Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Bachelor in Computer Science 15.0 credits

EMERGING EVOLUTIONARY ALGORITHMS FOR CONTINUOUS OPTIMIZATION AND MACHINE LEARNING

Leslie Dahlberg
ldg14001@student.mdh.se

Examiner: Mälardalen University, Västerås, Sweden

Supervisor: Ning Xiong
Mälardalen University, Västerås, Sweden

February 22, 2017

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed suscipit eu ante id aliquet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris condimentum iaculis erat, eget pretium leo consequat eget. Donec eu aliquam purus. Quisque scelerisque feugiat ligula, non vehicula urna blandit nec. Etiam a dignissim purus, at molestie dui. Vestibulum a venenatis mi, id hendrerit sem. Cras purus elit, dapibus eget est id, lacinia tincidunt neque. Mauris eu arcu ipsum. Sed enim leo, iaculis vitae nisl tincidunt, ultricies accumsan diam. Donec neque urna, pellentesque sit amet interdum a, pretium ac enim. Proin quis ante nunc. Fusce vitae lorem condimentum, porta velit blandit, commodo nisi.

Table of Contents

1 Introduction

Optimization is a problem-solving method which aims to find the most advantageous parameters for a given model. The model is known to the optimizer and accepts inputs while producing outputs. Usually the problem is formulated in such a manner that the desired output is the smallest possible one, therefore the process is referred to as minimization. The practicality of this is apparent when considering, for example, optimizing a circuit's power consumption since the desired power consumption is the lowest one possible. To achieve this the optimizer looks for combinations of inputs which produce this desirable output [**?**].

When dealing with simple mathematical models, optimization can be achieved using analytical methods, often calculating the derivative of the functional model, but these methods prove difficult to adapt to complex models which exhibit noisy behavior. The field of evolutionary computation (EC), a subset of computational intelligence (CI), which is further a subset of artificial intelligence (AI), contains algorithms which are well suited to solving these kinds of optimization problems [**?**, **?**].

Evolutionary computation focuses on problem solving algorithms which draw inspiration from natural processes. It is closely related to the neighboring field of swarm intelligence (SI), which often is, and in this thesis will be, included as a subset of EC. The basic rationale of the field is to adapt the mathematical models of biological Darwinian evolution to optimization problems. The usefulness of this can be illustrated by imagining that an organism acts as an "input" to the "model" of it's natural environment and produces an "output" in the form of offspring. Through multiple iterations biological evolution culls the population of organisms, only keeping the fit specimen, to produce organisms which become continuously better adapted to their environments. Evolutionary computation is, however, not merely confined to Darwinian evolution, but also includes a multitude of methods which draw from other natural processes such as cultural evolution and animal behavior [**?**].

1.1 Problem formulation

The purpose of this thesis is to explore the performance and usefulness of three emerging evolutionary algorithms: differential evolution, particle swarm optimization and estimation of distribution algorithms. The intention is to test compare these against each other on a set of benchmark functions and practical problems in machine learning and then, if possible, develop a new or modified algorithm which improves upon then aforementioned ones in some aspect. The research questions are summarized bellow:

1. Benchmark differential evolution, particle swarm optimization and estimation of distribution algorithms on standard benchmark functions
2. Benchmark differential evolution, particle swarm optimization and estimation of distribution algorithms on machine learning problems
3. Develop an improved algorithm which is inspired by the three aforementioned algorithms
4. Include the new algorithm in the first two benchmarks

1.2 Limitations

The scope of this work limits the number of algorithms which can be included in the testing. The individual algorithms also have numerous variations and parameters which can dramatically affect their behavior and it will not be possible to take all these considerations into account. Furthermore, the benchmarking will be restricted to a standard set of testing functions which may or may not provide reliable information regarding the general usability the algorithms. Since evolutionary algorithms have a large number of potential and actual use cases the practical testing will only concern a small subset of the these and may therefore not provide accurate data for all possible use cases.

