EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

Bioinspired Optimization Techniques

Assoc. Prof. Roman Šenkeřík

1.4.2019 – 31.1.2020

Development of Research-Oriented Study Programs at FAI (VyStuP FAI)

CZ.02.2.69/0.0/0.0/16_018/0002381

CONTENTS

INTRODUCTION

The text you are getting into your hands deals with the state of the art in the research are of bioinspired optimization methods. Primarily, it serves to students of the doctoral study program at the Faculty of Applied Informatics, Tomas Bata University in Zlín. It provides an overview of suitable techniques for solving interdisciplinary optimization problems related to the topic of research in the student's dissertation.

Bioinspired optimization methods represent a relatively broad field of science that includes areas such as evolutionary algorithms, principles of swarm intelligence and derived swarm algorithms, methods for multicriteria / multiobjective optimization, methods of symbolic regression and evolution of symbolic structures. The emphasis will be placed on understanding the principles of individual algorithms, internal population dynamics, as well as on optimization, adaptation and learning techniques for the control parameters of these algorithms. An equally important part of research in this field, it is the testing and benchmarking using the state-of-the-art benchmark sets and platforms before deployment in practical applications and research tasks.

The topics presented in the individual sections also deal with the possibilities of hybridization and modification of the above-mentioned original techniques "tailored" to different classes of problems. At present, in the areas of human acting, industry or the medical industry, very time consuming and demanding tasks with both, more or many objectives and the space of high dimensions (large-scale problems) of individual optimized parameters are processed, The algorithms need to be more or less modified to be able to use them correctly in the mentioned tasks. It is also necessary not to forget on modifications of algorithms within the currently solved application problems from the areas of not only continuous but also discrete space with permutation, combinatorial or similar derived combinations. Researchers are also involved in accelerating computations, including, for example, a very sophisticated assisted optimization approach for computational demanding real optimization models (surrogate assisted models).

Before the optimization algorithms from the soft computing area are used, they must be properly tested. Many experts are dedicated to benchmarking, appropriate approaches and the preparation of benchmark sets from various point of views, ensuring high quality and

robustness of the winning algorithms. The text also deals with theoretical aspects of this modern field, such as convergence and runtime analysis.

In the closing chapters, the student is acquainted with a number of interesting solved applications with a vision to future directions of the research in this area and with the use of bioinspired computational methods in various areas of human activities.

To deal with the area in more deep and detailed way exceeds the scope and focus of the text. The text is conceived as an offer of possible topics in the field of bioinspired optimization methods, which the student can use during his / her doctoral study and preparing the dissertation. For a detailed study of the issue, it is possible to use the cited literature, which also serves as recommended for completeness of further knowledge.

1 BIOINSPIRED OPTIMIZATION METHODS

There is no doubt that Evolutionary Algorithms (EA) are the most popular metaheuristics today. In general, in the field of informatics, EA or so-called evolutionary computing techniques are a sub-area of artificial intelligence, which allows to solve complex optimization problems. Evolutionary techniques are inspired by Darwinian principles of natural selection as a key mechanism of evolution and Mendel's law of heredity.

Although ongoing research has brought many efficient and robust evolutionary algorithms, scientists have to deal with the well-known No Free Lunch Theorem [256], [79], which states that there is no universal algorithm to be able to solve all the problems. This theorem then forces researchers and users to test different algorithms, their combinations, adaptations and parameter settings leading to good results for a wide range of task classes. The importance of finding a well-functioning algorithm grows with the increasing complexity, dimensionality and number of criteria in current optimization tasks. An overview of this area can be found in the following publications: [83], [133], [213], [9].

1.1 History and predecessors of evolutionary computation techniques

The motivation that led to the development of EA is relatively simple. The nature has managed to adjust to many rigmaroles during its existence – it handled to optimise its progress. There are known mechanisms which are simple and very clever in numerous ways. The human stands on the highest level with his brain which can be considered as the best "solver" of problems known in nature. It has served also as an inspiration for artificial neural networks and solving of complex and difficult tasks from the various areas of the human acting. All in nature, from the lowest levels up to the mentioned human brain, has been developed step by step. It has come through the evolution which is the key element for EA. The principle of evolution has thus become an idea for the creation of algorithms whose task is to find the best solution.

As a predecessor of evolutionary algorithms, we can consider a number of so-called iterative heuristics, where the solution of the optimization problem is gradually improved in iterations (but without direct "evolutionary" interactions between individual solutions). Undoubtedly, the most well-known heuristics include local search, hill climbing algorithm and a number

of other algorithms based on a similar principle, like tabu search, or simulated annealing [238].

The history of EA dates back to the 1960s, when L. J. Fogel introduced Evolutionary Programming in 1966 [98], [9] as a method for the evolutionary design of finite state machines. I. Rechenberg (in 1973) developed Evolutionary Strategies which were further expanded by H. P. Schwefel (in 1981) [8], [87], [14] as an optimization method in real numbers domain. The most important milestone was the introduction of Genetic Algorithms by J. H. Holland in 1975 [118], [104] as a universal method for solving optimization problems.

1.2 General definition of optimization problem

Formally, optimization problems can be formulated as follows:

The goal is to find such a solution (vector) $X = (x_1, x_2, \ldots x_D)$ that minimizes or maximizes the function $f_{cost}(X)$, due to functional constraints and the constraints of individual arguments (attributes) of the vector (solution).

Each optimization problem can then be viewed as a geometric problem, in which the lowest (minimum) or highest (maximum) position on the surface lying in $(D + 1)$ - dimensional space is searched. For this surface, the term "*space of possible solutions*" or "*hyper-plane*" of a given optimization problem is sometimes used. The number of dimensions D is given by the number of optimized attributes of the objective function.

In the case of EA, the quality of possible solutions (individuals) within a population is defined by the objective function (or a fitness function in the case of GA), with attributes representing the values of each solution. The best solution is the global extreme - the maximum or minimum of the objective function [119].

1.3 Basic terms and critical situations

The use of evolutionary algorithms can be generally characterized as the following sequence of operations:

(Randomly) generate the initial population of solutions;

Evaluate each solution in the population;

while termination condition not satisfied

Select highly-fit solutions for reproduction (parents);

Recombine pairs of parents (crossover);

Mutate the resulting offspring;

Evaluate the new solutions;

Replace the low-fit population members with the new solutions;

end while;

At the beginning, the population is usually generated randomly (from a uniform distribution). There are also specific cases where we use a different distribution of a random generator, other mathematical / physical phenomena / functions. Alternatively, it is possible to "inject" certain previously known promising solutions into the population. However, there must not be too many of these solutions, or parametrically "close to each other" - there is a risk of loss of population diversity and rapid convergence to "injected" local extremes [166], [130].

The arguments of the objective function (attributes of an individual) are defined within the limits of certain values, which brings the need for appropriate control of these arguments during the evolutionary process. When working with evolutionary algorithms, it is almost constantly the case that individuals tend not only to cluster around possible solutions, but also it also happens that they leave these areas and may find solutions that are meaningless or not-feasible. To avoid such situations, there are methods that are part of the evolutionary algorithm and which maintain the population by replacing / editing individuals within the defined search space of possible solutions. However, there are exceptions (PSO algorithm), where the original design of the algorithm does not count with controlling the boundaries of the search space (in the sense of modifying the individual) and relies on the spontaneous "return" of individuals back. However, the objective function is not calculated outside the

search space. The most commonly used methods (so-called boundary strategies) are: *Random, Clipping, Reflection, Periodic*. More information can be found in the publications: [258], [115], [278].

Real optimization problems are often highly constrained, when solutions in the selected intervals of arguments are unacceptable for various reasons. There are two approaches to resolving the situation. In the case of hard-constraints, an individual who would be in the restricted area is deleted from the population and replaced with a new randomly generated individual lying in the allowed area. The second option is so-called soft-constraints. In this case, the individual lying in the forbidden area is not deleted, but is disadvantaged by modifying the value of the objective function. E.g. in the case of a minimization problem, it is additionally penalized by adding a high value ($CF = CF + 10^6$), i.e. disadvantaged within the population for selection process. This method is preserving the evolutionary history of the individual. More on this issue can be found in the publications: [164], [71].

2 CLASSICAL EVOLUTIONARY ALGORITHMS

In general, classical evolutionary algorithms use iterative process representing the development of a population. This class of algorithms is characterized by utilizing special operators to guide the evolutionary process, such as mutation, crossover and others to find an ideal solution. Classical evolutionary algorithms include, in particular, Evolutionary strategies (ES) [14], Genetic Algorithms (GA) [103] and Differential Evolution (DE) [204]. The basics of these algorithms, including modern versions and links to survey studies, are further mentioned in this chapter.

2.1 Evolutionary Strategies

In the 1960s, P. Bienert, I. Rechenberg, and H. P. Schwefel developed an algorithm called the "Evolutionary Strategies" [10]. ES were gradually introduced in several variants - strategies. After the simplest so-called *two-membered ES*, further *multi-membered ES*, *recombinant (recombinative) ES*, and *self-adaptive ES* were introduced. At the turn of the millennium and later, research focused mainly on complex modifications of adaptation mechanisms. Evolutionary strategies differed from genetic algorithms at the moment of introduction mainly in that ES used representations of individuals in real numbers (while generic GA in binary), and ES did not use crossover operators, ES only used selection and mutation operators.

ES use the parameter μ to define the number of individuals from the group of parents, the parameter λ for the offspring, as well as the symbols „+“ and „„, for the strategy of selection of the solution(s) to the new population. The symbol + means that the best solutions from the union of parent and offspring populations are selected for the new population. The symbol "," means that the best solutions are selected for the new population only from the offspring population. The resulting syntax is therefore (μ + λ)-ES or (μ, λ)-ES.

Two-membered ES: (1+1)-ES, represent the simplest version of ES, which works with one individual-parent, from which a new offspring is created using the so-called Gaussian mutation operator. It is therefore a very simple iterative process that can be programmed very easily. The only influence on the efficiency of this variant of ES is the mutation, which is performed here by adding randomly generated numbers using the normal distribution N

(0, σ) to the parent (existing solution). The main input parameter is therefore the standard deviation σ. Various experiments and theoretical studies have established certain rules for the value of this deviation [13], or the golden rule 1/5 can be used.

Multi-membered ES: (μ+λ)-ES or (μ, λ)-ES, represent the so-called population ES and on many optimization problems they have demonstrated higher efficiency and quality of results than (1 + 1)-ES. The (μ + λ)-ES version can be described as ES with elitism, because both parents and offspring are selected for the new μ-size population on the basis of the achieved quality given by the evaluation of the objective function. For the (μ, λ)-ES version, elitism is not applied. But it must always be true that λ ≥ μ. It is also possible to create a hybrid combination of both versions with elitism (a pair of parents + the addition of μ from the offspring). The paper [228] describes a version with the parameter called *life span*, which indicates the maximum lifetime of the parent in the number of iterations. After exceeding this value, it can no longer be selected for a new generation (regardless of its quality in relation to other parents / offspring).

Recombinant (recombinative) ES: *(*μ/ρ+λ*)*-ES, ES, represent an approximation to the principles of genetic algorithms, as they contain functionality mimicking the crossover operation. Firstly, so-called recombinant (something like an intermediate stage of the offspring) is created by recombination of several parents, followed by a mutation process and the creation of λ offspring. The parameter ρ then determines the number of parents involved in the recombination process, and there are several ways to do that [82].

Although many decades have passed since the introduction of various versions of the EC, the one-member ES has recently experienced a renaissance thanks to a number of experts from the field of theory of evolutionary algorithms and their focus on research of runtime analysis and the complexity of evolutionary optimization. One-member ES in conjunction with simpler discrete optimization problems (Leading Ones, One Max) allow the derivation of a number of proofs, which would probably be very difficult or impossible to derive, mainly due to the presence of random mutation, recombination, elitism in the population and

impossibility of unambiguous tracking of solutions in time (iterations) [188], [148], [185], [147].

The development of evolutionary strategies is quite clearly summarized in [14], [42].

Adaptive Variants of ES

Adaptive ES represent an effective tool for blackbox optimizations, and are therefore closest to efficient and robust evolutionary algorithms in their properties and principle. The reason for the emergence of adaptive ES is obvious, mainly due to the limitations on the calculation of the optimal value of the standard deviation (see above), when the area of the optimal solution may not be known in advance. Furthermore, for multi-member ES it is necessary to consider the ratio of parents and offspring, which also affects the running of the algorithm, and therefore it is appropriate when the algorithm itself adjusts its control parameters during the optimization process, thus increasing the chances of achieving an optimal solution faster. In addition, the position of a global optimal solution can change its position over time, which means that even the purpose function itself changes dynamically over time.

There are currently a number of approaches to adaptation in the ES, the most frequently studied and applied approaches are listed below.

- Hyper heuristics based on the hierarchical arrangement of the ES, where the "master" ES optimizes the parameters of the "slave" ES [227],
- the direct use of adaptation and learning mechanisms for the control parameters of the algorithm [12], [68],
- and last but not least, the most studied method using covariance matrix adaptation (the method uses the abbreviation CMA) to determine the probability distribution of the mutation [110]. This type of ES is suitable for solving optimization problems, where the parameters of an individual are (can be) mutually correlated. The CMA-ES variants also achieved very good results in various benchmark competitions [113].

The research field around CMA-ES was summarized recently in survey paper [111].

2.2 Genetic Algorithms

The basic idea of genetic algorithms is a direct analogy with evolutionary processes taking place in biological systems. Genetic algorithms (GA) are one of the best-known Evolutionary optimization techniques.

Most of the terminology (population, individual, generation) of GA was therefore taken from biology. A typical feature of classical GA is that they work with binary coding of individual parameters. In genetic algorithms, an individual's quality is referred to as fitness, which is a positive number assigned to an individual's genetic information. When using GA, an individual's attributes are called genes (they determine a point in the searched space). The genotype is then called a set of chromosome parameters. All genes then give a chain, which we call a chromosome (i.e. the whole individual). The whole genetic information of an individual (the chromosome already mentioned) is usually expressed by a bit string, usually of constant length. A population is then a set of such chromosomes [233].

An individual can also be represented by a string of numbers, or a string of letters of the alphabet (character representation), or a tree of some objects (e.g. mathematical functions or programming language commands). This type of representation is then characteristic of so-called genetic programming. The advantages of binary representation are mainly easy implementation of GA operators (crossover of binary strings, bit mutation), and easy implementation of solving discrete optimization problems (combinatorial problems such as knapsack problem: 0 = object is not selected, 1 = object is selected). The disadvantages of binary representation are the search for a solution in a continuous region (increase in chromosome length) and problems with the representation of integer numbers in a binary string: adjacent numerical values are represented by strings that can differ substantially, even in all bits, e.g. the decimal value 7 is binary (0111), the following numeric value 8 is in binary (1000). A slight change in the solution (mutation) is then difficult to implement. The solution is Gray's code, or a direct representation of individuals in real numbers. However, variants of genetic algorithms with the representation of an individual as a vector of real numerical values require completely different crossover and mutation operations than in the case of a binary representation.

Integer representation has advantages in permutation optimization problems. For example, in the travelling salesman problem (TSP) problem, the chromosome (individual) directly represents one of the possible routes.

2.2.1 Basic Versions of GA

GA workflow is characterized by a sequence of three operations: selection, crossover and mutation. Parent selection is a key operation in GA affecting the speed of convergence and the quality of the final solution. The selection is performed using the value "fitness", which is the recalculated value of the objective function, most often in the following ways:

- roulette rule,
- rank selection,
- tournament selection.

