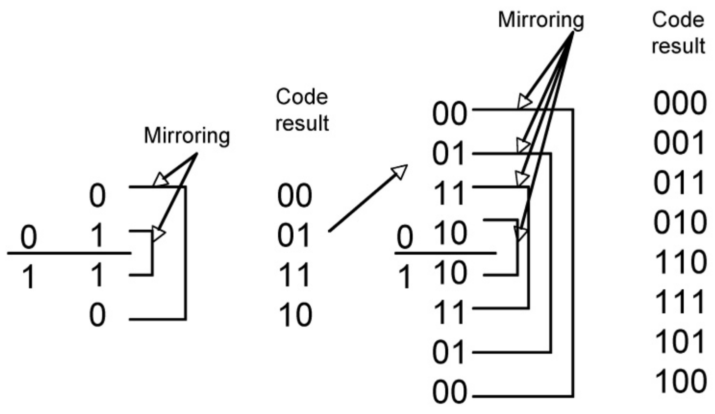


Fig. 2.18 Generation of the Gray code - reflection method

2.4.4.1 Binary and Gray Code Representation

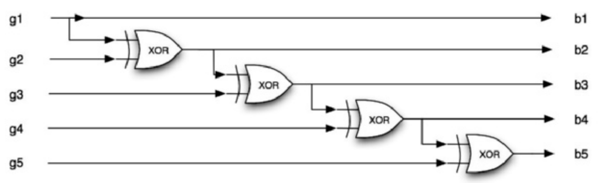
The binary code is formed over the alphabet $[0,1]$ as a number representation system with base–2. Any (integer) number n can be expressed by the binary string $(b_N b_{N-1} \ldots b_i \ldots b_2 b_1 b_0)$ with all the $b_i \in [0,1]$. The number n converts to the binary string by

$$n = b_N 2^N + b_{N-1} 2^{N-1} + \ldots + b_2 2^2 + b_1 2^1 + b_0. \tag{2.3}$$

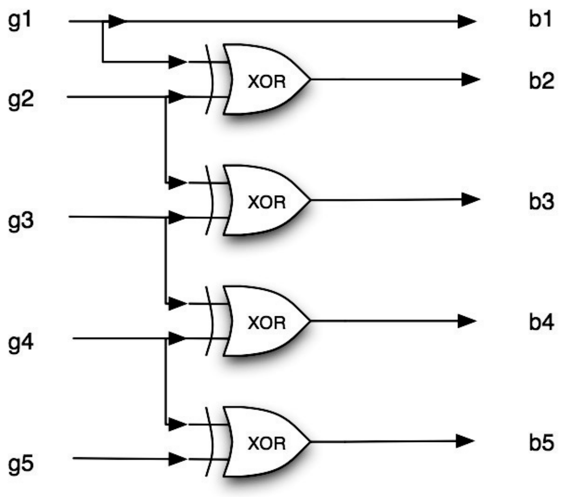
Note that in the binary string every digit (bit) b_i counts for a distinct numerical value, 2^{b_i}. So if this bit changes, it depends on the exact bit position how big the change in the represented numerical value is. Clearly, this change in represented numerical value can be considerable if the bit is far left in the bit string. Next to the binary code, the Gray code is frequently used for genetic algorithms. It is also called the constant change code since when mutating a Gray–coded individual, the real number that corresponds to the corresponding binary sequence does not change much.

The Gray code was patented in 1947, when Frank Gray asked to register it under the name of reflected binary code. However, users started using the name Gray code according to its founder [34]. One of the alternatives of the construction of the Gray code is described in Fig. 2.18. It is the so-called "reflection" method. It consists in taking the n bit code, for example, for n = 2 code 00, 01, 11, 10. This is then extended by its mirror copy 00, 01, 11, 10, **10, 11, 01, 00** and 0 is added to the original part, while to the reflected 1: **0**00, **0**01, **0**11, **0**10, **1**10, **1**11, **1**01, **1**00. This is repeated until we have the required m bit string of the Gray code.

Another method that can be easily implemented by a computer is using the XOR operation. The principle of transformation of the Gray code into the binary code is prescribed by relation (2.4) and illustrated in Fig. 2.19a. The inverse transformation, i.e. from the binary into the Gray code is given by relation (2.5) and illustrated in



(a)



(b)

Fig. 2.19 Generation of the Gray code - XOR method, a) Conversion from the Gray code into the binary code, b) Conversion from the binary code into the Gray code

Fig. 2.19b. Symbol *bk* and *gk* represents the *k*-th bit of the standard binary code or, as the case may be, Gray code.

As mentioned above, the binary strings in the Gray code always differ only in one bit when changing their decimal equivalent by one. The distances between individual numbers have therefore the Hamming distance equal to one. The Hamming distance is defined as the number of bites in which two binary strings differ. The difference between the binary and Gray coding for numbers from zero to seven can be seen in Table 2.1.

$$\begin{aligned}
&b1 = g1\\
&b2 = g1 \oplus g2 = b1 \oplus g2\\
&b3 = g1 \oplus g2 \oplus g3 = b2 \oplus g3\\
&b4 = g1 \oplus g2 \oplus g3 \oplus g4 = b3 \oplus g4\\
&b5 = g1 \oplus g2 \oplus g3 \oplus g4 \oplus g5 = b4 \oplus g5
\end{aligned} \tag{2.4}$$

$$\begin{aligned} g1 &= b1 \\ g2 &= b1 \oplus b2 \\ g3 &= b2 \oplus b3 \\ g4 &= b3 \oplus b4 \\ g5 &= b4 \oplus b5 \end{aligned} \tag{2.5}$$

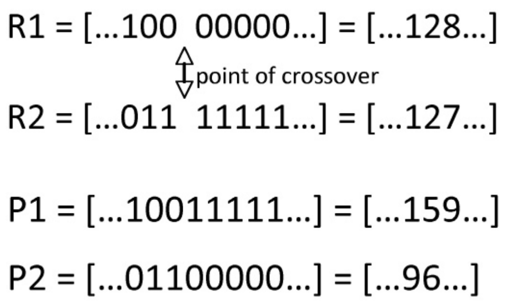
Table 2.1 Difference between the Gray and standard binary code

Decimal value	Gray code	Binary code
0	0	0
1	1	1
2	11	10
3	10	11
4	110	100
5	111	101
6	101	110
7	100	111

The advantage of the Gray code appears in the more uniform mutation of individuals and generally in faster convergence to the global optimum. However, there are differences in opinions. Some authors insist that the Gray code slows down the genetic algorithm (GA) due to the process of conversion [34]. On the contrary, other authors prefer the use of the Gray code ([11], [36]).

When using genetic algorithms during mutation or crossbreeding of standard binary individuals, the argument (gen) may change considerably, see Fig. 2.20. Such big changes do not occur in individuals in the Gray coding, Fig. 2.21.

Despite fact that evolutionary algorithms has very good performance, it is important to remember, that there are still problems, whose solution obtaining is still impossible and also, that there are some limits given to the computation by quantum physics. This is discussed in the following (last) section of this chapter.



R1 = [...100 00000...] = [...128...]

R2 = [...011 11111...] = [...127...]

P1 = [...10011111...] = [...159...]

P2 = [...01100000...] = [...96...]

Fig. 2.20 Crossbreeding of individuals in standard binary coding (R1, R2 - parents; P1, P2 - descendants)

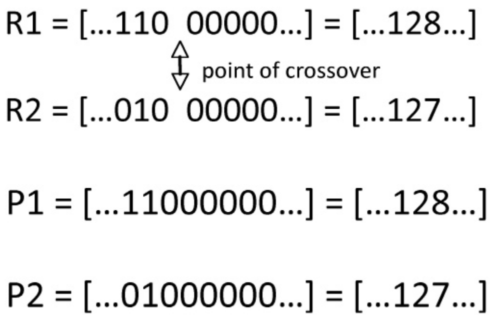

R1 = [...110 00000...] = [...128...]
point of crossover
R2 = [...010 00000...] = [...127...]
P1 = [...11000000...] = [...128...]
P2 = [...01000000...] = [...127...]