The right choice of selection method then allows to maintain the diversity of the population. Due to genetic operators, the best individuals may be lost in crossover or mutation. Elitism solves this problem by selecting the strongest (elite) individuals, who are inserted directly into the next generation, before the actual crossover and mutation takes place. This will ensure that in each generation, the best individual in the worst case will be of the same quality as in the previous generation. Sometimes the relation $k = (1 - Pc)\cdot N$ is used, where Pc represents the probability of crossing (but this is not always the rule, of course it applies to higher values of Pc).

The crossover operation is the most important operator, because it is the essence of genetic algorithms - a combination of different solutions. Crossover depends on the implemented representation of individuals, but always takes place between two parents. With a classical binary string, there are several types of crossings (single-point, two-point, multipoint and the least used uniform).

The mutation is usually implemented in genetic algorithms as a change in the value of a random selected bit in the chromosome, i.e. the value 0 changes to 1, the value 1 to 0. The mutation ensures that genetic information can be newly created in the population, and was not present in previous generations, or it ensures recovery of genetic information lost during the development of the solutions. Using a mutation, the algorithm can then escape from the local minimum region.

With respect to the No free lunch theorem (NFL), it is difficult to define which combination of selection, crossover, mutation, including parameter setting techniques is the most appropriate. The issues of the above areas are then summarized in detail in the following overview studies. Moreover, with regard to the existence of evidence of convergence, GA is still intensively studied from a theoretical point of view [40], [187], [186], [67], [66].

2.2.2 Versions of GA

There are a huge number of different versions of GA today. Below are descriptions and references to the most important and most studied versions.

microGA: genetic algorithms with very small populations. They are often used as a local search technique. The number of individuals is most often set to 5 [143].

Steady state GA: in "steady state" GA, unlike generic GA, generations overlap. Thus, new individuals represent only a certain part of the whole population. In one time (iteration) a whole new generation is not created - that would replace the old one, but only a few individuals are replaced. It is necessary to determine which individuals are to be replaced by new ones (e.g. replacing the individuals with the worst fitness - but there is a risk of the loss of diversity, and other approaches) [237], [156].

MessyGA: algorithm designed to accelerate convergence on selected multimodal problems. It has a variable chromosome length and genes do not have a fixed position [105], [129].

Hybridní GA: they combine the effectiveness of standard GA with the speed of local search techniques. First GA finds a suitable area and the local search technique then (theoretically) finds an extreme. Hybrid GA uses a variety of possible approaches: GA runs until it begins to "slow down", then a local search procedure is activated assuming that the GA is very close to the global extreme, or the local search procedure run after each x iteration and passes its best solution or several solutions as chromosomes to the population [85], [183].

2.3 Differential Evolution

The Differential Evolution (DE) algorithm was developed in 1995 by R. Storn and K. Price [234] and formed a basis for a family of successful algorithms for continuous optimization. The broad research field around DE was summarized recently in two surveys [43], [46].

DE is a population-based optimization method that works on real-number-coded individuals. DE is quite robust, fast, and effective, with global optimization ability. It does not require the objective function to be differentiable, and it works well even with noisy and time-dependent objective functions.

The core idea of the algorithm is defined by differential mutation and discrete recombination. The algorithm has also slightly unconventional "evolution" workflow. Firstly, the mutation operation is performed, followed by crossover and selection. Also, the minimum of 4 donor vectors (solutions/individuals) are required as "parents" for new solution creation.

DE has achieved its popularity among EC researchers due to its consistent performance and high ranks in various IEEE CEC (Congress on Evolutionary Computation) competition on real parameter optimization in complex scenarios like, constrained, multi-modal, multi- objective etc.

2.3.1 Basic Definitions

There are essentially four inputs to the algorithm (omitting D, which is the size of the problem), G_{max} is the maximum number of generations (or can be replaced by other termination criterion), NP is the total number of solutions, F is the scaling factor of the solution and CR is the crossover rate.

The individuals in the population are combined following basic ideas of evolution in order to create improved solution for the next generation. This process is repeated iteratively until the stopping criterion is met. DE is using simple elitism-based selection mechanism, where each following generation consists either of better solutions than those in previous generations, or the solutions with the same quality are accepted for the next generation, but with different attributes structure. The following is an example for the most basic strategy called DE/Rand/1/Bin.

In mutation, to produce a mutated vector $\boldsymbol{v}$, the attribute vectors of three randomly selected individuals are combined in simple vector operations using a control parameter scaling factor F (see eq. 1).

$$\boldsymbol{v}_{i,G} = \boldsymbol{x}_{r1,G} + F_i\left(\boldsymbol{x}_{r2,G} - \boldsymbol{x}_{r3,G}\right) \tag{1}$$

In the binomial version of crossover operation, a trial vector ***u*** is created by selection of attributes either from mutated vector ***v*** or the original vector ***x*** (current active individual), based on the crossover rate *CR*.

Finally, in the selection, the quality of trial vector is evaluated by an objective function and compared to the quality of original vector and based on the above-mentioned elitism mechanism, the selected individual is placed into the next generation.

From the basic description of the DE algorithm, it follows that F and CR together make the internal tuning parameters for the heuristic. It was shown in [46], that the setting of these parameters is crucial for the performance of DE. Fine-tuning of the control parameter values is a time-consuming task and, therefore, many state-of-the-art DE variants use self-adaptation in order to avoid this cumbersome task. Given the results achieved in recent years and the popularity, a more detailed description below focuses on two selected versions: jDE and SHADE.

2.3.2 Adaptive Parameter Propagation Strategy jDE

The jDE algorithm was firstly proposed by J. Brest in 2006 [25], introducing the simple adaptive parameter propagation strategy. The idea is very simple: the generated ensemble of two control parameters Fi and CRi is assigned to each i-th individual of the population and survives with the solution if an individual is transferred to the new generation. If the newly generated solution is not successful, i.e., the trial vector has worse quality than the compared original active individual; the new (possibly reinitialized-mutated) control parameters values disappear together with not successful solution.

It is necessary to highlight, that jDE uses similar “evolutionary logic” also for control parameters as the generic DE algorithm. Before the evolutionary operations with solutions are performed, the both aforementioned DE control parameters may be randomly mutated with predefined probabilities. If the mutation condition happens, a new random value of CR ∈ [0, 1] is generated, and possibly also thevalue of F is mutated in the similar defined bounded range. These new control parameters are then used for new solution creation. The initialization of values of F and CR for the first generation is designed to be either fully random with

uniform distribution for each individual in the population or can be set according to the recommended values in the original jDE paper, where also typical settings for control parameters bounds and re-initialization/mutation probabilities are also given [25].

Recently, the jDE version gained visibility thanks to the shared victory in 2019 IEEE CEC competition on real-parameter benchmark set (called 100 digits competition). The winning strategy was called jDE100 and used simple mechanisms of co-evolution and population structuring. Details are given in following research paper [28].

2.3.3 Modern Adaptive Variants SHADE/L-SHADE

Since 2013, an algorithm developed by Tanabe and Fukunaga -- Success-History based Adaptive Differential Evolution (SHADE) [240] represents the connection link between all the best recent DE variants [4]. This algorithm automatically adapts its both control parameters (*F* and *CR*), during the run according to the progress in solution of the given optimization problem, it also incorporates new effective mutation strategy with external archive of inferior solutions from JADE [276]. The SHADE algorithm was placed 3rd in the CEC2013 competition, and followed next year by the version with added linear decrease in population size (the algorithm was titled L-SHADE) [241]. L-SHADE won the CEC competition in 2014, and the winners of the following years were all based on this algorithm: SPS-L-SHADE-EIG -- winner of 2015 competition [108], LSHADE_EpSin -- joint winner in 2016 [7], jSO -- announced winner in 2017 [27], and DISH - join winner in 2019 [267], [253].

The versions of SHADE, especially those referred to as L-SHADE, are probably among the most effective evolutionary algorithms [201], [200]. All previously mentioned algorithms share the same idea of balancing between exploration and exploitation abilities, and they try to achieve it in various ways.

2.3.4 State of the art - Versions and strategies of DE

The continuing research in the DE area provides a variety of improvements to the original algorithm [205], [44], [170], [160], [26], [257], [207]. A few major research directions along this track include parameter adaptation based on success history (like JADE [276], self-adaptive DE (SaDE) [205], SHADE [240]), enhanced crossover and mutation techniques

(e.g. MDE-pBX [120], Pro-DE [89], crossover based on eigen vector in DE [107], generating a set of candidate offspring by multiple mutation strategies and criteria to select the most suitable one [280] etc.), ensemble principles or combination of different offspring generation techniques (e.g. EPSDE [160], multiple variant coordination framework [277]) and improved parameter tuning [242], competitive principles [29], population diversity analysis [202]. In a recent article [102], authors have shown that stochastic use of previously identified successful difference vectors can lead to significant performance enhancement in DE framework. Recently, [155] proposed an adaptive DE variant which incorporated historical experience of the population and heuristic information of individuals to enhance parameter adaptation process. Instead of progress in objective function, the distance between individuals is used for parameter adaptation in Db-SHADE, Db_L-SHADE, and DISH versions [253], keeping the population diversity at higher levels in critical first part of the optimization process, preventing the premature convergence of the algorithm.

As parameter control is one of the important aspects in evolutionary algorithms [128], and being highlighted experimentally in DE as well [268], the focus of many research papers is on improvement and analysis of an aspect of DE parameter control, specifically the SHADE variants that are currently leading in the DE progress [4].

While new enhancements of above-mentioned DE algorithms add new mechanism or auto-tuning of parameters, there are also more aspects that needs to be addressed by different DE application domains, and those are covered in respective surveys on these domains [175], [45], [165] ,[279] , [199].

Furthermore, theoretical analyses in support of DE also exist, like [266], [191].

3 SWARM INTELLIGENCE

Modern methods of global optimization include original techniques and methods inspired by the behavior of animals in swarms. Many animal swarms show a phenomenon called collective intelligence, in which a swarm acts as a whole without having any central control. Animal swarms can then effectively find food and defend themselves against predators, even though they have no decision-making central node. Although individuals are guided solely by information from their immediate surroundings, it is the sophisticated way in which individuals communicate with each other that allows the entire system to function optimally without individuals knowing the group's goals. Thus, swarm algorithms mimic the observed behavior of swarms of various animals, from ant colonies to wolf packs. These models of behavior are then more or less successfully transformed and used in the optimization of various real problems [81], [19], [131], [18], [17].

The search for further inspiration and rapid development was accelerated mainly by the success of Particle Swarm Optimizer (PSO) algorithm [132] and the Ant Colony Optimization (ACO) [73] at the turn of the millennium.

Each introduced swarm algorithm has its own original pattern of behavior inspired by nature. It is assumed that the observed model of behavior has already undergone an imaginary optimization process of natural selection and evolution, and can therefore be considered successful. The goal of swarm algorithms is a suitable simulation of this exemplary behavior. However, in many cases it is necessary to define the necessary simplifications of the model. Swarm algorithms basically works as multiagent systems, where agents represent individual solutions to the optimization problem. Depending on the modeled behavior, the way of close communication and reaction of agents differs. Swarm algorithms often use the so-called Lévy flights [161], [223], which best characterizes the movement of many animals (and especially insects). During their flight, the animals demonstrate the ability to fly even a very long route with variable time between individual stops and direction. Many swarm algorithms use Lévy's flight to generate random numbers, as this method allows a random shift, the step of which is taken from Lévy's stable distribution with infinite variance.

3.1 PSO Algorithm

The PSO algorithm is inspired in the natural swarm behavior of flock birds and fish. It was introduced by R. C. Eberhart and J. Kennedy in 1995 [132] as an alternative to other successful EAs, such as GA, and DE.

PSO proved itself to be able to find very good solutions for many optimization problems. The term "swarm intelligence" refers to the capability of particle swarms to exhibit surprising intelligent behavior assuming that some form of communication (even very primitive) can occur among the swarm particles (individuals).

Each particle in the population represents a possible solution of the optimization problem defined by its cost function. In each generation, a new location (combination of attributes) of the particle is calculated based on its previous location and velocity (or "velocity vector" as velocity may be different for each attribute - dimension).

3.1.1 Basic variant of PSO

The main principles of the PSO algorithm and its modifications are given in [132], [222]. At the beginning, PSO is initialized by a population of randomly located particles (solutions) and velocity vectors, which indicates the direction of individual particle movement in the next step, and they are generated for each particle individually. The individual with the best cost function value saves its current position in the global memory of the population and within the whole population, it is called *gBest* (global best). Meanwhile, every individual particle checks whether its current position is better than its previous position. If so, this new better position is saved in its own particle memory and it is referred as *pBest* (personal best). After *gBest* and *pBest* are calculated, all particles adjust its velocity and subsequent changes the position according to simple PSO formula (2), which defines the velocity vector update for the next iteration. The equation contains three different trends of movement. Particles tend to go by their own way, or to return to their best position or to adaptively follow the particle with the best value in the population.

$$v_{ij}^{t+1} = w \cdot v_{ij}^{t} + c_1 \cdot r_1 \cdot \left(pBest_{ij} - x_{ij}^{t}\right) + c_2 \cdot r_2 \cdot \left(gBest_j - x_{ij}^{t}\right) \quad (2)$$

where: $v(t+1)$ – new velocity of particle, $v(t)$ – current velocity of particle., c_1, c_2 – priority factors, *pBest* – best solution found by particle, *gBest* – best solution found in population, $x(t)$ – current position of particle, *Rand* – Random number, from the interval <0,1>.

Priority factors c_1, c_2 have the following influence to the movement of particles. Priority factor c_1 gives a preference to return to the best own position (*pBest*), whereas, the priority factor c_2 tends to shift particles in the direction of the best value in the population (*gBest*). The main PSO disadvantage is the rapid acceleration of particles, which causes them to abandon the defined area of interest. For this reason, several modifications of PSO were introduced to handle this problem. The inertia weight strategy [222] was firstly introduced in 1998 in order to improve the local search capability of PSO. Several modifications of inertia weight strategy are well described in [177]. Inertia weight is designed to influence the velocity of each particle differently over time. In the beginning of the optimization process, the influence of inertia weight factor w is minimal. As the optimization continues, the value of w is decreasing, thus the velocity of each particle is decreasing, since w is always the number < 1 and it multiplies the previous velocity of particle in the process of new velocity value calculation. This approach represents the simple standard for PSO utilization.

3.1.2 Modern variants of PSO

Since 1995, many modern modifications of PSO have been introduced. Below the list of state-of-the-art versions follows.

Heterogeneous particle swarm optimizers (HPSO): It allow particles to use different update equations, representing different types of behaviors, within the swarm. Dynamic HPSO allows the particles to change their behaviors during the search. This method alter the exploration/exploitation balance during the search based on the simple learning mechanism (fk-HPSO) [173]. Usually following pool of six behaviors is implemented:

- TVIW-PSO - time variant inertia weight with constant acceleration factors.
- TVAC-PSO - in this model, the values of c1 and c2 change linearly. The c1 is decreasing and c2 increasing therefore the algorithm has good exploration and exploitation ability.

- sPSO - social behavior favoring model.
- cPSO - cognitive behavior favoring model.
- modBB-PSO - is able to escape from local minima as well as explore the pBest areas. The algorithm is more likely to select pBest area for exploration in the end phase.
- QSO – it uses a linear decrease of "cloud size" around the gBest. Therefore, the particles are forced to move in shrinking space in every dimension.

The orthogonal learning PSO – OLPSO: this method consists of following simple steps. Firstly, an orthogonal matrix is created according to a set of predefined rules. Then, the trial solutions are created by crossover of *pBest* and *gBest* according to data in the matrix, followed by the evaluating of trial vectors and storing the best as x_b. Next step is the evaluation of the most beneficial dimensional components and subsequent creating of trial vector x_p according to the results. The quality of x_b and x_p is then compared, and the winner is used as a guidance vector in formula for updating the velocity [275]. Improved versions are described in [3], [206].

Comprehensive learning particle swarm optimization (CLPSO): It was firstly introduced in 2006 [151]. CLPSO represents an enhanced version of the original PSO algorithm. CLPSO utilizes the idea of using a personal best solution of other members of the swarm as a way to improve the search quality. Based on a learning probability, the movement of a particle in each dimension might be based on *pBest* location (dimension component) of another particle. At least one-dimensional component is always used from another particle than self.

Attractive and Repulsive PSO (ARPSO): This algorithm was introduced in 2002 by J. Riget and J. S. Vesterstrøm [210]. This modification was created due to the premature convergence of generic PSO on complex multimodal optimization problems. This phenomenon arises due to the reduction of population diversity in the searched area, which leads to the absolute stagnation of the value of the objective function of the entire swarm. To prevent this phenomenon, a mechanism for controlling the diversity of individuals (population of solutions) was introduced and the equation of calculating the new velocity for

the next iteration was modified. A simple variable (*dir*) indicating the direction of particle motion was introduced. This can only take the values 1 (normal behavior of the individual as in a generic PSO - attractive force), or -1 (repulsive direction of movement of the individual).

Heterogeneous Comprehensive Learning Particle Swarm Optimization (HCLPSO): This algorithm was firstly presented in 2015 by N. Lynn and P. N. Suganthan [158]. The swarm (population) is divided into two subpopulations. One is enhanced for exploration and the second one is enhanced for exploitation. For both subpopulations, a comprehensive learning (CL) strategy [151] is used to generate example particle $pBest_{fi(j)}$. For detailed information about selecting example particle, see the full paper [158].

3.2 SOMA

SOMA [271], [270] works with groups of individuals (population) whose behavior can be described as a competitive – cooperative strategy. The construction of a new population of individuals is based on the behavior of social group, e.g. a herd of animals looking for food. This algorithm can be classified as an algorithm of a social environment. In the case of SOMA, there is no velocity vector as in PSO, only the position of individuals in the search space is changed during one iteration, here called 'migration loop'.

The rules for the generic AlltoOne strategy are as follows: In every iteration, the best individual is chosen, and it is called the Leader. An active individual from the population moves in the direction towards the Leader in the search space. The movement consists of jumps determined by the Step parameter until the individual reaches the final position given by the PathLength parameter. For each step, the cost function for the actual position is evaluated and the best value is saved. The $PRTVector_j$ is generated for each new t-th step. This vector determines which dimensions j will be changed in a particular step t. In other words, in which dimensions the solution will travel towards the Leader or not. The $PRTVector_j$ consists only of values 0 or 1. The $PRTVector_j$ represents the mutation in SOMA. It should be ensured that at least one dimension will be affected by the migration process (heading towards leader).

At the end of the migration process, the position of the individual with minimum cost value is chosen. If the cost value of the new position is better than the cost value of an individual from the old population, the new one appears in new population. Otherwise the old one remains there. The main principle and the *„mutation operation"* are depicted in Figure 1, the movement is defined by equation (3).

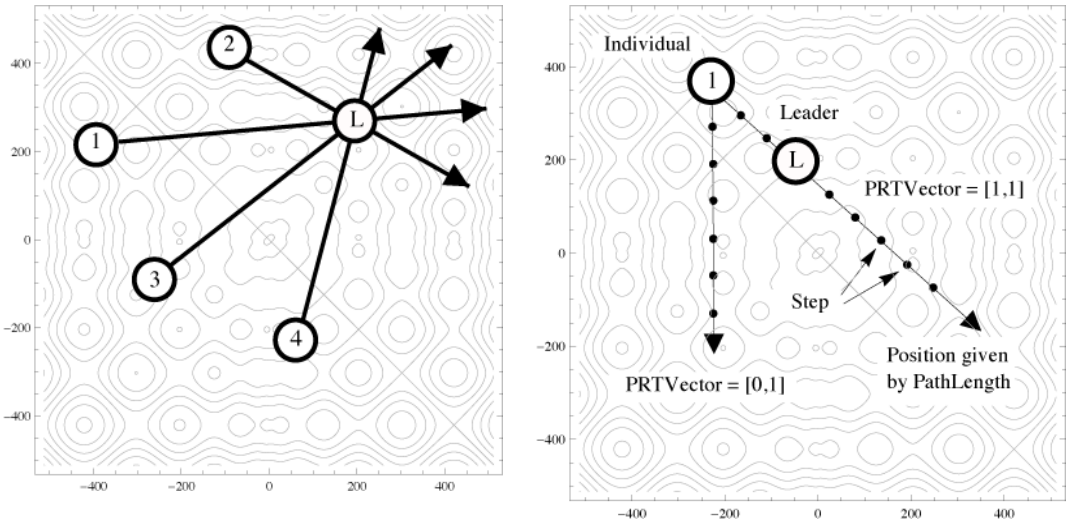

Figure 1: Basic principle of SOMA

$$x_{i,j}^{k+1} = x_{i,j}^{k} + \left(x_{L,j}^{k} - x_{i,j}^{k}\right) \cdot t \cdot PRTVector_j \tag{3}$$

Where $x_{i,j}^{k+1}$ is a new position of *i*-th solution (for iteration *k+1*) for dimension *j*. The current position of a solution *i* is $x_{i,j}^{k}$. The $x_{L,j}^{k}$ is a position of a so-called Leader, which is selected based on the chosen SOMA strategy. Parameter *t* represents steps from *i*-th solution to the Leader.

3.2.1 Basic strategies of SOMA

There exist four basic strategies of the SOMA algorithm – All To One, All To Random, All To All, All To All Adaptive. They differ in the behavior and relationships of so-called Leader and the rest of individuals (solutions) within the population and one migration loop.

All To One: All individuals from the population are migrating (moving) toward one selected Leader. This Leader is selected based on its objective function value, the individual with the best objective function value is selected in each iteration of an algorithm.

All To One Rand: The Leader is selected randomly at the beginning of each iteration from all individuals in the population.

All To All: All individuals migrate towards all others. After the end of migration of a particular individual, this individual returns to its original position. All individuals update their positions after all individuals completely perform their migrations.

All To All Adaptive: Again, all individuals migrate towards all others, but the updating process is being done immediately.

From the nature of the population behavior All To All and All To All Adaptive strategies can be much better in searching towards the global optimum, thus the possibility of finding the global extreme is then more probable. Nevertheless, these two strategies of SOMA require more computational time compared to the less time-demanding All To One, All To One Rand strategies.

Numerous applications either of canonical [76], [75], [77], [109] or special version [49], [50] of SOMA have proven that these heuristics are suitable for solving difficult classes of problems.

The SOMA algorithm is well described in the book [271] and it also have a special website[1].

3.2.2 Modern variants of SOMA

After almost 20 years from introduction of SOMA algorithm, the research has been restarted, and several successful strategies have been introduced in international conferences and benchmarking competitions.

[1] https://ivanzelinka.eu/somaalgorithm/

ESP-SOMA: The ensemble of Strategies and Perturbation Parameter in Self-organizing Migrating Algorithm was introduced to deal with the influence of user settings of parameters and selection of strategy before the algorithm run [124], [219]. In ESP-SOMA, user-predefined parameters are adaptive based on the actual performance of a running algorithm. Each individual from a population has assigned its ensemble of parameter values (*prt* value and strategy). Each individual randomly chooses its own SOMA strategy from a given set: {AllToOne, AllToAll, AllToOneRand}. Optionally, each individual randomly chooses its own *prt* value from given set {0.1, 0.3, 0.5, 0.7, 0.9} based on the user settings of adaptivePrt boolean parameter. The algorithm also introduces so called *Refreshing gap*, which defines the maximum number of unsuccessful migrations of the individual. If a solution reaches this maximum level, it is forced to adapt new *prt* value (optionally) and strategy.

SOMA TeamToTeamAdaptive (SOMA T3A): The organization process (structuring of population) plays an essential role in exploring promising subspaces and then focusing on exploiting these promising subspaces. Therefore, an individual selected to become the Leader is an individual with good fitness value, but should not always be the best. SOMA T3A [61], [63] introduces following organization process: random selection of a small group containing m individuals from the population, and then selection of the best n out of m individuals to become migrating individuals.

To select the Leader for each migrating individual, the algorithm randomly selects a small group containing k individuals from the population, and then select the best one from k, this individual is the Leader.

SOMA Pareto: In this strategy, to choose the migrating individual and Leader, the “Pareto" principle is applied [244], [62]. Overall 20% of the population with the best fitness value to form the group A, while the remaining 80% of the population will put into group B. The Leader should be in group A and the migrating individual should be in group B. Similar 20/80 ratio is applied again. The Leader is randomly selected as an individual within the best 20% of the group A (marked as group C), and the migrating individual is randomly selected within the best 20% of the group B (marked as group D).

3.3 Other swarm algorithms

Ant Colony Optimization (ACO): It is an algorithm whose activity mimics the actual behavior of ants in a colony when searching for and collecting food - thus belonging to the field of "swarm intelligence": The algorithm was first introduced by Marco Dorigo in his dissertation in 1992 [72]. The principle of activity could be classified as finding optimal paths through graphs. ACO simulates the movement of "ants across the landscape of an optimized problem and the marking of paths by pheromones". Thanks to pheromone marking, after some time all ants move along a shorter (optimal) path between the source and the target. This is due to the important function of "evaporation" of the pheromone. The shorter the path through the graph is, the less pheromone manages to "evaporate" until the ant returns, which marks the path again, making the path more attractive to other ants. Longer paths, where the amount of pheromone gradually decreases, will be abandoned over time.

As with other algorithms, there are a number of strategies: the Elitist Ant system, the Max-Min Ant system (MMAS), and the Rank-Based Ant System (ASrank)

ACO was originally designed for solving combinatorial problems (i.e. in discrete space). However, there are also versions for "global" continuous optimization. The representation of the pheromone, and the update is similar to the original version of the ACO, differs mainly in the construction of the new solution, which uses the solution archive and Gaussian functions [184]. More can be found in publications [74], [16].

Firefly Algorithm (FA): was firstly introduced in 2008 by X. S. Yang [260], [261], [262]. This nature-based algorithm tries to simulate the mating behavior of fireflies at night. Every firefly emits flashing light to lure appropriate mating partner. For the formulation of the FA, the flashing light is associated with the objective function value that is optimized. For simplicity, the following rules are used. All fireflies are sexless (each firefly can attract, or be attracted by, any of the remaining ones). The attractiveness of fireflies is proportional to their brightness. Thus, the less bright firefly will move toward the brighter one. The firefly brightness is based on the objective function value, and the brightness decreases with the distance between fireflies. If there is no brighter firefly, the particular one will move randomly. A comprehensive review is in the publication [96].

Artificial Bee Colony (ABC): The artificial bee colony algorithm is based on the generic bee algorithm and further specifies the specialization of individual bee species [126], [125]. It classifies bees into the following three groups: workers, observers and explorers. The number of workers corresponds to the amount of food sources (good solutions). The workers then communicate the information to the waiting bees (observers) in the hive so that they can choose a good source of food to collect. If the worker's food source is rejected, the bee becomes an explorer and sets out to find another food source. Like ACOs, bee algorithms are very effective for solving discrete optimization problems. Both algorithms are successfully used in the optimization of combinatorial problems. As in the case of ACO, after modifications of the general algorithm, ABC can be used for both continuous and discrete problems, but it is applied rather in discrete cases. A comprehensive review can be found in [127].

Greywolf Optimizer: This algorithm mimics the mechanism of the hunt and the hierarchy of the wolf pack [171], [92]. The wolf pack usually has five to twelve individuals and has a very strict and dominant social hierarchy. At its apex are males and females, referred to as alpha. Alpha are responsible for all decisions regarding hunting, location, etc. However, alpha may not necessarily be the strongest individual in the pack. The organization and controlling of the pack is more important. The second position is occupied by betas, the successors of the alpha. They must always obey the alpha, but at the same time they also have the right to give orders to wolves in lower social positions. The lowest position in the pack is occupied by omega. However, this position is also very important, as a prevention of too aggressive behavior of the pack and internal fights between individual members.

The algorithm thus distinguishes a total of four different species of wolves: alpha, beta, delta and omega. The process of hunting a pack consists of finding prey, encircling it, and attacking it. The social hierarchy is imitated in the algorithm so that the best available solution is occupied by alpha, the second by beta, etc.

Artificial Immune Systems (AIS): These algorithms also belong to the field of biologically inspired computing and collective intelligence [36], [47]. The development of AIS can be seen as two target domains: solving optimization problems through concepts inspired by the

real immune system; and providing models and simulations to study theories of the immune system (which is not a priority domain of computer science). For optimization purposes, AIS was defined as: "Adaptive intelligent systems, inspired by theoretical immunology and the observed immune functions, principles and models used in solving engineering optimization problems."

The motivation for research and creation of AIS were mainly the following key characteristics of the real immune system:

- self-organization of a large number of immune cells (agents) - there is a collective decision-making and solution without a central control unit (which meets the definition of swarm algorithms),
- distributed activity of the immune system in the body - the problem is solved by various agents at the same time - the immune system is naturally parallel,
- learning, adaptation and memory - the system can detect previous anomalies (so it has memory), adapt to changes and quickly and effectively solve a recurring problem,
- Effective pattern detection, anomaly detection, and classification.

The current most used algorithms are:

- Positive Response Algorithm (APS),
- Negative Response Algorithm (ANS)
- Clonal Selection Algorithm (ACS),
- Artificial Immune Networks,
- Dendritic Algorithm (DCA).

Further development of the 2nd generation of algorithms is focused mainly on a deeper understanding of self-organization, multi-agent behavior, parallelization of the immune system, but with a focus not only on humans, which was a mistake of I. generation algorithms and sometimes too fast and simplified "mimicking" partial complex system without the necessary theories. Here you can find a parallel with the rapid development of swarm algorithms and an incredible number of existing variants - see the next chapter).

Current research is therefore focused on Fast Artificial Immune Systems [41].

More information, including the necessary biologically oriented introduction (simulation of B activity, T-lymphocytes, antigens, etc.), are adduced in [52].

Memetic algorithms: Another important group of bio-inspired algorithms consists of the so-called Memetic algorithms [174]. These are complex structures, most often a combination of simple agents and so-called "memes", whose mutual evolutionary / iterative interaction leads to finding the optimal solution to the problem. In this group of algorithms, "meme" is understood as the evolution of ideas, i.e. essentially as natural evolution without DNA (i.e. without exact information). In general, it is an evolutionary framework with an embedded local search function - i.e. rather a complex and well-thought-out structure of communication together with a combination of evolutionary algorithms and local search. It is used practically to solve complex multicriteria, discrete, and "constrained" problems.