Fig. 2.21 Crossbreeding of individuals in Gray coding (R1, R2 - parents; P1, P2 - descendants)

2.4.4.2 Real, Integer and Discrete

Another way of individual representation is, when individuals are represented not only by by binary strings, as written above, but also in strings of real and integer numbers. Special case of integer representation are so called discrete sets, as explained later.

In the real representation is individual represented by string of real numbers like for example 2.3, 22.56, -569.2, ... Process of mutation, crossover, etc are then governed by used evolutionary algorithm.

In its canonical form, EAs are usually only capable of handling continuous variables. However, extending it for optimization of integer variables is rather easy. Only a couple of simple modifications are required. First, for evaluation of the cost-function, integer values should be used. Despite this, the EAs itself may still work internally with continuous floating-point values. Thus,

$$\begin{aligned} & f_{\text{cost}}(y_i) \;\; i = 1,..,n_{param} \\ & where: \\ & \quad y_i = \begin{cases} x_i & \text{for continuous variables} \\ INT(x_i) & \text{for integer variables} \end{cases} \\ & x_i \in X \end{aligned} \tag{2.6}$$

INT() is a function for converting a real value to an integer value by truncation. Truncation is performed here only for purposes of cost function value evaluation. Truncated values are not assigned elsewhere. Thus, EA works with a population of continuous variables regardless of the corresponding object variable type. This is essential for maintaining the diversity of the population and the robustness of the algorithm.

Secondly, in case of integer variables, the population should be initialized as follows:

$$\begin{aligned} & P^{(0)} = x_{i,j}^{(0)} = r_{i,j}\left(x_j^{(High)} - x_j^{(Low)} + 1\right) + x_j^{(Low)} \\ & i = 1,...,n_{pop}, \quad j = 1,...,n_{param} \end{aligned} \tag{2.7}$$

Additionally, the boundary constraint handling for integer variables should be performed as follows:

$$x_{i,j}^{(ML+1)} = \begin{cases} r_{i,j}\left(x_j^{(High)} - x_j^{(Low)} + 1\right) + x_j^{(Low)} \\ \quad if \quad INT\left(x_{i,j}^{(ML+1)}\right) < x_j^{(Low)} \vee INT\left(x_{i,j}^{(ML+1)}\right) > x_j^{(High)} \\ x_{i,j}^{(ML+1)} \quad \text{otherwise} \end{cases} \tag{2.8}$$

where,
$i = 1,...,n_{pop}, \quad j = 1,...,n_{param}$

Discrete values can also be handled in a straight forward manner. Suppose that the subset of discrete variables, *X(d)*, contains *i* elements that can be assigned to variable *x*:

$$X^{(d)} = x_i^{(d)} \qquad i = 1,...,l \quad \text{where} \quad x_i^{(d)} < x_{i+1}^{(d)} \tag{2.9}$$

Instead of the discrete value x_i itself, its index, *i*, can be assigned to *x*. Now the discrete variable can be handled as an integer variable that is boundary constrained to range $\{1,2,3,..,N\}$. In order to evaluate the objective function, the discrete value, x_i , is used instead of its index *i*. In other words, instead of optimizing the value of the discrete variable directly, the value of its index *i* is optimized. Only during evaluation is the indicated discrete value used. Once the discrete problem has been converted into an integer one, the previously described methods for handling integer variables can be applied. The principle of discrete parameter handling is depicted in Fig 2.22. This technique is called *discrete set handling* (DSH), see [46].

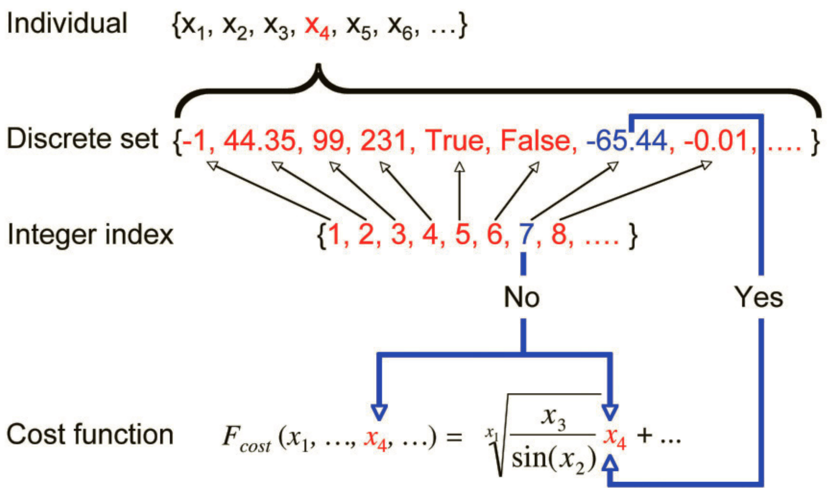

Individual
{x1, x2, x3, x4, x5, x6, …}
Discrete set
{-1, 44.35, 99, 231, True, False, -65.44, -0.01, …. }
Integer index
{1, 2, 3, 4, 5, 6, 7, 8, …. }
No
Yes
Cost function
$F_{cost}(x_1, \ldots, x_4, \ldots) = \sqrt[x_1]{\frac{x_3}{\sin(x_2)}}\, x_4 + \ldots$

Fig. 2.22 Discrete parameter handling

2.4.4.3 Tree Representation

The tree representation used by EAs, come from initial idea of so called symbolic regression by means of a computer program. It was proposed in Genetic Programming (GP), [43], [42]. Genetic programming was the first tool for symbolic regression carried out by means of computers instead of humans. The main idea comes from genetic algorithms (GA), which was used in GP [43], [42]. Its ability to solve very difficult problems is well proved; for example, GP performs so well that it can be applied to synthesize highly sophisticated electronic circuits [44].

The main principle of GP is based on GA, which is working with populations of individuals represented in LISP programming language. Individuals in a canonical form of GP are not binary strings, different from GA, but consist of LISP symbolic objects like *sin*, +, *Exp*, etc. These objects come from LISP, or they are simply user-defined functions. Individuals in genetic programming, in its canonical form, are thus commands of Lisp language. There are also another techniques like Read's linear coding [59] and DSH technique. DSH technique mentioned above, allow EAs to manipulate with such a structure also via integer index. Individual is in fact integer string, which is converted into symbolic expression. Individuals can be then visualized in the form of so called trees, which are graphical unfolding of nonnumerical expressions, see Fig. 2.23. Individuals in this form can represent not only classical mathematical expressions, but also logical functions (Fig. 2.24 and Fig. 2.25), or elements of electronics circuits (Fig. 2.26). Read's linear code is representation based on string of integer numbers, as shown on Fig. 2.27 and Fig. 2.28. In the case of Read's representation each vertex has associated number according to number of

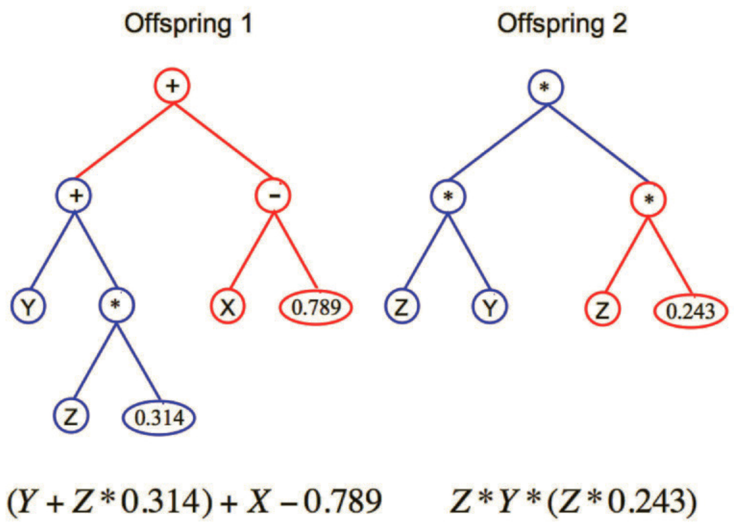


Fig. 2.23 An example of tree representation. Individuals are represented graphically by trees. Crossover is nothing more than cutting and exchange of randomly selected sub-trees.

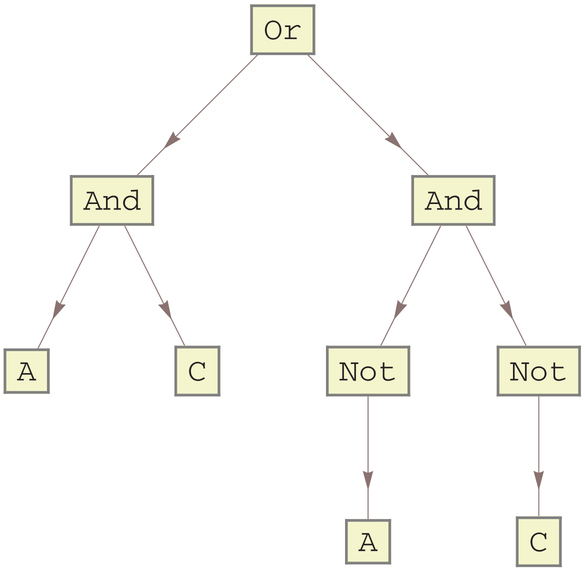


Fig. 2.24 Tree representation of logical function

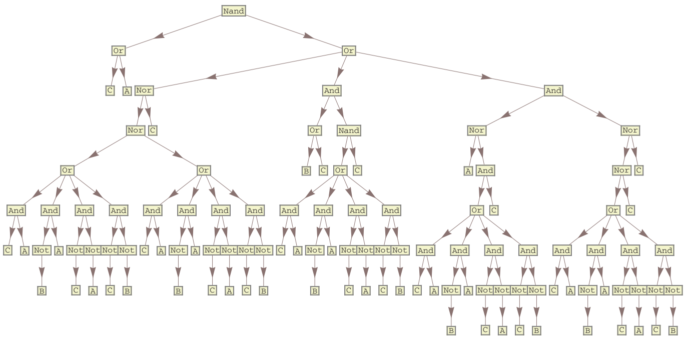


Fig. 2.25 Tree representation - more complicated example

outcomming vertexes. Code of arbitrary tree is obtained so that labels of vertexes are “joined” (according to dotted line with arrows, see Fig. 2.28) into integer string with a such condition that used vertex label is further ignored, to keep unicity of description. There is more techniques of how to represent individuals for genetic programming techniques, however above mentioned techniques (Lisp, Read’s linear code) are well known. DSH technique can be regarded like experimental novelty technique, which is mostly used in this book.

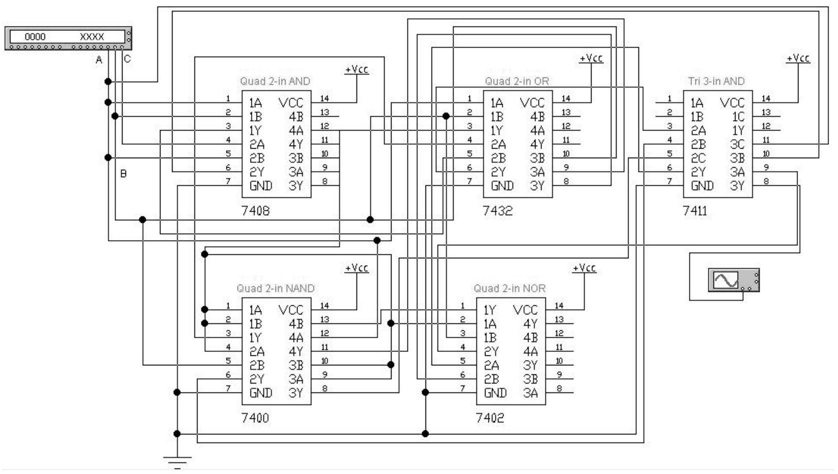


Fig. 2.26 Electronic realization of evolutionary designed circuit via evolution with individuals in DSH representation

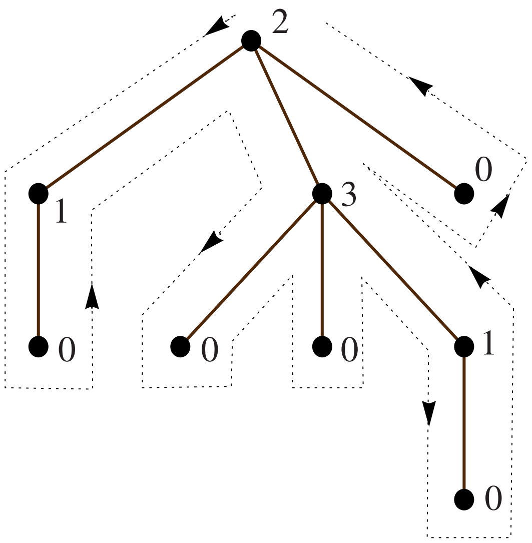


Fig. 2.27 Read's tree. Each vertex has associated number according to edges coming out of vertex. Unique code of tree is constructed according to dashed arrows.

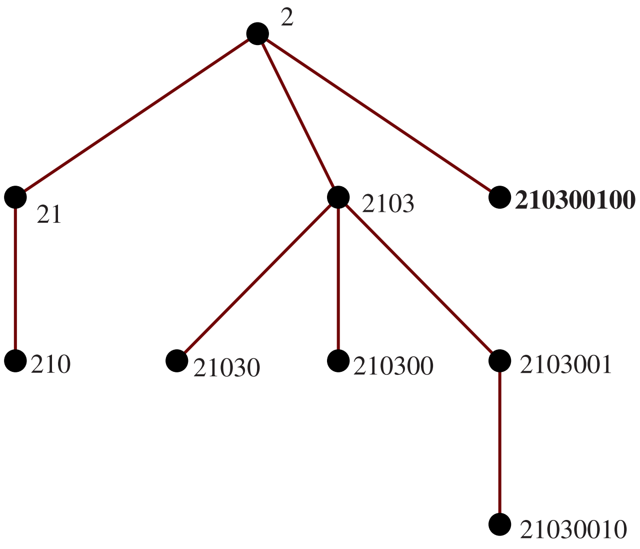


Fig. 2.28 Code of Read's Tree - 210300100

2.4.5 Evolutionary Operators: Selection, Recombination, Mutation

The algorithmic cycle of evolutionary algorithms relies upon the working of three main operators, selection, recombination and mutation. All three operators play a distinctive role in solving the posed optimization problem. In general, during the evolutionary run, we can separate two phases, exploring and exploiting. In the exploring the individuals should cover large (ideally all) parts of the search space in order to find regions where optimal values are likely to be found. Hence, individuals should be different, the diversity of the population should be high. After that exploration phase, a promising region in the search space should be searched more detailed, so exploiting the knowledge about the distribution of fitness in the search space should set in. In this phase the individuals of the population should be pushed towards the actual optima. Clearly, both phases are necessary for successful problem solving. If the exploration phase is missing or too short, the individuals might settle for local optima, missing the global optima in this process. Without exploitation, the exact location of the best solution might not be found. Overlaying both phases is the intention to exclude clearly inferior solutions from hindering the search process. Against this background, the working of the evolutionary operators can be understood.

Selection: The selection mechanism organizes that individuals with higher fitness become the material from which the next generation is produced. In doing so, it is important (particularly in the exploration phase) that not only the very best individuals are kept, but also "promising second-bests". A good selection mechanism should balance the selective pressure (that is only the best are to survive) with maintenance of residual diversity in the population. For achieving this balance, different kinds of selection schemes have been proposed. They can be roughly distinguished in purely deterministic selection, where either based on ranking or by setting a fitness threshold a certain percentage of the population is kept, and guided stochastic selection, where samples of the population are randomly picked and based on the comparison of their fitness values a decision is made to discard or keep them.