3.4 Decadence of swarm algorithms

As mentioned in selected examples above, not all swarm algorithms are concerned with mimicking the behavior of a group of animals / insects in the foraging process. There are also those whose main evolutionary goal is to choose the right partner in the mating process. An example is Marriage in Honeybee Algorithm or Bird Mating Optimizer.

There are also a large number of algorithms describing a number of other physical, chemical and other natural (social) processes, examples are below.

Inspiration from physics: artificial physics optimization, atmosphere clouds model optimization, charged system search, electro-magnetism optimization, gravitational search algorithm, harmony search, intelligent water drops algorithm, simulated annealing, spiral optimization, and others.

Inspiration from human society: anarchic society optimization, brain storm optimization, imperialist competitive algorithm, league championship algorithm, social emotional optimization algorithm, society civilization algorithm, and others.

Inspiration from biology: squirrel algorithm, bat algorithm, fruit fly optimization, cuckoo search, cultural algorithms, glowworm swarm optimization, shuffled frog- leaping algorithm, and others.

However, this system of "development" of algorithms has been widely criticized [232] recently. At the same time, a repository of algorithms was created, which were created as a simple mimicry of a model in nature. Evolutionary Computation Bestiary[2].

Currently, there are growing voices in the scientific community (similar to classical EAs) that attention should be more focused on understanding the dynamics / communication in the population of swarm algorithms, simplifying the overall understandability of the algorithm, complexity, improving adaptability (but not increasing the computational complexity of algorithm mechanisms), dealing with population diversity, premature convergence and stagnation in local extremes, and others.

[2] https://github.com/fcampelo/EC-Bestiary

4 PARALLEL EA

One of the common features of evolutionary algorithms is the population of solutions (individuals). Operations within the population can be performed both with a whole population, but also only with parts of it. This is how the idea of parallelism arose in EA. The parallel calculation of the objective function (which is a bottleneck in EA applications) does not require a central approach to the entire population. It can be processed with the help of several separate computing units, especially the individual CPU cores. This method of parallelism is called *physical*, there is no change in the sense of performing sequences of operations in the evolutionary or swam algorithm, and indeed there is a reduction in real-time computation. Another method of parallelism in EA, referred to as *pseudo-parallelism*, essentially changes the original intended dynamics and evolution of the population over time, while allowing, along with computational time reduction, the ability to find better solutions than would not be possible with the original unmodified version of a particular algorithm.

The most well-known models of parallelism include a total of three, each of which exhibits specific properties [6], [33], [180]. These are the "master-slave", the "neighborhood model" and the "migration model". The latter is also sometimes referred to as the island model. In the literature it is also possible to find the classification of models into so-called *coarse-grained* and *fine-grained* for population distribution. The two above-mentioned master-slave models and the migration model fall into the category of coarse-grained parallelism. The last-mentioned neighborhood model falls into the category of fine-grained population distribution in EA.

4.1 Topology of Models

Master-slave: A detailed description can be found, for example, in [180]. The running of the evolutionary algorithm and the evaluation of individuals in the population in this model is managed by a central control node called “master”. The processes take place at a so-called low level, so central memory access is not required. Typically, the evolutionary algorithm runs on a central computational node, and at the moment when it is necessary to evaluate the newly created offspring (calculation of the objective function), parts of the population are assigned to partial computational nodes - called slaves. The advantage here is a relatively

easy implementation on HW, the disadvantage is the need for communication between nodes and a relatively small area of applicability of parallelism (from the overall algorithmic scheme) - basically just the calculation of the objective function. The application and use of the model are described in [53], [64], [197], [263].

Neighborhood model: Compared to the previous one, this model is completely decentralized. According to the principle of how individuals from such a distributed population communicate with each other, this model is called as a diffuse or also as a cellular model due to the geometric distribution of individuals [33], [180], [235]. In the master-slave model, the parallel distribution of the calculation is performed on the condition that the participation of either a larger number or all individuals from the population will not be necessary. In the neighborhood model, on the other hand, it is assumed that the algorithm will work with geometrically defined subpopulations that intersect with each other, and within these subpopulations, operations such as selection, crossover and mutation are performed. The most commonly used layout of individuals is a 2D or 3D grid. This grid then clearly identifies the numbers of neighboring computing nodes that participate in the evolutionary operations. This divides the population into several separate decentralized compute nodes, thus reducing the total time required to run the evolutionary algorithm. More detailed information is given, for example, in the following publications [172], [218], [211].

Migration ("Island") model: represents a distributed approach, which can be described as a compromise between the above-mentioned centralized (master-slave) and decentralized (neighborhood) models. The computational nodes of this model are called islands. The most important element here is the operation of migration of individuals between islands [180]. The way of connecting distributed parts - topology - also plays a big role here. A possible disadvantage of the implementation is that the migration model contains a number of parameters. Their correct setting significantly affects the quality of the results achieved in solving the optimization problem. These parameters are: number of islands (subpopulations), size of islands, topology, migration rate, immigration policy, emigration and frequency of migration. Experiments with parameters setting and application are summarized in publications [34], [35], [157], [144].

Hybrid models: Of course, the three above-mentioned models can be combined in various ways and the interconnection of partial advantages of individual models can be used within the so-called hybrid models [5].

Applications and research of various models of parallelism, especially for DE and GA, can be found for example in [255], [176].

4.2 HW (physical) parallelism

Physical parallelism for EA already represents the technical side of the real application of individual computational nodes for different models and topologies to a specific HW and language. There are a number of implementations, e.g. in the CUDA environment [248], [153], [90], or other HPC servers [239].

5 DISCRETE EA

EAs are generally an effective method for solving discrete optimization problems. Due to the nature of operations, a number of algorithms are designed to work with discrete attributes of an individual (ACO, ABC,…).

Thanks to possible variants of coding an individual, genetic algorithms are suitable for operations with real numbers and discrete values.

Other algorithms (DE, SOMA, PSO) [51], [37], [195], which were primarily designed for a continuous space, need to be modified. The solution is the operation of rounding the individual (there are several approaches) and subsequent processes of repairing the individual (so-called repairment), so that the individual represents a valid solution (e.g. in a TSP problem it is not possible to have 2 identical cities). Special techniques for mapping of solutions to combinatorial/permutation problems are required too. Examples are in [254], [265], [196], [154], [142].

6 EVOLUTION OF SYMBOLIC STRUCTURES

Generally, in classical statistics, regression is a method of data fitting, which represents the selection of possible type of function (curve) and finding proper numerical constants by means of regression analysis to fit a data series. Considering the same type of practical problem, then symbolic regression (also called symbolic function identification) is a function discovery approach for analysis of data sets for a purpose of whole synthesis of data-generating systems (models). Symbolic regression represents the process of synthesis of a final formula from basic simple functions, operators, terminals (variables and constants), that fit the given data set.

The development of symbolic regression comes from simple challenge:

"*How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it*?" Arthur Samuel (1959).

The idea of automatic program creation attracted researchers together with the first steps in development of evolutionary (genetic) algorithms. The symbolic regression approach was firstly introduced by John Koza as well known genetic programming (GP) [139], [137], [140], [141], [179], later followed by Conor Ryan with grammatical evolution (GE) [215], [182], [181]. Another technique so-called analytic programming, was developed by Ivan Zelinka [273], [272], in 2001.

6.1 Genetic programming

In genetic programming, the new population is not evolved by a "classical" numerical approach, but at a symbolic level. This means that the population does not contain numeric attributes, but symbolic objects themselves (mathematical functions, operators, parts of programs, etc.). GP represents an extension of genetic algorithms, i.e. also nomenclature and the central evolutionary dogma is the same here. In the original version of GP, the individual's body consists of a function in the LISP language. For clear visualization and

transformation of an individual into the space of complex structures, GP uses a tree structure (i.e. nonlinear notation of the individual).

The evolutionary process contains the same sequence of operations as GA. After random population formation, selection and crossover are performed. A place (tree node) is chosen where the parents representing the individual complex structures (trees) are divided, and these parts are then swapped between them, creating new offspring. Crossover is followed by a mutation operation. Here, as with GA, there is no random bit inversion in the genes, but a certain part of the individual is replaced by a randomly generated string - a functional structure.

Overall, there are several approaches to genetic programming that differ mainly in the principles of coding an individual compared to the generic version, which promoted the LISP format. Individuals are encoded in a linear form, but again express themselves by more complex nonlinear structures (for example, the tree structures mentioned above). Other differences may be whether usually only one solution is encoded in one individual, or whether information about multiple solutions is stored in one chromosome at the same time. An overview of surveys can be found in [91], [138], [249], [203], [2], [134].

6.2 Grammatical Evolution

Grammatical evolution can be understood as a type of grammar-based genetic programming. Using grammatical evolution, programs can be synthesized in any programming language using the Backus-Naur form (BNF) to represent grammar.

Complex structures (functions, programs) created by GE are not written directly in the tree structure as in GP, but by the linear genome, most often by sequences of integers. The transformation of the genome into functions using grammar is then performed using modulo n operations, where n is determined by the number of possibilities for a certain grammar rewriting rule.

GE represents a combination of genetic algorithms and appropriate grammar, therefore, the process of evolutionary breeding of new solutions is performed by parent selection, crossover and mutations. However, crossover takes place at a different level than in GP (not

in tree nodes), in addition, the evolutionary algorithm used must cope with the variable length of the individual. Related research is published in [163], [58], [178], [116].

6.3 Analytic programming

Analytical Programming (AP) is an original approach to symbolic structure synthesis which uses EAs. Since it can utilize arbitrary classical evolutionary or swarm-based algorithm and it can be easily applied in any programming language, it can be considered as a powerful open framework for symbolic regression. Since its introduction in 2001; it has been proven on numerous problems to be as suitable for symbolic structure synthesis as GP.

Various applications of AP can be found in papers [194], [220], [274], [221], [250], [247]. The comprehensive description of the whole methodology is given in book chapter [272]. Next sections below are focused on the description of the basic functionality of AP.

The AP is formed by three parts – General Functional Set (GFS), Discrete Set Handling (DSH) and Security Procedures (SPs). GFS contains all elementary objects (functions, operators, terminals), which can be used for complex formula synthesis. DSH represents the procedure of mapping of individuals to programs and SPs are implemented into mapping process to avoid mapping to pathological programs and to prevent critical situations.

***General Function Set*:** AP uses sets of functions and terminals used for mapping from individual domain to program domain. The synthesized structure (program) is branched by functions requiring two and more arguments, and the length is then extended by functions which require only one argument. Terminals do not contribute to the complexity of the synthesized program (length) but are needed to synthesize a non-pathological structure (a program that can be evaluated by user defined cost function). Therefore, each non-pathological program must contain at least one terminal (variable).

The content of GFS is dependent on user choice. GFS is nested and can be divided into subsets according to the number of arguments that the subset requires. GFS_{0arg} is a subset which requires zero arguments (only terminals). GFS_{1arg} contains all terminals and functions requiring one argument, GFS_{2arg} contains all objects from GFS_{1arg} and functions requiring

two arguments, and so on, GFS_{all} is a complete set of all elementary objects. For mapping procedure, objects in GFS are ordered by some arguments they require in descending order.

***Discrete Set Handling*:** DSH is the core operation used for mapping the individual to the synthesized structure. Since the most of EAs use individuals with real number encoded attributes, the first important step in DSH workflow is to get individual with integer components which are obtained by simple rounding of real number values. The integer values of an individual represent indexes into the discrete set, in this case, GFS, and its subsets. If the index value is greater than the size of used GFS, modulo operation with the size of the discrete set is performed. An illustrative example of mapping is given in (4), where the objects in GFS_{all} are indexed from 0.

$$\begin{aligned} Individual &= \begin{Bmatrix} 0.12, 4.29, 6.92, 6.12, 2.45, \\ 6.33, 5.78, 0.22, 1.94, 7.32 \end{Bmatrix} \\ Rounded\ individual &= \{0, 4, 7, 6, 2, 6, 6, 0, 2, 7\} \\ GFS_{all} &= \{+, -, *, /, sin, cos, x, k\} \\ \boldsymbol{Program}&\boldsymbol{:}\ \boldsymbol{\sin x + k} \end{aligned} \tag{4}$$

The first rounded individual feature is 0, which represents a + operator in GFS_{all}. It requires two arguments, and those are represented by next two indexes – 4 and 7, which are mapped to function sin() and constant k. The sin() function requires one argument which is given by the next index (rounded feature) – 6, and it is mapped to variable x. Since there is no possible way of branching the program further, other features are ignored and synthesized program is: sin x + k.

***Security Procedures*:** As mentioned above, SPs are used in AP to avoid critical situations. Some of the SPs are implemented into the AP core operations, and some have to be applied into the cost function evaluation. The typical representatives of the later are checking synthesized programs for loops, infinity, and imaginary numbers if not expected (dividing by 0, square root of negative numbers, etc.). SPs implemented in core AP operations are used to check for pathological programs. Pathological programs are those which cannot be evaluated

due to the absence of arguments in the synthesized function. For example, individual with rounded features of {4, 4, 4, 4, 4} would be mapped to program sin(sin(sin(sin(sin_)))) which lacks constant or variable at the empty position denoted by _ and thus represent a pathological program. Such situation can be avoided by a simple procedure which checks remaining positions (rounded parameters) of the individual during mapping and according to this value maps rounded individual features to subsets of GFS_{all} which do not require too many arguments.

***Constants Handling*:** The simpler approach towards constant handling is the generation of constants and adding them as terminals into GFS (for AP) or source set of functions for GP/GE. This approach does not increase the computational time requirements; however, it increases the complexity of the search space. Thus, the constant values in synthesized complex structures are usually estimated by the second EA (meta-evolution/hyper-heuristics) or by nonlinear fitting techniques (or equivalents according to the programming environment and available libraries/functions). The big disadvantage is the high computational cost, since within single evaluation of quality of synthesized formula, we have to spend whole budget of FES for secondary EA used for finding of constants. Another approach represents the utilization of extended (not-spent) part of the individual in EA for the evolution of constant values. The important task is to determine, what is the correct size of an extension. While mapping the individual into a program, the constants are indexed and later replaced by the value directly from the not-rounded individual. A simple example can be seen in (5) using the very same GFS set as the example above (4). Individual features in bold are the constant values, and features which are going to be mapped to GFS are rounded and the rest is omitted.

$$\begin{aligned} Individual &= \left\{ \begin{matrix} 5.08, 1.64, 6.72, 1.09, 6.20, \\ 1.28, \mathbf{0.07}, \mathbf{3.99}, \mathbf{5.27}, \mathbf{2.64} \end{matrix} \right\} \\ Rounded\ individual &= \{5, 2, 7, 1, 6, 1\} \\ GFS_{all} &= \{+, -, *, /, sin, cos, x, k\} \\ \boldsymbol{Program}&: \mathbf{cos}(\boldsymbol{k1} * (\boldsymbol{x} - \boldsymbol{k2})) \\ \boldsymbol{Replaced}&: \mathbf{cos}(\mathbf{0.07} * (\boldsymbol{x} - \mathbf{3.99})) \end{aligned} \tag{5}$$

The first index to GFS_{all} is 5, which represents *cos* function, its argument is chosen by next index – 2 representing function * which needs two arguments. Arguments are indexed 7 and 1 – constant *k1* and function -. After this step, two arguments are required and only two features are left in the program part of the individual. Therefore, the security procedure takes place, and those last two features are indexed into GFS_{0arg}. Thus indexes 6 and 1 are mapped to variable *x* (6 mod size(GFS_{0arg}) = 0) and constant *k2*. The synthesized program is therefore *cos(k1 * (x – k2))*. The constants are replaced by the remaining features 0.07 and 3.99 respectively. Types of constant estimation are in [252], [193], [246].

6.4 Other methods

The introduction of GP and its applicability caused the intensive development of related techniques. Other approaches have been based on an artificial immunological system [123], on the generation of programs from the adaptive probability distribution of all possible solutions in Probabilistic Incremental Program Evolution (PIPE) [217]. Grammatical evolution was preceded by GADS (Genetic Algorithm for Deriving Software) as an improved genetic programming using an alternative grammatical approach to programming [198]. Furthermore, Gene Expression Programming techniques [94], [93] and Multiexpression Programming have been developed [190].

An important area of evolutionary computing techniques is also the area of so-called evolutionary hardware. An important method for evolutionary HW is especially Cartesian genetic programming [169], [101], [251].

7 HYPER-HEURISTICS

Hyper heuristic techniques [32], [30], [31], [78] or sometimes called meta-evolution processes represents complex evolutionary approaches that connect different evolutionary optimization techniques at strictly defined hierarchic level. Generally speaking, “meta” means going beyond the basic term and meta-evolution means evolution of evolution. Hyper heuristic or meta-evolution represents several commonly used approaches.

The first possibility represents the usage of an evolutionary optimization technique for tuning or controlling of another evolutionary technique [230], [84], [60]. During this process, usually the best types of evolutionary operators, strategies and numerical settings of their parameters are evolutionarily selected (estimated).

Another example represents the process of the estimation of coefficients within symbolic regression when two evolutionary algorithms are used. The master evolutionary algorithm drives the main process of symbolic regression, and the secondary slave algorithm is used for the coefficients estimation [192].

Finally, it can represent an evolution for synthesis of a complex logical structures and parameters from the set of evolutionary operators such as selection crossover or mutation, i.e. evolutionary synthesis of a new evolutionary technique [65], [136], [243]. This approach is generally very close to the idea of artificial intelligence evolving better artificial (general) intelligence.

The research related to the hyper heuristics is on spotlight now, thanks to the complex idea mentioned in the previous example. Also, there is a research growing in development of benchmarking methods and methods for quality evaluation.

8 MULTIOBJECTIVE EVOLUTIONARY ALGORITHMS

Most difficult optimization problems in nature have several (possibly conflicting) objectives to be satisfied (e.g., design a bridge for which want to minimize its weight and cost while maximizing its safety). For easier applications of well-known EA, many of these problems are frequently treated as single-objective optimization problems by transforming all but one objective into constraints. Nevertheless, such an approach is many times difficult to implement, especially with growing numbers of objectives.

8.1 Definition of MOP and Pareto Optimality

In case, there exists several objective functions, the notion of "optimum" changes, because in multiobjective optimization problems (MOP), the aim is to find good compromises (or "trade-offs") rather than a single solution.

This notion was generalized by the italian economist Vilfredo Pareto, in 1896 in his book *Cours d'Economie Politique*, as commonly accepted term: Pareto optimum.

The definition of Pareto optimum says that solution is Pareto optimal if there exists no feasible vector of attributes which would decrease some criterion without causing a simultaneous increase in at least one other criterion. This concept normally produces a set of solutions called the Pareto optimal set. The solutions (vectors) included in the Pareto optimal set are called nondominated. The image of the Pareto optimal set is called the Pareto front. The Pareto dominance is defined as: a solution to dominate another one, it should not be worse in any objective and must be strictly better in at least one of them. More details can be found in [39].

In general, it is impossible to find an analytical expression that represents the line or hypersurface corresponding to the Pareto optimal front. The very demanding procedure for generating (a close approximation of) the Pareto optimal front of a problem is to compute all the (or as many as possible) feasible points and to obtain their corresponding objective function values. When we had obtained a sufficient number of such points, it is possible to determine which are the nondominated solutions from them (the Pareto optimal set). Although there exists many mathematical programming techniques for nonlinear multi-objective optimization, they have several limitations. For example, objectives (and the constraints) must be

differentiable, Pareto fronts must not be disconnected or non-convex, etc. Furthermore, most of them generate only a single solution per execution. Those are the reasons, why EAs have been improved and modified for MOPs.

8.2 Classical MOEA

The potential of evolutionary algorithms for solving multi-objective optimization problems has been explored in the PhD thesis of Richard Rosenberg from 1967 [212]. Later, John David Schaffer developed the first actual implementation of a multi-objective evolutionary algorithm (MOEA), which is provided in his PhD thesis [224]. His approach was called Vector Evaluated Genetic Algorithm (VEGA) [225]. This algorithm was later criticized by David Goldberg in his famous book on genetic algorithms [103]. In this book, Goldberg proposed to identify the nondominated solutions in the population. This idea was improved by Carlos M. Fonseca in the so-called Multi-Objective Genetic Algorithm (MOGA), which was proposed in 1993 [99]. Some comparative studies from the 1990s showed that MOGA was the best MOEA from the first generation of MOEA algorithms using non-elitist Pareto-based methods.

The importance of elitism for ensuring the convergence of MOEA to the true Pareto optimal set was later studied in [214]. At the same time, Eckart Zitzler proposed the Strength Pareto Evolutionary Algorithm (SPEA) in his doctoral thesis. Its algorithm integrates various mechanisms from the above mentioned approaches (including external archives and generalized elitism) [282]. In 2002, Kalyanmoy Deb designed Nondominated Sorting Genetic Algorithm II (NSGA-II) [57]. The effectiveness of NSGA-II along with publicly available source code has made it the standard in evolutionary multi-objective optimization for more than the following 10 years (and probably the most popular MOEA ever). However, NSGA-II does not work properly with more than 3 objectives and with binary coded individuals (real numbers have proven to be a better choice).

Taxonomy of the first-generation algorithms and other examples are in [38], [55].

8.3 Modern MOEA algorithms

The first step towards modern and effective MOEAs is represented by IBEA (Indicator-Based Evolutionary Algorithm), that is an algorithmic framework that allows the usage of any kind of performance indicator in the selection mechanism for individuals [281]. Later, Emmerich et al. proposed SMS-EMOA (S Metric Selection Evolutionary Multiobjective Algorithm) based on NSGA-II and operations with archive [88], [11]. This algorithm uses steady state selection and the crossover and mutation operators from NSGA-II. Then, it applies Pareto ranking. When the last nondominated front has more than one solution, SMS-EMOA uses hypervolume to decide which solution should be removed.

Further research was focused mainly on the alternative indicators instead of hypervolume, that is very computationally demanding to evaluate [226], [106].

An extension of NSGA-II, that is specifically designed to deal with many-objective problems (i.e., multi-objective optimization problems having 4 or more objectives) was introduced in 2014 as NSGA-III [56].

Nevertheless, still there exist many challenges (like scalability, visualization) and some of them are summarized in [149], [245], [150], [1].

References, public domain implementations of algorithms and many more can be found in repository for MOEA (EMOO repository)[3].

[3] https://emoo.cs.cinvestav.mx/

9 SURROGATE MODELING

The effective parallelization (described in dedicated chapter) can also be supported by surrogate modeling [22]. Surrogates are representing so-called substitute/metamodel techniques, where the objective function is not evaluated directly, but instead, an approximation is used as feedback to the optimization algorithm. Surrogates are used mostly in industrial optimization and so-called expensive scenarios. Examples can be found in [121], [145], [122], [24].

9.1 Surrogate and EA

The basic feature of EA is that they typically require thousands of cost function evaluation and they explore the whole search space. During the exploring, EA find solutions that could be unexpected from engineering point of view. Hence, there are many scenarios when it is necessary to use surrogates:

- evaluating a single solution can be computationally very expensive,
- evaluating a solution can be costly (using of expensive licenses for simulators),
- evaluating a solution can be dangerous (chemical reactors, etc),
- evaluating a solution may require user interaction,
- number of cost function evaluations is strictly limited.

Surrogates are using various approaches:

- Gaussian processes,
- k-Nearest Neighbour,
- Decision Trees and Random Forests
- Artificial Neural Networks as universal function approximators
- Diverse ensembles,

Surrogate models depend on distance function. In higher dimensions, distance function becomes less useful. Last but not least, it is necessary to define the model of population management for handling the real objective function calculations and approximators interleaving throughout individuals in population and iterations [23].

9.2 Surrogate and symbolic regression

Usage of surrogates and symbolic regression methods are currently being investigated. Situation here is a bit more complicated. Possible solution is application of phenotypic distance with particular focus to address the problem of higher dimensionality, where distance functions for surrogates/approximations become less useful [117].

10 BENCHMARKING OF EA

When developing new algorithms or modifications of learning and adaptation mechanisms in existing algorithms, it is always necessary to numerically verify together with statistical validation whether there has been a significant change in the efficiency and performance of the new or modified algorithm. The correct way to perform statistical tests (not only) for EA is, for example, in [59].

Two different methods are used to test evolutionary algorithms. In the first case, we usually use already existing (real optimization) problems, which have already been solved by other algorithms. We then compare the results of the currently tested algorithm with the existing results.

The second method is to use a set of test functions (so-called benchmark set), containing functions with different properties, such as nonlinearity, different pathologies like the plane around the extreme, or hybrid / composite functions, where each dimension can be calculated by a different function, etc. The advantage of official benchmark sets is that the analytical formulas are known, and therefore it is very easy to calculate the position and value of the extreme for any dimension for most of those functions.

In the case of using benchmark sets, there are generally 2 different procedures described below (budget / target). In any case, it is always considered that it is a test of heuristics, so each test case must be repeated at least 30-51 times.

10.1 Budget type benchmarking

With this type of testing, there is a maximum limit on the number of evaluations of the objective function (often set to 10,000 x D). The statistical properties of the results of repeated execution are evaluated (min, max, median, mean, standard deviation,…) both for the final population and in the selected time stamps (again given the % of total number of the

objective function evaluations). Examples are mainly IEEE CEC benchmark sets[4][5], which contain a number of scenarios and limitations in individual years of release:

- IEEE CEC 2011 competition (real-life problems)[6]
- IEEE CEC 2015 competition (15 functions)
- IEEE CEC 2017 competition (30 functions)[7]
- IEEE CEC 2020 competition (10 functions)[8]

10.2 Target type benchmarking

It consists in evaluating how many times and how quickly the evolutionary algorithm found a sub-target, or with what numerical accuracy. Often, the time limitation or the number of evaluations of the purpose function does not play a role here, and algorithms with the so-called restarting are preferred - i.e. re-initialization of the population after finding a partial target. Examples are:

- BBOB test bed[9] [112]
- IEEE CEC 2019 100 digits competition[10]

10.3 Benchmarking of discrete algorithms

In the years 2018 - 2019, a new benchmark was introduced containing a carefully selected set of discrete optimization problems – BB-DOB. Detailed information can be found in [269], [70].

[4] https://www.ntu.edu.sg/home/epnsugan/index_files/cec-benchmarking.htm
[5] https://github.com/P-N-Suganthan?tab=repositories
[6] https://github.com/P-N-Suganthan/CEC-2011--Real_World_Problems
[7] https://github.com/P-N-Suganthan/CEC2017-BoundContrained
[8] https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark
[9] https://coco.gforge.inria.fr/
[10] https://github.com/P-N-Suganthan/CEC2019

10.4 Benchmarking platforms and other sets

Along with the emphasis on increasing the expertise and quality of EA development, there is also a growing emphasis on quality and proven benchmark sets and platforms. Recently, a very well-developed platform for testing iterative optimization heuristics, the so-called IOH profiler[11] [69], was created. A working group of leading experts (Benchmarking Network[12]) was also established, and within the IEEE CIS subsection - Computational Intelligence Society, a Benchmarking task force was recently established. It is also worth mentioning the increasingly popular testing of EA using games (Mario Bros and others)[13].

[11] http://iohprofiler.liacs.nl/
[12] https://sites.google.com/view/benchmarking-network/home
[13] http://www.gm.fh-koeln.de/~naujoks/gbea/

11 APPLICATIONS OF EA

In the previous chapters, when describing a specific method or algorithm thanks to references to the literature, a relatively wide range of possible EA applications was always mentioned.

In general, EA can be used for almost any optimization problem, as long as it is possible to clearly define the objective function evaluating the quality of individual solutions, from which the evolutionary process breeds the best available solution while meeting time limitations or budget of objective function evaluations.

The analysis of literature sources further shows that the PSO algorithm has the largest representation in applications, followed by DE and Genetic algorithms.

The most common application areas are the following:

- Electrotechnics,
- Automation and control systems,
- Energy,
- Telecommunications,
- Operational research,
- Industrial design.

Other review studies are listed, for example, in [48], [15], [97], [54], [21], [159], [20], [229], [231], [80], [259], [135], [208], [162], [114].

12 FUTURE DIRECTIONS OF ECT

EA gained high popularity especially in the period of 1990 - 2010, thanks to the development and modifications of numerous very successful algorithms, which gradually supplemented the portfolio of available unconventional optimization techniques and partially replaced the previously preferred genetic algorithms. After 2010, the attention of experts working in fields close to the artificial intelligence turned to the reincarnation of machine learning, i.e. deep neural networks and deep learning in general. At the same time, the development of new algorithms slowed down, as criticism began to emerge that it was too simple to imitate almost any natural process containing a random element and to compare these new algorithms only with the oldest versions (either classic or swarm). Increased emphasis on population analyses, communication dynamics, diversity, and stagnation of population has logically reduced interest in this previously popular area of research.

However, in recent years, EA research has come to life again. Firstly, thanks to interesting new concepts of algorithms and adaptation mechanisms, new theoretical studies [100], the constant increase in the dimension of solved problems, non-traditional space search mechanisms such as "Novelty search" by Kenneth O. Stanley [146], and especially due to the connection with structural design and tuning parameters of artificial (deep) neural networks: “Neuro-evolution”, where EAs are considered as a powerful creative tool [167].

EAs have been used to optimize neural network structures or parameters in a field called Neuro-evolution for many years [264]. Modern and robust EAs strategies, due to their universality, can be used to solve emerging problems related to machine learning, like the finding of optimal hyper-parameters for the model, select the optimal previous models based on experiences and gained results, optimal features selection, or optimizing the convolution network structure used as a feature extractor for further processing/classifications. All tasks as mentioned above belong to the paradigm of automated machine learning (AutoML) [95] and the subfield of (deep) neural architecture search, where evolutionary algorithms have been successfully explored [86], [209], [152].

Currently, there exists different successful workflows and implementations: simple yet efficient evolutionary strategies [216], genetic algorithms for deep architectures [236], [168], dedicated genetic programming frameworks for evolving machine learning pipelines [189].

Last but not least, also Google DeepMind is investigating the utilization of evolutionary algorithms in AutoML[14][15].

In conclusion, it can be said that there are a number of as yet undiscovered possibilities for the application of evolutionary optimization techniques and especially for the research of their theories.

[14] https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html
[15] https://deepmind.com/blog/article/how-evolutionary-selection-can-train-more-capable-self-driving-cars

CONCLUSION

This text, which is primarily intended for the course Bioinspired Optimization Methods in the doctoral study program at the Faculty of Applied Informatics, Tomas Bata University in Zlín, is designed as an offer of topics from the state of the art in the mentioned research field that can be used by doctoral students during their studies and in their dissertation. For a detailed study of the issue, it is possible to use the cited literature which also serves as recommended for the completeness of the further knowledge.