Recombination: In recombination (also called crossover, in particular for GAs) new individuals are created based on those previously selected for their superior fitness. In a first step, two (or sometimes even more) individuals are appointed to be parents. This appointment can be either deterministic by working off the whole selected population, or stochastic where the individuals are determined randomly. Then, the schemes proposed for performing the step differ largely for different kinds of representation. For evolutionary algorithms that use binary representation, we find that both parents swap or shuffle (sometimes at more than one place) subsections of their binary string. For real valued representation, a (sometimes weighted) arithmetic or geometric mean between both parents yields the offspring.

Mutation: Following the recombination step, the produced offspring are altered randomly by mutation, mostly in a marginal manner only. Therefore, mutation rate (which defines the probability that the offspring are subjected to mutation) and the mutation strength (which fixed the magnitude of the changes in the offspring) must be set. Again, we find a difference between binary and real valued representation. For binary stings, we have a flipping of a bit ($0 \rightarrow 1$ or $1 \rightarrow 0$) at one or more places in the binary string. For vectors of real numbers, realizations of a (mostly normally distributed) random variable are added to the offspring.

The evolutionary operators just considered can be regarded as the backbone for the majority of ECT methods, although not all of them must be part in a specific implementation and also their respected role and importance might be largely different. In general, mutation is next to a sensible choice of the initial population the source of random and diversity enhancement in the algorithm. It helps to explore the search space. Selection, on the other hand, acts mainly as a filtering for superior solution candidates. In this, it exercises selection pressure on the population. However, for a given type of ECT the same operator might have a different flavor. So, in GAs mutation is a minor operator, while in ESs it is the main component. Such differences depend frequently on the type of representation. For a binary representation, as GAs use, a single bit change in the right place of the binary string, as induced by mutation, can cause a very dramatic change of the encoded solution. The very same bit flip can alter the string largely or a little, depending on where in the string it happens. This is not the case for real value representation, where the magnitude of the change caused by mutation can be closely controlled but mutation rate and strength.

2.5 Limits to Computation

Unfortunately, many people believe that everything can be computed if we have a sufficiently powerful computer and elegant algorithm. The goal of this chapter is to show that some problems cannot be solved algorithmically due to their nature. Popularly speaking, there is not, has not been and will not be enough time for their solution.

Part of these restrictions are also physical limits that follow from the material nature of the universe, which restricts the output of every computer and algorithm by its space-time and quantum-mechanical properties. These limits, of course, are based on the contemporary state of our knowledge in physical sciences, which means that they might be re-evaluated in the case of new experimentally confirmed theories (strings, etc.). At this moment, however, this is only a speculation and we must adhere to the generally accepted and confirmed facts from which these limits follow.

2.5.1 Searched Space and Its Complexity

The complexity of the optimization problems can be demonstrated by many examples. Let us follow examples from [54]. A typical representative is the so-called SAT problem (boolean satisfiability problem). This is a problem from the field of logic that is represented by a complex logical function with a great number of logical variables. Relation 2.10 is an example from [54].

$$F(x) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \ldots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97}), \tag{2.10}$$

that contains 100 variables and the objective is to find such values of individual arguments of this function for which the resulting value of relation 2.10 is TRUE. At first sight, this problem looks very trivial; nevertheless, it is a problem that cannot be solved by classical methods. If we take into account that the expression contains 100 unknown variables that can assume two values (0,1), then the number of all possible combinations is 2^{100}, which is approximately 10^{30}. In order to get a better impression on the monstrous size of this number, it is sufficient to imagine how long it would take to evaluate all the combinations, if 10^{13} of these combinations are evaluated within one second (which is of course impossible on single processor). The correct answer is 10^9 years. This essentially means that the solution of this problem would take approximately the time of the existence of the universe.

Another complication related to this problem is the fact that function 2.10 as defined does not make it possible to evaluate the quality of the current solution. This is a substantial drawback, particularly if the evolutionary techniques are used, because there is no possibility how to determine whether the qualities of two subsequently found solutions are close or not. As will be shown further, when using the evolutionary algorithms, it is of vital importance that the information on the quality of the solution is available for the determination in which “direction” the optimum solution lays. This is not possible in the case of the SAT problem, because the function

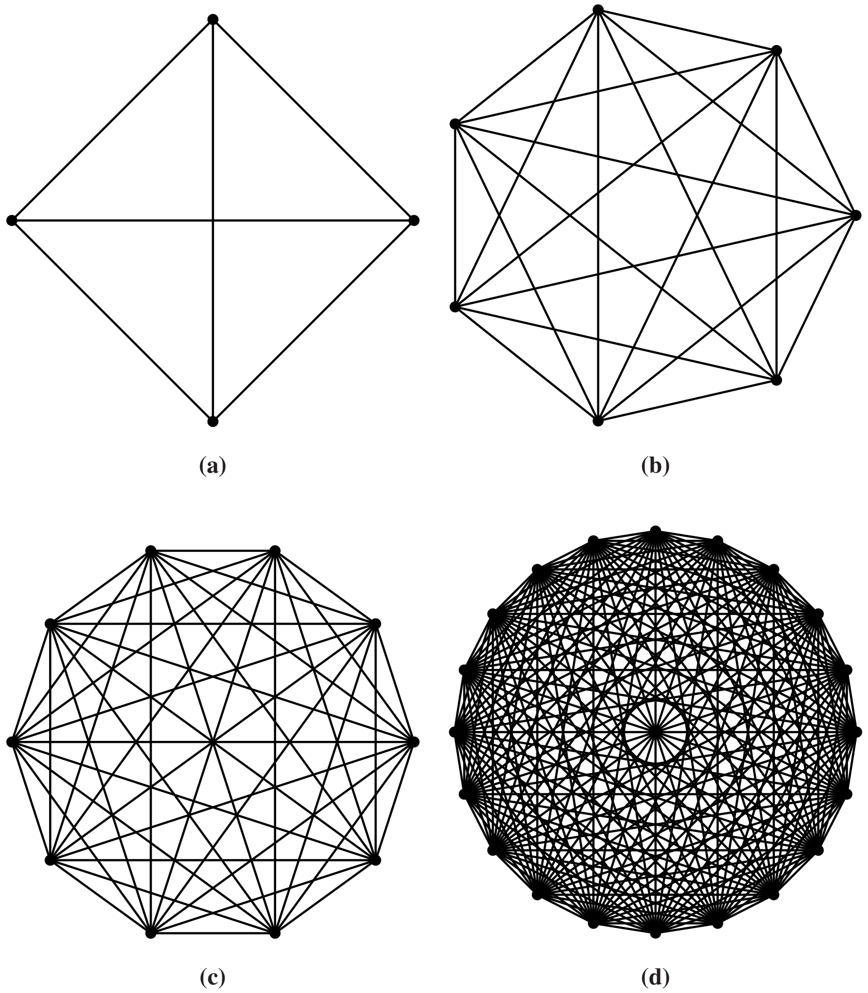


Fig. 2.29 Connections in the traveling salesman problem that form n! possible trajectories (see Fig. 2.32). We indicate the number of cities / number of connections between the cities a) 4/6, b) 7/21, c) 10/45, d) 20/190.

only returns TRUE or FALSE, i.e. "good" or "bad". It does not return how good or bad a given solution is.

The SAT problem is more or less a scholastic problem. As a more practical problem from real life, one can use the well known traveling salesman problem. This is a problem, in which a traveling salesman must visit a set of *N* cities in the shortest possible time or with the smallest fuel consumption or, as the case may be, fulfill other criteria. The traveling salesman problem can be visualized by means of graphs, as demonstrated in Fig. 2.29 - 2.31.

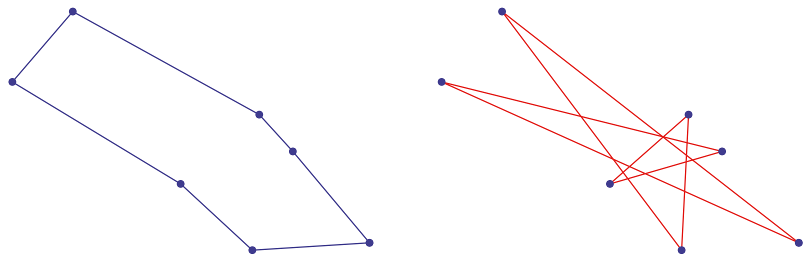


Fig. 2.30 Traveling salesman visiting seven cities: The best route is on the left and the worst route is on the right

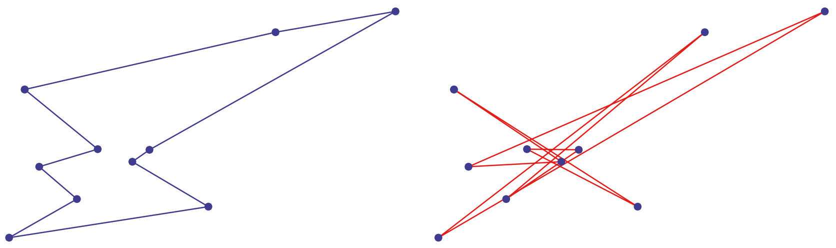


Fig. 2.31 Traveling salesman visiting ten cities: The best route is on the left and the worst route is on the right

The condition is that each route must start and end in the same city and each city should be visited only once. This is therefore a purely practical problem. The trajectory of the traveling agent represents a sequence of dots, such as, for example, “2 - 3 - ... - 7 - 26 ... ”. The number of all possible combinations is $n!$. In the case of a symmetrical problem of a traveling salesman (the distance from city A to B is the same as from city B to A), $2n$ routes repeats. In this case the final number of all possible combinations is $(n\text{-}1)!/2$. However, this number is still large. As shown in Fig. 2.33, the number of all possible combinations very quickly grows with the number of cities. Already for $n > 6$, there are more combinations in the traveling agent problem than in the SAT problem. Fig. 2.33 shows the growth of the number of solutions of the SAT problem in comparison with the growth of the complexity of the traveling salesman problem.

Let us look further. The traveling salesman problem has 181,440 possible solutions for 10 cities. There are 10^{16} possible solutions for 20 cities and 10^{62} for 50 cities. If 60 cities is used, then there is 10^{79} of possible solutions. This number is equal to the estimated number of protons in our universe, i.e. if one proton is used like memory to store one possible solution, then all protons in universe can store only TSP with size 60 cities. No more. It is worth mentioning that there is approximately 10^{21} liters of water on our Planet [54]. It is a trivial task to calculate how many globes could be covered with this volume of water had we used a reservoir

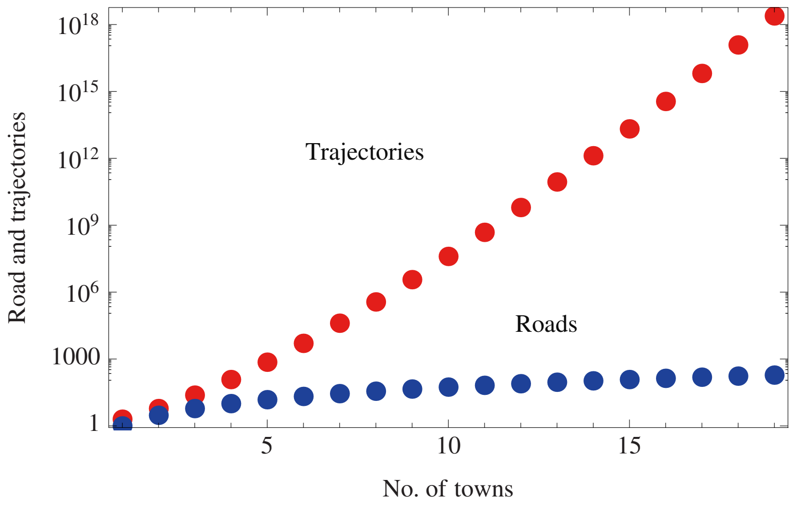


Fig. 2.32 Visualization of the travelling salesman complexity. The difference is illustrated between the number of roads (blue dots) and possible trajectories (red dots).

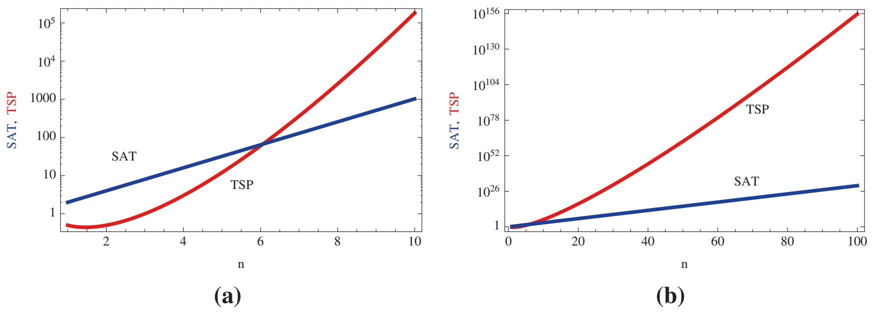


Fig. 2.33 Growth of the problem complexity for SAT (blue curve) and traveling salesman (red curve). Starting with seven cities (or variables in SAT), the traveling salesman problem is more time consuming.

with a volume of 10^{62} liters water. It is therefore obvious that even from such a trivial example as the optimum distribution of parcels, a problem may arise, whose optimum solution is not known. It is worth mentioning that at the present time, there are special types of evolutionary algorithms (ACO - Ant Colony Optimization) that manage up to 10,000 cities satisfactorily. We leave it to the kind reader to calculate what is the number of combinations (hint: 2.846259680910^{35659}).

The third and last sample problem is the artificial testing function, depicted in Fig. 2.34 that is used as a testing function for various types of evolutionary techniques;

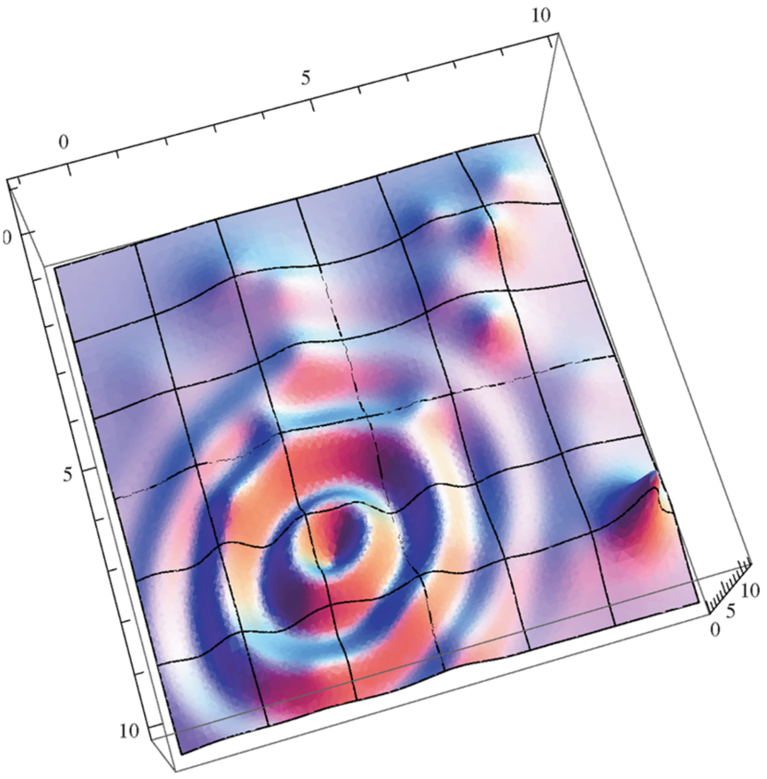


Fig. 2.34 Graph of test fuction

(for another example see [54]). This function is strongly nonlinear (for another similar functions see Fig. 2.11 and Fig.2.12) and it is complicated. Although the function in this example is artificial, one can encounter even "wilder" functions that represent real physical problems. This type of function looks innocent, however, it is the contrary in this case. It is necessary to realize that everything is running in computers, thus also the optimization of such a function, is digitized. If this would not be so, then it would necessary to calculate the value of the function in an infinite amount of points. Due to digitization, this infinity reduces to a set of values of the function, whose cardinality is finite, even though it is still immense. Let us assume that the computational accuracy of the computer used is 6 decimals. In this case every variable in a given function assumes real values. Through digitization, the infinity mentioned above reduces to a set of possible solutions, however the cardinality of this is still immense. Let us assume that variable in a given function Fig. 2.34 may assume up to 10^7 different values. In general terms, this function will assume 10^{7^n} values (n is a number of variables here). This number is many times greater than the number of solutions of the traveling salesman problem for $n \leq 10^7$. For $n = 50$, there are 10^{350} solutions. It is necessary to realize that the accuracy of present computers is much higher and the problem therefore generates a gigantic number of possible solutions.