The text deals with an overview of the latest modern algorithms solving continuous and discrete optimization, optimization of parameters of high-dimensional (large-scale) problems or optimization of symbolic structures, modification or hybridization for multicriteria and multi-objective methods or assisted surrogate optimization methods. The textbook begins with the basic concepts, history and predecessors of bioinspired optimization algorithms. In the following chapters, the student gets acquainted with the principles of individual algorithms, internal population dynamics, as well as optimization, adaptation and learning techniques for the control parameters of these algorithms.

In chapter 2, the author deals with the well-known method of genetic algorithms and the currently leading algorithm in competitions at large scientific congresses, differential evolution and its variants and strategies. In the next third chapter, the topic covers swarm intelligence in general and derived swarm algorithms, including PSO, SOMA and other developed techniques. In the period of time and space consuming simulations, it is necessary to address the issue of algorithm acceleration using parallelization, which is discussed in Chapter 4.

The topics presented in the individual sections also deal with the possibilities of hybridization and modification of the above-mentioned original techniques "tailored" to different classes of problems. At present, in the areas of human action, industry or the medical industry, very demanding tasks with more or many criteria (multi-objective) (see Chapter 8) and also in the space of high dimensions (large-scale problems) of individual optimized parameters are processed. The algorithms have to be edited more or less they to be able to deal with the problems in a correct way. The modification of algorithms is also necessary to keep in mind within currently solved application problems from the areas of not only continuous but also discrete space (in detail in Chapter 5) with permutation, combinatorial or similar derived combinations. Researchers also deal with the acceleration of computations, which include,

for example, a very sophisticated approach of assisted optimization for computational demanding real optimization models (surrogate assisted models), which the reader learns about in Chapter 9. The sixth chapter deals with the superstructure of evolutionary and swarm algorithms, with the evolution of symbolic structures. In this chapter, the student gets acquainted in more detail with genetic programming, grammatical evolution and analytical programming.

In the tenth chapter, the author introduced the field of testing optimization techniques, which is an important part of the development of new or modifying or hybridizing of known bioinspired soft computing optimization algorithms. The chapter describes benchmarking, i.e. suitable approaches and preparation of test tasks from various points of view ensuring high quality and robustness of the most modern algorithms winning the competitions.

In the closing chapters 11 and 12, the student is acquainted with a number of interesting solved applications with a view to future directions of research and use of bioinspired computational methods in various areas of human activities.

To deal with individual areas in full and detailed way exceeds the scope and focus of the text. However, I believe that the offered text will help you to orient yourself in the field of currently used optimization techniques, their variants, strategies and settings. possible uses and will inspire you to choose the appropriate technique applicable to your dissertation.

BIBLIOGRAPHY

[1] Abouhawwash, M., Seada, H., & Deb, K. (2017). Towards faster convergence of evolutionary multi-criterion optimization algorithms using Karush Kuhn Tucker optimality based local search. *Computers & Operations Research*, *79*, 331-346.

[2] Agapitos, A., Loughran, R., Nicolau, M., Lucas, S., O'Neill, M., & Brabazon, A. (2019). A Survey of Statistical Machine Learning Elements in Genetic Programming. *IEEE Transactions on Evolutionary Computation*, *23*(6), 1029-1048.

[3] Al-Bahrani, L. T., & Patra, J. C. (2018). A novel orthogonal PSO algorithm based on orthogonal diagonalization. *Swarm and Evolutionary Computation*, *40*, 1-23.

[4] Al-Dabbagh, R. D., Neri, F., Idris, N., & Baba, M. S. (2018). Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy. *Swarm and Evolutionary Computation*, *43*, 284-311.

[5] Alba, E., & Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE transactions on evolutionary computation*, *6*(5), 443-462.

[6] Alba, E., & Troya, J. M. (1999). A survey of parallel distributed genetic algorithms. *Complexity*, *4*(4), 31-52.

[7] Awad, N. H., Ali, M. Z., Suganthan, P. N., & Reynolds, R. G. (2016, July). An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems. In *2016 IEEE congress on evolutionary computation (CEC)* (pp. 2958-2965). IEEE.

[8] Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

[9] Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of evolutionary computation*. CRC Press.

[10] Back, T., Hoffmeister, F., & Schwefel, H. P. (1991, July). A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms* (Vol. 2, No. 9). Morgan Kaufmann Publishers San Mateo, CA.

[11] Beume, N., Naujoks, B., & Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, *181*(3), 1653-1669.

[12] Beyer, H. G. (1995). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, *3*(3), 311-347.

[13] Beyer, H. G. (2001). *The theory of evolution strategies*. Springer Science & Business Media.

[14] Beyer, H. G., & Schwefel, H. P. (2002). Evolution strategies–A comprehensive introduction. *Natural computing*, *1*(1), 3-52.

[15] Biethahn, J., & Nissen, V. (Eds.). (2012). *Evolutionary algorithms in management applications*. Springer Science & Business Media.

[16] Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, *2*(4), 353-373.

[17] Blum, C., & Li, X. (2008). Swarm intelligence in optimization. In Swarm intelligence (pp. 43-85). Springer, Berlin, Heidelberg.

[18] Blum, C., & Merkle, D. (2008). Swarm intelligence. *Swarm Intelligence in Optimization; Blum, C., Merkle, D., Eds*, 43-85.

[19] Bonabeau, E., Dorigo, M., Marco, D. D. R. D. F., Theraulaz, G., & Théraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems* (No. 1). Oxford university press.

[20] Bonyadi, M. R., Michalewicz, Z., Wagner, M., & Neumann, F. (2019). Evolutionary computation for multicomponent problems: opportunities and future directions. In *Optimization in Industry* (pp. 13-30). Springer, Cham.

[21] Bozorg-Haddad, O., Solgi, M., & Loáiciga, H. A. (2017). *Meta-heuristic and evolutionary algorithms for engineering optimization*. John Wiley & Sons.

[22] Branke, J., & Schmidt, C. (2005). Faster convergence by means of fitness estimation. *Soft Computing*, *9*(1), 13-20.

[23] Branke, J., Asafuddoula, M., Bhattacharjee, K. S., & Ray, T. (2016). Efficient use of partially converged simulations in evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, *21*(1), 52-64.

[24] Branke, J., Schmidt, C., & Schmeck, H. (2001, July). Efficient fitness estimation in noisy environments. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation* (pp. 243-250).

[25] Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, *10*(6), 646-657.

[26] Brest, J., Korošec, P., Šilc, J., Zamuda, A., Bošković, B., & Maučec, M. S. (2013). Differential evolution and differential ant-stigmergy on dynamic optimisation problems. *International Journal of Systems Science*, *44*(4), 663-679.

[27] Brest, J., Maučec, M. S., & Bošković, B. (2017, June). Single objective real-parameter optimization: algorithm jSO. In *2017 IEEE congress on evolutionary computation (CEC)* (pp. 1311-1318). IEEE.

[28] Brest, J., Maučec, M. S., & Bošković, B. (2019, June). The 100-Digit Challenge: Algorithm jDE100. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (pp. 19-26). IEEE.

[29] Bujok, P., Tvrdík, J., & Poláková, R. (2014, July). Differential evolution with rotation-invariant mutation and competing-strategies adaptation. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2253-2258). IEEE.

[30] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, *64*(12), 1695-1724.

[31] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2019). A classification of hyper-heuristic approaches: revisited. In *Handbook of Metaheuristics* (pp. 453-477). Springer, Cham.

[32] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics* (pp. 457-474). Springer, Boston, MA.

[33] Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, *10*(2), 141-171.

[34] Cantú-Paz, E. (1999, July). Topologies, migration rates, and multi-population parallel genetic algorithms. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1* (pp. 91-98).

[35] Cantú-Paz, E. (2001). Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of heuristics*, *7*(4), 311-334.

[36] Castro, L. N., De Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach*. Springer Science & Business Media.

[37] Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering* (pp. 219-239). Springer, Berlin, Heidelberg.

[38] Coello, C. A. C., Lamont, G. B., & Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems* (Vol. 5, pp. 79-104). New York: Springer.

[39] Collette, Y., & Siarry, P. (2013). *Multiobjective optimization: principles and case studies*. Springer Science & Business Media.

[40] Corus, D., & Oliveto, P. S. (2017). Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *22*(5), 720-732.

[41] Corus, D., Oliveto, P. S., & Yazdani, D. (2018, September). Fast artificial immune systems. In *International Conference on Parallel Problem Solving from Nature* (pp. 67-78). Springer, Cham.

[42] Črepinšek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, *45*(3), 1-33.

[43] Das, S., & Suganthan, P. N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, *15*(1), 4-31.

[44] Das, S., Abraham, A., Chakraborty, U. K., & Konar, A. (2009). Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, *13*(3), 526-553

[45] Das, S., Maity, S., Qu, B. Y., & Suganthan, P. N. (2011). Real-parameter evolutionary multimodal optimization—A survey of the state-of-the-art. *Swarm and Evolutionary Computation*, *1*(2), 71-88.

[46] Das, S., Mullick, S. S., & Suganthan, P. N. (2016). Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, *27*, 1-30.

[47] Dasgupta, D. (Ed.). (2012). *Artificial immune systems and their applications*. Springer Science & Business Media.

[48] Dasgupta, D., & Michalewicz, Z. (Eds.). (2013). *Evolutionary algorithms in engineering applications*. Springer Science & Business Media.

[49] Davendra, D., Zelinka, I., & Senkerik, R. (2010). Chaos driven evolutionary algorithms for the task of PID control. *Computers & Mathematics with Applications*, *60*(4), 1088-1104.

[50] Davendra, D., Zelinka, I., Bialic-Davendra, M., Senkerik, R., & Jasek, R. (2013). Discrete self-organising migrating algorithm for flow-shop scheduling with no-wait makespan. *Mathematical and Computer Modelling*, *57*(1-2), 100-110.

[51] Davendra, D., Zelinka, I., Pluhacek, M., & Senkerik, R. (2016). DSOMA—discrete self organising migrating algorithm. In *Self-Organizing Migrating Algorithm* (pp. 51-63). Springer, Cham.

[52] De Castro, L. N., & Von Zuben, F. J. (1999). Artificial immune systems: Part I–basic theory and applications. *Universidade Estadual de Campinas, Dezembro de, Tech. Rep*, *210*(1).

[53] De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., & Tarantino, E. (2007, April). Satellite image registration by distributed differential evolution. In *Workshops on Applications of Evolutionary Computation* (pp. 251-260). Springer, Berlin, Heidelberg.

[54] De Jong, K. (2016, July). Evolutionary computation: a unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion* (pp. 185-199).

[55] Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms* (Vol. 16). John Wiley & Sons.

[56] Deb, K., & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation*, *18*(4), 577-601.

[57] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.

[58] Dempsey, I., O'Neill, M., & Brabazon, A. (2009). *Foundations in grammatical evolution for dynamic environments* (Vol. 194). Heidelberg: Springer.

[59] Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3-18.

[60] Deugo, D., & Ferguson, D. (2004, June). Evolution to the Xtreme: evolving evolutionary strategies using a meta-level approach. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)* (Vol. 1, pp. 31-38). IEEE.

[61] Diep, Q. B. (2019, June). Self-Organizing Migrating Algorithm Team To Team Adaptive–SOMA T3A. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1182-1187). IEEE.

[62] Diep, Q. B., Zelinka, I., & Das, S. (2019, June). Self-organizing migrating algorithm pareto. In *MENDEL* (Vol. 25, No. 1, pp. 111-120).

[63] Diep, Q. B., Zelinka, I., Das, S., & Senkerik, R. (2019). SOMA T3A for Solving the 100-Digit Challenge. In *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing* (pp. 155-165). Springer, Cham.

[64] Digalakis, J., & Margaritis, K. (2003). Parallel Evolutionary Algorithms on Message-Passing Clusters. In *Proceedings of Parallel Computing Conference*.

[65] Dioşan, L., & Oltean, M. (2009). Evolutionary design of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, *10*(3), 263-306.

[66] Doerr, B., & Doerr, C. (2018). Optimal Static and Self-Adjusting Parameter Choices for the $$(1+(\lambda,\lambda)) $$ Genetic Algorithm. *Algorithmica*, *80*(5), 1658-1709.

[67] Doerr, B., Doerr, C., & Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, *567*, 87-104.

[68] Doerr, B., Doerr, C., & Lengler, J. (2019, July). Self-adjusting mutation rates with provably optimal success rules. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1479-1487).

[69] Doerr, C., Wang, H., Ye, F., van Rijn, S., & Bäck, T. (2018). IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv preprint arXiv:1810.05281*.

[70] Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., & Bäck, T. (2020). Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, *88*, 106027.

[71] Domshlak, C., Prestwich, S., Rossi, F., Venable, K. B., & Walsh, T. (2006). Hard and soft constraints for reasoning about qualitative conditional preferences. *Journal of Heuristics*, *12*(4-5), 263-285.

[72] Dorigo, M. (1992). *The metaphor of the ant colony and its application to combinatorial optimization* (Doctoral dissertation, PhD thesis, Politecnico di Milano, Italy).

[73] Dorigo, M., & Di Caro, G. (1999, July). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)* (Vol. 2, pp. 1470-1477). IEEE.

[74] Dorigo, M., & Stützle, T. (2019). Ant colony optimization: overview and recent advances. In *Handbook of metaheuristics* (pp. 311-351). Springer, Cham.

[75] dos Santos Coelho, L. (2009). Self-organizing migrating strategies applied to reliability-redundancy optimization of systems. *IEEE Transactions on Reliability*, *58*(3), 501-510.

[76] dos Santos Coelho, L. (2009). Self-organizing migration algorithm applied to machining allocation of clutch assembly. *Mathematics and Computers in Simulation*, *80*(2), 427-435.

[77] dos Santos Coelho, L., & Mariani, V. C. (2010). An efficient cultural self-organizing migrating strategy for economic dispatch optimization with valve-point effect. *Energy Conversion and Management*, *51*(12), 2580-2587.

[78] Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2019). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*.

[79] Droste, S., Jansen, T., & Wegener, I. (1999, July). Perhaps not a free lunch but at least a free appetizer. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1* (pp. 833-839).

[80] Ducatelle, F., Di Caro, G. A., & Gambardella, L. M. (2010). Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intelligence*, *4*(3), 173-198.

[81] Eberhart, R. C., Shi, Y., & Kennedy, J. (2001). *Swarm intelligence*. Elsevier.

[82] Eiben, A. E., & Bäck, T. (1997). Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, *5*(3), 347-365.

[83] Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing* (Vol. 53, p. 18). Berlin: springer.

[84] Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, *3*(2), 124-141.

[85] El-Mihoub, T. A., Hopgood, A. A., Nolle, L., & Battersby, A. (2006). Hybrid Genetic Algorithms: A Review. *Engineering Letters*, *13*(2), 124-137.

[86] Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*.

[87] Emmerich M., Shir O.M., Wang H. (2018) *Evolution Strategies*. In: Martí R., Panos P., Resende M. (eds) Handbook of Heuristics. Springer, Cham.

[88] Emmerich, M., Beume, N., & Naujoks, B. (2005, March). An EMO algorithm using the hypervolume measure as selection criterion. In *International Conference on Evolutionary Multi-Criterion Optimization* (pp. 62-76). Springer, Berlin, Heidelberg.

[89] Epitropakis, M. G., Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., & Vrahatis, M. N. (2011). Enhancing differential evolution utilizing proximity-based mutation operators. *IEEE Transactions on Evolutionary Computation*, *15*(1), 99-119.