Let us mention that the complexity of problems is not measured in theoretical informatics by the time demand factor (even though it is so de facto in the result), but primarily by the complexity or dependence of the capacity of the algorithm on the growing number of input data. As was already mentioned, there are problems whose complexity grows nonlinearly with the growing input (for example, the traveling salesman problem, see Fig. 2.33). We then speak about algorithms with polynomial, exponential, etc., complexity. The examples of the complexity of problems are in Table 2.2 - 2.4 (taken from [51]). Table 2.2 gives the number of possible solutions for n input parameters. If testing one solution takes the predefined time, the time demand factor for searching all possible solutions is in Table 2.3. If faster computers are used, the gross estimation of the acceleration of computation is in Table 2.4. It is obvious from these tables that there are many problems that no computer can help to solve.

Table 2.2 Estimation of the values of some functions

n Function	10	50	100	300	1,000
			Polynomial		
5n	50	250	500	1,500	5,000
$n\ log_2\ n$	33	282	665	2,469	9,966
n^2	100	2,500	10,000	90,000	1 million (7 digits)
n^3	1,000	125,000	1 million (7 digits)	27 million (8 digits)	1 billion (10 digits)
			Exponential		
2”	1,024	16 digit number	31 digit number	91 digit number	302 digit number
$n!$	3.6 million (7 digits)	65 digit number	161 digit number	623 digit number	giant number
n^n	10 billion (11 digits)	85 digit number	201digit number	744 digit number	giant number

For comparison: The number of protons in the visible Universe has approximately 79 digits The number of microseconds from the “big bang” has 24 digits.

Table 2.3 Estimation of the time of *f(n)* operations if 1 operation takes 1 μs

n / Function	10	20	50	100	300
			Polynomial		
n^2	1/10,000 s	1/2,500 s	1/400 s	1/100 s	9/100 s
n^5	1/10 s	3.2 s	5.2 s	2.8 hours	28.1 days
			Exponential		
2^n	1/1,000 s	1 s	35.7 years	400 trillion centuries	75 digit # of centuries
n^n	2.8 days	3.3 trillion years	70 digit # of centuries	185 digit # of centuries	728 digit # of centuries

Table 2.4 Estimation of the time of *f(n)* operations if 1 operation takes 1 μs

	Maximum dimension of the input manageable in a reasonable time		
Function	Current computers	100 times faster computers	1,000 times faster computers
n	N_1	100 N_1	1,000 N_1
n^2	N_2	10 N_2	31.6 N_2
2^n	N_3	N_3 + 6.64	N_3 + 9.97
$n!$	N_4	N_4 + 1	N_4 + 2

2.5.2 Physical Limits of Computation

As was already mentioned, there are limits restricting the output of any computer that follow from the quantum-mechanical nature of mass. These limits restrict both the output of the computer and its memory. It is obvious from these restrictions that there are many problems that no computer can help to solve.

Basic restriction in this direction is the so-called Bremermann's limit [8], according to which it is not possible to process more than 10^{51} bites per second in every kilogram of matter. In the original work of this author [8], the value of $2x10^{47}$ bites per second in one gram of matter is indicated. At first sight, this limit does not look frightening, but only until we take "elementary" real examples for comparison. Let us consider chess-mate for illustration. For this game, the estimated number of combinations is 10^{120}. As another example, let us consider the lattice of cellular automata [39] of 100 x 100 cells that can only assume black and white values that represents $2^{10,000}$, which is approximately $10^{3,000}$ combinations - images. The current TV sets with a LCD monitor have approximately 1,300x700 pixels, which can assume various colors and degrees of brightness. It is clear that the number of combinations is much higher on an LCD monitor.

This limit can be derived in the following relatively simple manner: For making it possible to measure, process and transfer information, it is **necessary** to store it on some physical carrier. This information may be electromagnetic radiation, paper tape, laser beam, etc., therefore always something material. Information alone, i.e., without a physical carrier, cannot exist. Because elementary particles and their energy states can also be used as a carrier of information, it is obvious that the limit of how much information the matter can carry follows from the restriction that was discovered at this physical level.

In order to make it possible to measure this information, it must be modulated on the corresponding carrier to resolve the individual carrier's states that represent the value of the information. Von Neumann[69] called the resolvable states "markers". The lowest resolvable energy states are the quantum states of matter, whose resolvability from the bottom is limited by Heisenberg's uncertainty relation. When deriving the already mentioned limits, it does not matter whether mass or energy types of carriers are considered. Both types are physically interchangeable. Therefore, if quantum states are considered as the smallest resolvable energy states, which will be considered as bits in this case, then the "energy-bit" resolution is given by Heisenberg's uncertainty relation. Generally, one can say that according to the Heisenberg principle of uncertainty it is possible to always identify the final number of states. Because nobody can say which state will be observed, probability has to be used. It is common to say that variable X will have n different values with probability $p_1, p_2, .., p_n$ Based on information theory is clear that we can get

$$H(p_1, p_2, .., p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2.11}$$

bits of information. This function has one global extreme only if it hold $p_1 = p_2 = ... = p_n = 1/n$ true. Then

$$H(1/n, ..., 1/n) = -\sum_{i=1}^{n} (1/n) \log_2 (1/n) = n(1/n) \log_2 n = \log_2 n. \tag{2.12}$$

Such marker can carry maximally $\log_2 n$ bits of information. Based on quantum nature of our world it is clear that there is no better marker than marker represented by n states (i.e. energy levels) of selected quantum system. All levels have to be in interval $[0, E_{\max}]$ where E_{max} is maximum of energy. If one can measure energy with precision ΔE, then in the marker, can be distinguished maximally $n + 1 = (E_{\max}/\Delta E) + 1$ energy levels. When one marker with n+1 energy levels will be taken into consideration, then by this marker can be represented maximally $\log_2 (n+1)$ of bits. On the contrary, when two markers will be used with energy levels in $[0, 1/2E_{\max}]$ it can represent $2\log_2 (n/2+1) = \log_2 (n/2+1)^2$ bits whereas $n+1 << (n/2+1)^2 = (n^2/4) + n + 1$ and so on. Based on this, it is clear that for representation of the maximal information carried by marker is optimal, when n different markers with energy levels in $[0,\Delta E]$ is used, i.e. with two energy levels which represents 0 and 1. In total it is possible to represent maximally $n\log_2(n/n+1) = n\log_2(n/n+1) = n\log_2 2$ i.e. n bits of information because clearly $\log_2 2 = 1$ hold.

Carrier with mass m is according to Einstein equation equal to $E_{\max} = mc^2$. It is obvious that in such a carrier it is possible to maximally have

$$n = \frac{E_{\max}}{\Delta E} = \frac{mc^2}{\Delta E} \tag{2.13}$$

bits of information. To calculate the exact amount of information stored by 2.13, then we need to use Heisenberg principle of uncertainty

$$\Delta E \Delta t \geq \frac{\hbar}{2} \tag{2.14}$$

In which $\hbar = h/2\pi$ (h is Planck constant, $\hbar$ is Dirac constant). If in 2.14 the equality is taken into consideration, then one obtains for the upper estimation

$$n = \frac{mc^2}{\frac{\hbar}{2\Delta t}} = 4\pi\frac{mc^2}{h}\Delta t \tag{2.15}$$

During time interval Δt it is possible to process maximally $4\pi\frac{mc^2}{h}\Delta t$ bits of information. When $\Delta t = 1s$ one can get maximal number of bits which can be processed or stored in mass per 1s. For $m = 1kg$ this number (lets call it BL) is

$$BL = 4\pi\frac{c^2}{h} \tag{2.16}$$

where $[BL] = 1kg^{-1}s^{-1}$

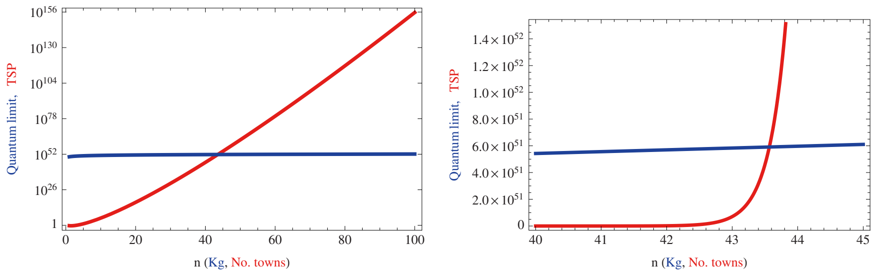
In this moment it is only a matter of simple calculation to get exact numerical value of BL, lets: speed of light and Planck constant $h = 6,62607 \cdot 10^{-34} J.s$. Finally we get

$$BL \approx 1,7045 \cdot 10^{51} kg^{-1} \cdot s^{-1} \tag{2.17}$$

This number, which we call BL here, is the so called Bremermann limit. It is the definite limit which gives maximal number of bits which can be processed or / and stored by an arbitrary matter. In the original paper Bremermann suggested 10^{47} which is caused by the use of nonstandard units (*cm* instead of *m* and *grams* instead of *kg*, as already mentioned before).

Based on this, it is visible that in our universe the computational power is limited by matter and basically there is no computer (existing or theoretical) which would be able to solve arbitrary problems.

If the mass of the Earth (5.9742×10^{24} kg) is taken into account, then a computer of such a mass might store (and subsequently also process) approximately 10^{76} bits every second. During the life of Earth (10^9 years), a computer of its mass might process maximally 10^{92} bits. If the output of a fictive computer is plotted against its mass, it is obvious (Fig. 2.35) that its "computational capacity" is exceeded already during the solution of the traveling salesman problem for a small number of cities/computer mass.



(a) **(b)** Detailed view

Fig. 2.35 Simultaneously plotted dependence of the number of possible solutions of the traveling salesman problem on the number of cities n (red) and the number of bits processed in a computer of mass m (blue). Let us add for more attentive readers that there is a logarithmic scale in the left figure, while a "normal" in the right figure. This is the reason why the plots appear considerably different in both figures.

It is clearly obvious from Fig. 2.35 that the break between the number of cities and the computer mass occurs somewhere between 43 and 44. Perhaps it is not necessary to mention that the output of our computer is illustrated in bits, which is a little bit misleading, because one bit is not sufficient for storing information on one possible solution of the traveling salesman problem. Had this been taken into consideration during the computation, then the result would have been different, nevertheless approximately the same as for the order of magnitude.

If we take into account the ACO (Ant Colony Optimization) algorithm that satisfactorily solves the traveling salesman problem up to approximately 10,000 cities, then we would need a computer of the mass of 10^{35608} kg for storing and processing the information on all possible trajectories. In other words, 10^{35566} computers of the mass of the Earth, should the computation be finished during the life of the universe (10^{17} s). In a similar way [50], we would derive the shortest possible time during which it is possible to process the stored information. This value is $t = 10^{-12}$ s; the current computers work in a region of 10^{-9} s.

In the publication [50], these considerations have been worked out in more details and applied to the transfer of information through an information channel (computation can also be considered as a transfer of information through a special channel). Beside other things, it was found that if a certain mass (or energy) of the transfer medium is reached, further information cannot be transferred through the channel, because the channel collapses into an astrophysical object called black hole. According to[50], the transfer of information is efficient (optimum, maximally usable), if the information channel is on the brink of collapsing into a black hole.

Independently of whether these calculations are accurate or only approximate, it is obvious that physical limits restrict the possibilities of any computer and also of the mathematical computational methods.

2.6 Conclusion

The principles of evolutionary techniques are described in many publications focused both on evolutionary diagrams and in publications "outside" the field, where it is necessary to inform the corresponding community of experts about these techniques. A representative example is [29]. In these and similar monographs, the problems of evolutionary algorithms are introduced at a very vague level. However, there are more suitable sources of literature intended for the needs of the technical community. Here we mention several publications that are suitable for the possible extension of knowledge on ECT. We can recommend the book [2], which is very comprehensive. The book [53] is in principle sufficient for understanding the basic principles; the paper [1] can also be recommended. Among many book monographs, it is possible to mention [23] and also [54]. Both are written in a very understandable manner and the reader will not get lost in theorems, definitions and proofs that do not bring much information for practitioners and beginners. Description of specialized algorithms can be found in [53] and [38] (GA), [15] and [71] (PSO), [57], [56], [58] and [24] (DE), [73] (SOMA), [35], [32] and [65] (MA), [41] and [13] (SA), [45] (SS).

Popularly written books from the field of computers are much less represented than specialized expert books. Much information can be found on the Internet, however, one should mention that this source of information is not always reliable and one can also encounter untrue and misleading information.

There are many publications on the limits of computational technologies based on quantum physics. However, these publications are relatively very demanding on the knowledge from the field of quantum mechanics and mathematics. For extending the information indicated in this chapter, we recommend the already mentioned publications [8] and [50]. The substantial part of limits imposed by mass on processing and storing data is described in the first part [8]. The explanation is so understandable that even a reader at a high school level will understand it. In the paper [50], the relation between transfer channels and black holes is discussed.

You can also read in [2] on the representation of individuals, basic concepts of ETV and the properties of the test functions. Of course, there are other monographs and Internet sources providing this information, but we consider publications mentioned above as sufficiently representative.

References

1. Babu, B.: Evolutionary Computation - At a Glance. NEXUS, Annual Magazine of Engineering Technology Association, BITS, Pilani, 3–7 (2001)
2. Back, T., Fogel, B., Michalewicz, Z.: Handbook of Evolutionary Computation, Institute of Physics, London (1997)