[90] Escobar, J. J., Ortega, J., Díaz, A. F., González, J., & Damas, M. (2019). Energy-aware load balancing of parallel evolutionary algorithms with heavy fitness functions in heterogeneous CPU-GPU architectures. *Concurrency and Computation: Practice and Experience*, *31*(6), e4688.

[91] Espejo, P. G., Ventura, S., & Herrera, F. (2009). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(2), 121-144.

[92] Faris, H., Aljarah, I., Al-Betar, M. A., & Mirjalili, S. (2018). Grey wolf optimizer: a review of recent variants and applications. *Neural computing and applications*, *30*(2), 413-435.

[93] Ferreira, C. (2002). Gene expression programming in problem solving. In *Soft computing and industry* (pp. 635-653). Springer, London.

[94] Ferreira, C. (2006). *Gene expression programming: mathematical modeling by an artificial intelligence* (Vol. 21). Springer.

[95] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in neural information processing systems* (pp. 2962-2970).

[96] Fister, I., Fister Jr, I., Yang, X. S., & Brest, J. (2013). A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, *13*, 34-46.

[97] Fleming, P. J., & Purshouse, R. C. (2002). Evolutionary algorithms in control systems engineering: a survey. *Control engineering practice*, *10*(11), 1223-1241.

[98] Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

[99] Fonseca, C. M., & Fleming, P. J. (1993, June). Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization. In *Icga* (Vol. 93, No. July, pp. 416-423).

[100] Friedrich, T., & Neumann, F. (2017, February). What's hot in evolutionary computation. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[101] Gajda, Z., & Sekanina, L. (2009, May). Gate-level optimization of polymorphic circuits using Cartesian genetic programming. In *2009 IEEE Congress on Evolutionary Computation* (pp. 1599-1604). IEEE.

[102] Ghosh, A., Das, S., Das, A. K., & Gao, L. (2019). Reusing the Past Difference Vectors in Differential Evolution--A Simple But Significant Improvement. *IEEE transactions on cybernetics*.

[103] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. AddisonWesley.

[104] Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning.

[105] Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, *3*(5), 493-530.

[106] Gómez, R. H., & Coello, C. A. C. (2013, June). MOMBI: A new metaheuristic for many-objective optimization based on the R2 indicator. In *2013 IEEE Congress on Evolutionary Computation* (pp. 2488-2495). IEEE.

[107] Guo, S. M., & Yang, C. C. (2014). Enhancing differential evolution utilizing eigenvector-based crossover operator. *IEEE Transactions on Evolutionary Computation*, *19*(1), 31-49.

[108] Guo, S. M., Tsai, J. S. H., Yang, C. C., & Hsu, P. H. (2015, May). A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set. In *2015 IEEE congress on evolutionary computation (CEC)* (pp. 1003-1010). IEEE.

[109] Hadaš, Z., Ondrůšek, Č., & Kurfürst, J. (2010). Optimization of vibration power generator parameters using self-organizing migrating algorithm. In *Recent Advances in Mechatronics* (pp. 245-250). Springer, Berlin, Heidelberg.

[110] Hansen, N. (2006). The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation* (pp. 75-102). Springer, Berlin, Heidelberg.

[111] Hansen, N. (2016). The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

[112] Hansen, N., Auger, A., Mersmann, O., Tusar, T., & Brockhoff, D. (2016). COCO: A platform for comparing continuous optimizers in a black-box setting. *arXiv preprint arXiv:1603.08785*.

[113] Hansen, N., Auger, A., Ros, R., Finck, S., & Pošík, P. (2010, July). Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation* (pp. 1689-1696).

[114] Harman, M. (2011). Software engineering meets evolutionary computation. *Computer*, (10), 31-39.

[115] Helwig, S., Branke, J., & Mostaghim, S. (2012). Experimental analysis of bound handling techniques in particle swarm optimization. *IEEE Transactions on Evolutionary computation*, *17*(2), 259-271.

[116] Hemberg, E. A. P. (2010). *An exploration of grammars in grammatical evolution* (Doctoral dissertation, University College Dublin).

[117] Hildebrandt, T., & Branke, J. (2015). On using surrogates with genetic programming. *Evolutionary computation*, *23*(3), 343-367.

[118] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[119] Horst, R., Pardalos, P. M., & Van Thoai, N. (2000). *Introduction to global optimization*. Springer Science & Business Media.

[120] Islam, S. M., Das, S., Ghosh, S., Roy, S., & Suganthan, P. N. (2011). An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *42*(2), 482-500.

[121] Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, *9*(1), 3-12.

[122] Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, *1*(2), 61-70.

[123] Johnson, C. G. (2003, April). Artificial immune system programming for symbolic regression. In *European Conference on Genetic Programming* (pp. 345-353). Springer, Berlin, Heidelberg.

[124] Kadavy, T., Pluhacek, M., Senkerik, R., & Viktorin, A. (2019, June). The Ensemble of Strategies and Perturbation Parameter in Self-organizing Migrating Algorithm Solving CEC 2019 100-Digit Challenge. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (pp. 372-375). IEEE.

[125] Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, *214*(1), 108-132.

[126] Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, *39*(3), 459-471.

[127] Karaboga, D., Gorkemli, B., Ozturk, C., & Karaboga, N. (2014). A comprehensive survey: artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review*, *42*(1), 21-57.

[128] Karafotias, G., Hoogendoorn, M., & Eiben, Á. E. (2014). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, *19*(2), 167-187.

[129] Kargupta, H. (1996, May). The gene expression messy genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation* (pp. 814-819). IEEE.

[130] Kazemzadeh Azad, S. (2018). Seeding the initial population with feasible solutions in metaheuristic optimization of steel trusses. *Engineering Optimization*, *50*(1), 89-105.

[131] Kennedy, J. (2006). Swarm intelligence. In *Handbook of nature-inspired and innovative computing* (pp. 187-219). Springer, Boston, MA.

[132] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks* (Vol. 4, pp. 1942-1948). IEEE.

[133] Kenneth A.. De Jong. (2006). *Evolutionary Computation: a unified approach*. MIT Press.

[134] Khan, A., Qureshi, A. S., Wahab, N., Hussain, M., & Hamza, M. Y. (2019). A recent survey on the applications of genetic programming in image processing. *arXiv preprint arXiv:1901.07387*.

[135] Kicinger, R., Arciszewski, T., & De Jong, K. (2005). Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, *83*(23-24), 1943-1978.

[136] Komínková Oplatková, Z. (2009). Metaevolution: synthesis of optimization algorithms by means of symbolic regression and evolutionary algorithms.

[137] Koza, J. R. (1994). Genetic programming II: Automatic discovery of reusable subprograms. *Cambridge, MA, USA*, *13*(8), 32.

[138] Koza, J. R. (1995, November). Survey of genetic algorithms and genetic programming. In *Wescon conference record* (pp. 589-594). WESTERN PERIODICALS COMPANY.

[139] Koza, J. R., & Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection* (Vol. 1). MIT press.

[140] Koza, J. R., Andre, D., Keane, M. A., & Bennett III, F. H. (1999). *Genetic programming III: Darwinian invention and problem solving* (Vol. 3). Morgan Kaufmann.

[141] Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., & Lanza, G. (2006). *Genetic programming IV: Routine human-competitive machine intelligence* (Vol. 5). Springer Science & Business Media.

[142] Krause, J., Cordeiro, J., Parpinelli, R. S., & Lopes, H. S. (2013). A survey of swarm algorithms applied to discrete optimization problems. In *Swarm Intelligence and Bio-Inspired Computation* (pp. 169-191). Elsevier.

[143] Krishnakumar, K. (1990, February). Micro-genetic algorithms for stationary and non-stationary function optimization. In *Intelligent control and adaptive systems* (Vol. 1196, pp. 289-296). International Society for Optics and Photonics.

[144] Lardeux, F., & Goëffon, A. (2010, December). A dynamic island-based genetic algorithms framework. In *Asia-Pacific Conference on Simulated Evolution and Learning* (pp. 156-165). Springer, Berlin, Heidelberg.

[145] Le, M. N., Ong, Y. S., Menzel, S., Jin, Y., & Sendhoff, B. (2013). Evolution by adapting surrogates. *Evolutionary computation*, *21*(2), 313-340.

[146] Lehman, J., & Stanley, K. O. (2008, August). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE* (pp. 329-336).

[147] Lehre, P. K., & Oliveto, P. S. (2016, July). Runtime Analysis of Population-based Evolutionary Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion* (pp. 435-462).

[148] Lehre, P. K., & Oliveto, P. S. (2017). Theoretical analysis of stochastic search algorithms. arXiv preprint arXiv:1709.00890.

[149] Li, B., Li, J., Tang, K., & Yao, X. (2015). Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, *48*(1), 1-35.

[150] Li, H., & Deb, K. (2017, June). Challenges for evolutionary multiobjective optimization algorithms in solving variable-length problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2217-2224). IEEE.

[151] Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE transactions on evolutionary computation*, *10*(3), 281-295.

[152] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., & Miikkulainen, R. (2019, July). Evolutionary neural automl for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 401-409).

[153] Lin, F. P. C., & Phoa, F. K. H. (2017, March). A performance study of parallel programming via CPU and GPU on swarm intelligence based evolutionary algorithm. In *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence* (pp. 1-5).

[154] Lin, Y. C., Hwang, K. S., & Wang, F. S. (2004). A mixed-coding scheme of evolutionary algorithms to solve mixed-integer nonlinear programming problems. *Computers & Mathematics with Applications*, *47*(8-9), 1295-1307.

[155] Liu, X. F., Zhan, Z. H., Lin, Y., Chen, W. N., Gong, Y. J., Gu, T. L., ... & Zhang, J. (2018). Historical and heuristic-based adaptive differential evolution. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *49*(12), 2623-2635.

[156] Lozano, M., Herrera, F., & Cano, J. R. (2008). Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences*, *178*(23), 4421-4433.

[157] Lu, G., & Areibi, S. (2004, June). An island-based GA implementation for VLSI standard-cell placement. In *Genetic and Evolutionary Computation Conference* (pp. 1138-1150). Springer, Berlin, Heidelberg.

[158] Lynn, N., & Suganthan, P. N. (2015). Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation. *Swarm and Evolutionary Computation*, *24*, 11-24.

[159] Maier, H. R., Razavi, S., Kapelan, Z., Matott, L. S., Kasprzyk, J., & Tolson, B. A. (2019). Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environmental modelling & software*, *114*, 195-213.

[160] Mallipeddi, R., Suganthan, P. N., Pan, Q. K., & Tasgetiren, M. F. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, *11*(2), 1679-1696.

[161] Mantegna, R. N., & Stanley, H. E. (1994). Stochastic process with ultraslow convergence to a Gaussian: the truncated Lévy flight. *Physical Review Letters*, *73*(22), 2946.

[162] Mavrovouniotis, M., Li, C., & Yang, S. (2017). A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm and Evolutionary Computation*, *33*, 1-17.

[163] Mckay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., & O'neill, M. (2010). Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, *11*(3-4), 365-396.

[164] Meseguer, P., Rossi, F., & Schiex, T. (2006). Soft constraints. In *Foundations of Artificial Intelligence* (Vol. 2, pp. 281-328). Elsevier.

[165] Mezura-Montes, E., & Coello, C. A. C. (2011). Constraint-handling in nature-inspired numerical optimization: past, present and future. *Swarm and Evolutionary Computation*, *1*(4), 173-194.

[166] Michalewicz, Z. (1996). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, *1*(2), 177-206.

[167] Miikkulainen, R. (2019). Creative AI through evolutionary computation. *arXiv preprint arXiv:1901.03775*.

[168] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., ... & Hodjat, B. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing* (pp. 293-312). Academic Press.

[169] Miller, J. F., & Harding, S. L. (2008, July). Cartesian genetic programming. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation* (pp. 2701-2726).

[170] Mininno, E., Neri, F., Cupertino, F., & Naso, D. (2010). Compact differential evolution. *IEEE Transactions on Evolutionary Computation*, *15*(1), 32-54.

[171] Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46-61.

[172] Muhammad, A., Bargiela, A., & King, G. (1997). Fine-grained parallel genetic algorithm: A stochastic optimisation method. In *Proceedings of the first world congress on systems simulation* (pp. 199-203).

[173] Nepomuceno, F. V., & Engelbrecht, A. P. (2013, June). A self-adaptive heterogeneous pso for real-parameter optimization. In *2013 IEEE congress on evolutionary computation* (pp. 361-368). IEEE.

[174] Neri, F., & Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, *2*, 1-14.

[175] Neri, F., & Tirronen, V. (2010). Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, *33*(1-2), 61-106.

[176] Nesmachnow, S., Cancela, H., & Alba, E. (2012). A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Applied Soft Computing, 12(2), 626-639.

[177] Nickabadi, A., Ebadzadeh, M. M., & Safabakhsh, R. (2011). A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied soft computing*, *11*(4), 3658-3670.

[178] Noorian, F., de Silva, A. M., & Leong, P. H. (2016). gramEvol: Grammatical evolution in R. *Journal of Statistical Software*, *71*(1), 1-26.

[179] Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic programming*. W. Banzhaf (Ed.). Springer.

[180] Nowostawski, M., & Poli, R. (1999, June). Parallel genetic algorithm taxonomy. In *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No. 99TH8410)* (pp. 88-92). Ieee.

[181] O'Neill, M., & Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, *5*(4), 349-358.

[182] O'Neil, M., & Ryan, C. (2003). Grammatical evolution. In *Grammatical evolution* (pp. 33-47). Springer, Boston, MA.

[183] Oh, I. S., Lee, J. S., & Moon, B. R. (2004). Hybrid genetic algorithms for feature selection. *IEEE Transactions on pattern analysis and machine intelligence*, *26*(11), 1424-1437.

[184] Ojha, V. K., Abraham, A., & Snášel, V. (2014, November). ACO for continuous function optimization: A performance analysis. In *2014 14th International Conference on Intelligent Systems Design and Applications* (pp. 145-150). IEEE.

[185] Oliveto, P. S., & Witt, C. (2013, July). Improved runtime analysis of the simple genetic algorithm. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (pp. 1621-1628).

[186] Oliveto, P. S., & Witt, C. (2014). On the runtime analysis of the simple genetic algorithm. *Theoretical Computer Science*, *545*, 2-19.

[187] Oliveto, P. S., & Witt, C. (2015). Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, *605*, 21-41.

[188] Oliveto, P. S., He, J., & Yao, X. (2009). Analysis of the $(1+ 1) $-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation*, *13*(5), 1006-1029.

[189] Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., & Moore, J. H. (2016, March). Automating biomedical data science through tree-based pipeline optimization. In *European Conference on the Applications of Evolutionary Computation* (pp. 123-137). Springer, Cham.

[190] Oltean, M., & Dumitrescu, D. (2002). Multi expression programming. *Journal of Genetic Programming and Evolvable Machines, Kluwer, second tour of review*.

[191] Opara, K. R., & Arabas, J. (2019). Differential Evolution: A survey of theoretical analyses. Swarm and evolutionary computation, 44, 546-558.

[192] Oplatkova, Z. K., & Senkerik, R. (2016). Control law and pseudo neural networks synthesized by evolutionary symbolic regression technique. In *Seminal Contributions to Modelling and Simulation* (pp. 91-113). Springer, Cham.

[193] Oplatková, Z. K., Viktorin, A., Senkerik, R., & Urbánek, T. (2017). Different Approaches For Constant Estimation In Analytic Programming. In *ECMS* (pp. 326-332).

[194] Oplatková, Z., & Zelinka, I. (2006, July). Investigation on artificial ant using analytic programming. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 949-950).

[195] Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, *55*(4), 795-816.

[196] Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, *35*(9), 2807-2839.

[197] Parsopoulos, K. E. (2012). Parallel cooperative micro-particle swarm optimization: A master–slave model. *Applied Soft Computing*, *12*(11), 3552-3579.

[198] Paterson, N. R. (2003). *Genetic programming with context-sensitive grammars* (Doctoral dissertation, University of St Andrews).

[199] Piotrowski, A. P. (2017). Review of differential evolution population size. *Swarm and Evolutionary Computation*, *32*, 1-24.

[200] Piotrowski, A. P. (2018). L-SHADE optimization algorithms with population-wide inertia. *Information Sciences*, *468*, 117-141.

[201] Piotrowski, A. P., & Napiorkowski, J. J. (2018). Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure?. *Swarm and evolutionary computation*, *43*, 88-108.

[202] Poláková, R., Tvrdík, J., & Bujok, P. (2017). Adaptation of population size according to current population diversity in differential evolution. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1-8). IEEE.

[203] Poli, R., Langdon, W. B., McPhee, N. F., & Koza, J. R. (2007). Genetic programming: An introductory tutorial and a survey of techniques and applications. *University of Essex, UK, Tech. Rep. CES-475*.

[204] Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media.

[205] Qin, A. K., Huang, V. L., & Suganthan, P. N. (2008). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, *13*(2), 398-417.

[206] Qin, Q., Cheng, S., Zhang, Q., Wei, Y., & Shi, Y. (2015). Multiple strategies based orthogonal design particle swarm optimizer for numerical optimization. *Computers & Operations Research*, *60*, 91-110.

[207] Qu, B. Y., Suganthan, P. N., & Liang, J. J. (2012). Differential evolution with neighborhood mutation for multimodal optimization. *IEEE transactions on evolutionary computation*, *16*(5), 601-614.

[208] Qu, B. Y., Zhu, Y. S., Jiao, Y. C., Wu, M. Y., Suganthan, P. N., & Liang, J. J. (2018). A survey on multi-objective evolutionary algorithms for the solution of the environmental/economic dispatch problems. *Swarm and Evolutionary Computation*, *38*, 1-11.

[209] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., ... & Kurakin, A. (2017, August). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 2902-2911). JMLR. org.

[210] Riget, J., & Vesterstrøm, J. S. (2002). A diversity-guided particle swarm optimizer-the ARPSO. *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, *2*, 2002.

[211] Rocco, C. M., Barker, K., Moronta, J., & Ramirez-Marquez, J. E. (2018). Multiobjective Formulation for Protection Allocation in Interdependent Infrastructure Networks Using an Attack-Diffusion Model. *Journal of Infrastructure Systems*, *24*(1), 04018002.

[212] Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties: technical report*.

[213] Rozenberg, G., Bäck, T., & Kok, J. N. (Eds.). (2012). *Handbook of natural computing* (pp. 461-477). Springer Berlin Heidelberg.

[214] Rudolph, G., & Agapie, A. (2000, July). Convergence properties of some multi-objective evolutionary algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)* (Vol. 2, pp. 1010-1016). IEEE.

[215] Ryan, C., Collins, J. J., & Neill, M. O. (1998, April). Grammatical evolution: Evolving programs for an arbitrary language. In *European Conference on Genetic Programming* (pp. 83-96). Springer, Berlin, Heidelberg.

[216] Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

[217] Salustowicz, R., & Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary computation*, *5*(2), 123-141.

[218] Sekaj, I., & Oravec, M. (2009). Selected population characteristics of fine-grained parallel genetic algorithms with re-initialization. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation* (pp. 945-948).

[219] Senkerik, R., Kadavy, T., Viktorin, A., & Pluhacek, M. (2019, June). Ensemble of Strategies and Perturbation Parameter Based SOMA for Constrained Technological Design Optimization Problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2872-2877). IEEE.

[220] Senkerik, R., Oplatkova, Z., Zelinka, I., & Davendra, D. (2013). Synthesis of feedback controller for three selected chaotic systems by means of evolutionary techniques: Analytic programming. *Mathematical and Computer Modelling*, *57*(1-2), 57-67.

[221] Senkerik, R., Viktorin, A., Pluhacek, M., Kadavy, T., & Zelinka, I. (2017, June). Differential evolution driven analytic programming for prediction. In *International Conference on Artificial Intelligence and Soft Computing* (pp. 676-687). Springer, Cham.

[222] Shi, Y., & Eberhart, R. (1998, May). A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)* (pp. 69-73). IEEE.

[223] Shlesinger, M. F., & Klafter, J. (1986). Lévy walks versus Lévy flights. In *On growth and form* (pp. 279-283). Springer, Dordrecht.

[224] Schaffer, J. D. (1984). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms, *PhD thesis*, Vanderbilt University, USA.

[225] Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic*

algorithms and their applications, 1985. Lawrence Erlbaum Associates. Inc., Publishers.

[226] Schutze, O., Esquivel, X., Lara, A., & Coello, C. A. C. (2012). Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, *16*(4), 504-522.

[227] Schwefel, H. P. (1992). Imitating evolution: Collective, two-level learning processes. *Explaining Process and Change–Approaches to Evolutionary Economics*, 49-63.

[228] Schwefel, H. P., & Rudolph, G. (1995, June). Contemporary evolution strategies. In *European conference on artificial life* (pp. 891-907). Springer, Berlin, Heidelberg.

[229] Sinha, A., Malo, P., & Deb, K. (2017). A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, *22*(2), 276-295.

[230] Smith, J. E., & Fogarty, T. C. (1997). Operator and parameter adaptation in genetic algorithms. *Soft computing*, *1*(2), 81-87.

[231] Smith, S. L. (2018, July). Medical applications of evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1141-1169).

[232] Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, *22*(1), 3-18.

[233] Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *computer*, *27*(6), 17-26.

[234] Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341-359.

[235] Sudholt, D. (2015). Parallel evolutionary algorithms. In *Springer Handbook of Computational Intelligence* (pp. 929-959). Springer, Berlin, Heidelberg.

[236] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for

training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

[237] Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of genetic algorithms* (Vol. 1, pp. 94-101). Elsevier.

[238] Talbi, E. G. (2009). *Metaheuristics: from design to implementation* (Vol. 74). John Wiley & Sons.

[239] Talbi, E. G. (2019). A unified view of parallel multi-objective evolutionary algorithms. *Journal of Parallel and Distributed Computing*, *133*, 349-358.

[240] Tanabe, R., & Fukunaga, A. (2013, June). Success-history based parameter adaptation for differential evolution. In *2013 IEEE congress on evolutionary computation* (pp. 71-78). IEEE.

[241] Tanabe, R., & Fukunaga, A. S. (2014, July). Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)* (pp. 1658-1665). IEEE.

[242] Tang, L., Dong, Y., & Liu, J. (2014). Differential evolution with an individual-dependent mechanism. *IEEE Transactions on Evolutionary Computation*, *19*(4), 560-574.

[243] Tavares, J., Machado, P., Cardoso, A., Pereira, F. B., & Costa, E. (2004, April). On the evolution of evolutionary algorithms. In *European Conference on Genetic Programming* (pp. 389-398). Springer, Berlin, Heidelberg.

[244] Truong, T. C., Diep, Q. B., Zelinka, I., & Senkerik, R. (2019). Pareto-Based Self-organizing Migrating Algorithm Solving 100-Digit Challenge. In *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing* (pp. 13-20). Springer, Cham.

[245] Tušar, T., & Filipič, B. (2014). Visualization of Pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method. *IEEE Transactions on Evolutionary Computation*, *19*(2), 225-245.

[246] Urbánek, T., Prokopova, Z., Silhavy, R., & Kuncar, A. (2016). New Approach Of Constant Resolving Of Analytical Programming. In *ECMS* (pp. 231-236).

[247] Urbanek, T., Prokopova, Z., Silhavy, R., & Vesela, V. (2015). Prediction accuracy measurements as a fitness function for software effort estimation. *SpringerPlus*, *4*(1), 778.

[248] Van Luong, T., Melab, N., & Talbi, E. G. (2010, July). Parallel hybrid evolutionary algorithms on GPU. In *IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE.

[249] Vanneschi, L., Castelli, M., & Silva, S. (2014). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, *15*(2), 195-214.

[250] Vařacha, P. (2013). Asynchronous synthesis of a neural network applied on head load prediction. In *Nostradamus: Modern Methods of Prediction, Modeling and Analysis of Nonlinear Systems* (pp. 225-240). Springer, Berlin, Heidelberg.

[251] Vasicek, Z., & Sekanina, L. (2012, June). On area minimization of complex combinational circuits using cartesian genetic programming. In *2012 IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE.

[252] Viktorin, A., Pluhacek, M., Oplatková, Z. K., & Senkerik, R. (2016). Analytical Programming With Extended Individuals. In *ECMS* (pp. 237-244).

[253] Viktorin, A., Senkerik, R., Pluhacek, M., Kadavy, T., & Zamuda, A. (2019). Distance based parameter adaptation for success-history based differential evolution. *Swarm and Evolutionary Computation*, *50*, 100462.

[254] Wang, L., Pan, Q. K., Suganthan, P. N., Wang, W. H., & Wang, Y. M. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, *37*(3), 509-520.

[255] Weber, M., Neri, F., & Tirronen, V. (2009). Distributed differential evolution with explorative–exploitative population families. *Genetic Programming and Evolvable Machines*, *10*(4), 343.

[256] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, *1*(1), 67-82.

[257] Wu, G., Shen, X., Li, H., Chen, H., Lin, A., & Suganthan, P. N. (2018). Ensemble of differential evolution variants. *Information Sciences*, *423*, 172-186.

[258] Xu, S., & Rahmat-Samii, Y. (2007). Boundary conditions in particle swarm optimization revisited. *IEEE Transactions on Antennas and Propagation*, *55*(3), 760-765.

[259] Xue, B., Zhang, M., Browne, W. N., & Yao, X. (2015). A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, *20*(4), 606-626.

[260] Yang, X. S. (2008). Firefly algorithm. *Nature-inspired metaheuristic algorithms*, *20*, 79-90.

[261] Yang, X. S. (2010). Firefly algorithm, Levy flights and global optimization. In *Research and development in intelligent systems XXVI* (pp. 209-218). Springer, London.

[262] Yang, X. S., & He, X. (2013). Firefly algorithm: recent advances and applications. *arXiv preprint arXiv:1308.3898*.

[263] Yang, Y., Wu, J., Sun, X., Wu, J., & Zheng, C. (2013). Development and application of a master-slave parallel hybrid multi-objective evolutionary algorithm for groundwater remediation design. *Environmental earth sciences*, *70*(6), 2481-2494.

[264] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423-1447.

[265] Yuan, X., Su, A., Nie, H., Yuan, Y., & Wang, L. (2009). Application of enhanced discrete differential evolution approach to unit commitment problem. *Energy Conversion and Management*, *50*(9), 2449-2456.

[266] Zaharie, D. (2009). Influence of crossover on the behavior of differential evolution algorithms. *Applied soft computing*, *9*(3), 1126-1138.

[267] Zamuda, A. (2019, July). Function evaluations upto 1e+ 12 and large population sizes assessed in distance-based success history differential evolution for 100-digit challenge and numerical optimization scenarios (DISHchain 1e+ 12) a competition entry for" 100-digit challenge, and four other numerical optimization competitions" at the genetic and evolutionary computation conference (CECCO) 2019. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 11-12).

[268] Zamuda, A., & Brest, J. (2015). Self-adaptive control parameters' randomization frequency and propagations in differential evolution. *Swarm and Evolutionary Computation*, *25*, 72-99.

[269] Zamuda, A., Nicolau, M., & Zarges, C. (2018, July). A black-box discrete optimization benchmarking (BB-DOB) pipeline survey: taxonomy, evaluation, and ranking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1777-1782).

[270] Zelinka, I. (2004). SOMA—self-organizing migrating algorithm. In *New optimization techniques in engineering* (pp. 167-217). Springer, Berlin, Heidelberg.

[271] Zelinka, I. (2016). SOMA—self-organizing migrating algorithm. In *Self-Organizing Migrating Algorithm* (pp. 3-49). Springer, Cham.

[272] Zelinka, I., Davendra, D., Jasek, R., Senkerik, R., & Oplatkova, Z. (2011). *Analytical programming-a novel approach for evolutionary synthesis of symbolic structures*. INTECH Open Access Publisher.

[273] Zelinka, I., Oplatkova, Z., & Nolle, L. (2005). Analytic programming–Symbolic regression by means of arbitrary evolutionary algorithms. *International Journal of Simulation: Systems, Science and Technology*, *6*(9), 44-56.

[274] Zelinka, I., Skanderova, L., Saloun, P., Senkerik, R., & Pluhacek, M. (2014). Chaos powered symbolic regression in be stars spectra modeling. In *ISCS 2013: Interdisciplinary Symposium on Complex Systems* (pp. 131-139). Springer, Berlin, Heidelberg.

[275] Zhan, Z. H., Zhang, J., Li, Y., & Shi, Y. H. (2010). Orthogonal learning particle swarm optimization. *IEEE transactions on evolutionary computation*, *15*(6), 832-847.

[276] Zhang, J., & Sanderson, A. C. (2009). JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, *13*(5), 945-958.

[277] Zhang, S. X., Zheng, S. Y., & Zheng, L. M. (2017). An efficient multiple variants coordination framework for differential evolution. *IEEE transactions on cybernetics*, *47*(9), 2780-2793.

[278] Zhang, W. J., Xie, X. F., & Bi, D. C. (2004, June). Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)* (Vol. 2, pp. 2307-2311). IEEE.

[279] Zhou, A., Qu, B. Y., Li, H., Zhao, S. Z., Suganthan, P. N., & Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, *1*(1), 32-49.

[280] Zhou, X. G., & Zhang, G. J. (2018). Differential evolution with underestimation-based multimutation strategy. *IEEE transactions on cybernetics*, *49*(4), 1353-1364.

[281] Zitzler, E., & Künzli, S. (2004, September). Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature* (pp. 832-842). Springer, Berlin, Heidelberg.

[282] Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation*, *3*(4), 257-271.

LIST OF SYMBOLS AND ABBREVIATIONS

ABC	artificial bee colony
AIS	artificial immune systems
AP	analytic programming
AutoML	automated machine learning
CEC	congress on evolutionary computation
CF	cost function
CPU	central processing unit
CR	crossover rate in differential evolution
CUDA	Compute Unified Device Architecture
D	dimension of optimization problem
DE	differential evolution
EA	evolutionary algorithm
ECT	evolutionary computation (computing) techniques
ES	evolutionary strategies
F	mutation (scaling) factor in differential evolution
FA	firefly algorithm
FES	cost function evaluations
GA	genetic algorithm
gBest	best solution shared within the whole swarm (PSO)
GE	grammar evolution
GFS	general function set
GP	genetic programming
HPC	high performance computing
MOEA	multi objective evolutionary algorithms

MOO	multi objective optimization
MOP	multi objective problem
NFL	no free lunch theorem
NP	population size
pBest	personal best solution for individual (in PSO)
PSO	particle swarm optimizer
SOMA	self-organizing migrating algorithm
TSP	travelling salesman problem

LIST OF PICTURES