3. Baluja, S.: Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, USA (1994)
4. Barricelli, N.A.: Esempi Numerici di processi di evoluzione. Methodos, 45–68 (1954)
5. Barricelli, N.A.: Symbiogenetic evolution processes realized by artificial methods. Methodos 9(35-36), 143–182 (1957)
6. Barricelli, N.A.: Numerical testing of evolution theories: Part I: Theoretical introduction and basic tests. Acta Biotheor. 16(1-2), 69–98 (1962)
7. Box, G.E.P.: Evolutionary Operation: A Method for Increasing Industrial Productivity. Appl. Stat. 6(2), 81–101 (1957)
8. Bremermann, H.: Optimization through evolution and recombination Self- Organizing Systems. In: Yovits, M., Jacobi, G., Goldstine, G. (eds.), pp. 93–106. Spartan Book, Washington (1962)
9. Bull, L., Kovacs, T.: Foundations of Learning Classifier Systems. Springer, Heidelberg (2005)
10. Carlson, E.: Doubts about Mendel's integrity are exaggerated. In: Mendel's Legacy, pp. 48–49. Cold Spring Harbor Laboratory Press, Cold Spring Harbor (2004)
11. Caruana, R., Schaffer, J.: Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In: Proc. 5th Int. Conf. on Machine Learning, Los Altos, pp. 153–161. Morgan Kaufmann, San Francisco (1988)
12. Castro, L., Timmis, J.: Artificial Immune Systems: A New Computational Intelligence Approach. Springer, Heidelberg (2002)
13. Cerny, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Opt. Theory Appl. 45(1), 41–51 (1985)
14. Chu, P.: A Genetic Algorithm Approach for Combinatorial Optimisation Problems. Ph.D. Thesis. The Management School Imperial College of Science, Technology and Medicine, London, p. 181 (1997)
15. Clerc, M.: Particle Swarm Optimization. ISTE Publishing Company (2009)
16. Coveney, P., Highfield, R.: Mezi chaosem a radem, Mlada fronta (2003)
17. Darwin, C.: On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life, 1st edn. John Murray, London (1859)
18. Dasgupta, D.: Artificial Immune Systems and Their Applications. Springer, Berlin (1999)
19. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, Berlin (1996)
20. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer, Heidelberg (2009)
21. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
22. Dreo, J., Petrowski, A., Siarry, P., Tailard, E.: Metaheuristic for Hard Optimization: Methods and Case Studies. Springer, Heidelberg (2005)
23. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Springer, Heidelberg (2007)
24. Feoktistov, V.: Differential Evolution - In Search of Solutions. Springer, Heidelberg (2006)
25. Fogel, B., Corne, W.: Evolutionary Computation in Bioinformatics. Morgan Kaufmann, San Francisco (2002)
26. Fogel, D.B.: Unearthing a Fossil from the History of Evolutionary Computation. Fundamenta Informaticae 35(1-4), 1–16 (1998)
27. Fogel, D.B.: Evolutionary computation: the fossil record. IEEE Press, Piscataway (1998)
28. Fogel, D.B.: Nils Barricelli - Artificial Life, Coevolution, Self-Adaptation. IEEE Comput. Intell. Mag. 1(1), 41–45 (2006)

29. Fogel, L., Owens, J., Walsh, J.: Artificial Intelligence through Simulated Evolution. John Wiley, Chichester (1966)
30. Friedberg, R.M.: A learning machine: Part I. IBM Journal Research and Development 2, 2–13 (1958)
31. Glover, F., Laguna, M.: Tabu Search. Springer, Heidelberg (1997)
32. Goh, C., Ong, Y., Tan, K.: Multi-Objective Memetic Algorithms. Springer, Heidelberg (2009)
33. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company Inc., Reading (1989)
34. Haupt, R., Haupt, S.: Practical genetic algorithms, 2nd edn. John Wiley & Sons, USA (2004)
35. Hart, W., Krasnogor, N., Smith, J.: Recent Advances in Memetic Algorithms. Springer, Heidelberg (2005)
36. Hinterding, R., Gielewski, H., Peachey, T.: The nature of mutation in genetic algorithms. In: Eshelman, L. (ed.) Proc. 6th Int. Conf. on Genetic Algorithms, Los Altos, pp. 70–79. Morgan Kaufmann, San Francisco (1989)
37. Holland, J.: Adaptation in natural and artificial systems. Univ. of Michigan Press, Ann Arbor (1975)
38. Holland, J.: Genetic Algorithms. Sci. Am., 44–50 (1992)
39. Ilachinski, A.: Cellular Automata: A Discrete Universe. World Scientific Publishing Company, Singapore (2001)
40. Jones, T.: Evolutionary Algorithms, Fitness Landscapes and Search, Ph.D. Thesis, University of New Mexico, Alburquerque (1995)
41. Kirkpatrick, S., Gelatt Jr., C., Vecchi, M.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)
42. Koza, J.: Genetic Programming. MIT Press, Cambridge (1998)
43. Koza, J.: Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University, Computer Science Department, Technical Report STAN-CS-90-1314 (1990)
44. Koza, J., Keane, M., Streeter, M.: Evolving inventions, pp. 40–47. Scientific American (2003)
45. Laguna, M., Martí, R.: Scatter Search - Methodology and Implementations in C. Springer, Heidelberg (2003)
46. Lampinen, J., Zelinka, I.: Mechanical Engineering Design Optimization by Differential Evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 127–146. McGraw-Hill, London (1999)
47. Lampinen, J., Zelinka, I.: Mechanical Engineering Design Optimization by Differential Evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization. McGraw-Hill, London (1999)
48. Langdon, W.: Genetic Programming and Data Structures. Springer, Heidelberg (1998)
49. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Dordrecht (2002)
50. Lloyd, S., Giovannetti, V., Maccone, L.: Physical limits to communication. Phys. Rev. Lett. 93, 100501 (2004)
51. Marik, V., Stepankova, O., Lazansky, J.: Artificial Intelligence III. Czech (ed.) Artificial Intelligence III. Academia, Praha (2001)
52. Mendel, J.: Versuche über Plflanzenhybriden Verhandlungen des naturforschenden Vereines in Brünn, Bd. IV für das Jahr. Abhandlungen, 3–47 (1865); For the English translation, see: Druery, C.T., Bateson, W.: Experiments in plant hybridization. Journal of the Royal Horticultural Society 26, 1–32 (1901), http://www.esp.org/foundations/genetics/classical/gm-65.pdf

53. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin (1996)
54. Michalewicz, Z., Fogel, D.: How to Solve It: Modern Heuristics. Springer, Berlin (2000)
55. O'Neill, M., Ryan, C.: Grammatical Evolution - Evolutionary Automatic Programming in an Arbitrary Language. Springer, Heidelberg (2003)
56. Onwubolu, G., Babu, B.: New Optimization Techniques in Engineering. Springer, New York (2004)
57. Price, K.: An introduction to differential evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimisation, pp. 79–108. McGraw Hill, International, UK (1999)
58. Price, K., Storn, R., et al.: Differential Evolution - A Practical Approach to Global Optimization. Springer, Heidelberg (2005)
59. Read, R.C.: Coding of Unlabeled Trees. In: Read, R. (ed.) Graph Theory and Computing. Academic Press, London (1972)
60. Rechenberg, I.: (1971) Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis), Printed in Fromman-Holzboog (1973)
61. Reeves, C.: Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications, Oxford (1993)
62. Rego, C., Alidaee, B.: Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search. Springer, Heidelberg (2005)
63. Russell, Norvig, S.J., Peter: Artificial Intelligence: A Modern Approach, 2nd edn., pp. 111–114. Prentice Hall, Upper Saddle River (2003)
64. Schwefel, H.: Numerische Optimierung von Computer-Modellen, PhD thesis (1974); Reprinted by Birkhäuser (1977)
65. Schönberger, J.: Operational Freight Carrier Planning, Basic Concepts. In: Optimization Models and Advanced Memetic Algorithms. Springer, Heidelberg (2005)
66. Telfar, G.: Acceleration Techniques for Simulated Annealing. MSc Thesis. Victoria University of Wellington, New Zealand (1996)
67. Turing, A.: Intelligent machinery, unpublished report for National Physical Laboratory. In: Michie, D. (ed.) Machine Intelligence, vol. 7 (1969); Turing, A.M. (ed.): The Collected Works, vol. 3, Ince D. North-Holland, Amsterdam (1992)
68. Vesterstrom, J., Riget, J.: Particle Swarms (May 2002), Dostupny z www.evalife.dk/publications/JSV_JR_thesis_2002.pdf (cit.10.2.2007)
69. Von Neumann, J.: The computer and the brain. Yale University Press, New Haven (1958)
70. Wolpert, D., Macready, W.: No Free Lunch Theorems for Search, Technical Report SFI-TR-95-02-010, Santa Fe Institute (1995)
71. Li, X.: Particle Swarm Optimization - An introduction and its recent developments (2006), www.nical.ustc.edu.cn/seal06/doc/tutorial_pso.pdf (4.10.2006) (cit. 20. 2. 2007)
72. Zelinka, I.: Artificial Intelligence in problems of global optimization. Czech (ed.) BEN, Praha (2002) ISBN 80-7300-069-5
73. Zelinka, I.: SOMA - Self Organizing Migrating Algorithm. In: Onwubolu, Babu, B. (eds.) New Optimization Techniques in Engineering. Springer, New York (2004)
74. Zvelebil, M., Jeremy, B.: Understanding Bioinformatics. Garland Science (2007)