2 Background

This section will aim to provide a general overview of the field of evolutionary compution. General terms and procedured which are often utizilied in EC will be explained and the most well known traditional approaches will be presented. Current research will also be introduced. The emerging algorithms relevant to this thesis can be found in the section "Theory".

2.1 The basic structure of evolutionary algorithms

Evolutionary algorithms work on the concept of populations. A population is a set of individuals which in the case of optimization problems contain a vector of parameters which the model we wish to optimize can accept and transform into an output. The population is initialize by some procedure to contain a random set of parameter vectors, these should cover the whole parameter range of the model uniformely. The inital population is evaluated and an iterative process is started which continues as long as no suitable solution is found. During this iterative process the current population is selected, altered and evaluated. During selection a set of individuals which display promising characterisitcs are selected to live on to the next generation of the population. They are then altered randomly to create diversity in the population and evaluated. This process created a new generation of the population on each iteration and continues until a solution is found or some other restriction is encoutered [**?**]. The concept is demonstrated in figure 1 with $P(t)$ representing the population at generation t.

$t \leftarrow 0$
initialize $P(t)$
evaluate $P(t)$
while termination-condition not fulfilled **do**
    $t \leftarrow t + 1$
    select $P(t)$ from $P(t-1)$
    alter $P(t)$
    evaluate $P(t)$
end while

Figure 1: Basic evolutionary algorithm

2.2 Components of evolutionary algortihms

Here the fundamental builing blocks of evolutionary algorithms will be presented and explained. Most of these term are universal to all approaches which will be covered in this thesis and neccesary to properly understand them.

2.2.1 Representation

The first step in using evolutionary algorithms is creating a representation which can encode all possible solutions to the problem at hand. Here two different terms are distinguished. The term phenotype denotes the representation that can be directly applied to the problem and the genotype denotes the specific encoding of the phenotype which is manipulated inside the evolutionary algorithm. In optimization tasks a valid phenotype could be a vector of integer numbers which act as parameters to a function while the genotype would be a binary representation of the numbers which can be altered by manipulating individuals bits. The terms phenotype, candidate solution and individual are used interchangably to denote the representation as used by the model while chromosome, genotype and individual are used to refer to the representation inside the evolutionary algorithm [**?**].

2.2.2 Evaluation function

The evalutation function is responsible for improvement in the population. It is a function which assings a fitness or cost value to every genotype and thus enables us to compare the quality of

the genotypes in the population. It also the only information about the problem that it available to the evolutionary algorithm and should therefore include all domain knowledge that's available about the problem [**?**]. The evaluation is also the process which takes up the most computational resources, 99% of the total computational cost in real-world problems [**?**].

2.2.3 Population

The population is a set of genotypes which contain the current best solutions to a problem. While genotypes remain stable and unchanging, the population continually changes through the application of selection mechanisms which decide which genotypes are allow into the next generaiton of the population. The size of the population almost always remains constant during the lifetype of the algorithm. This in turn creates selection pressure which pushes the population to improve. A populations diversity is the measure of difference amoung the genotypes, phenotypes or fitness values [**?**].

2.2.4 Parent selection mechanism

Parent selection serves to improve the quality of a population by selecting which individuals will survive to the next generation. The selected individuals are called parents as they usualy undergo some form of alteration or combination with other individuals before progressing to the next generation. The selection method is usually probabilistic and given better solutions a higher probability and worse solutions a lower probability to survive. It's important that bad solutions still recieve a positive probabilty since the population might otherwise lose diversity and coalesce around a false local optimimum [**?**].

2.2.5 Variation operators

Variation operator introduce new features into the genotypes of a population by modifying or mixing existing genotypes. They can be divided into two types: unary operators which take one genotype and stochastically alter it to introduce random change and n-ary operators which mix the features of 2 or more genotypes. Unary operators called mutation operators while n-ary operators are called cross-over or recombination operators. The biological equivalents to the are random mutation and sexual reproduction. Mutation operators allow evolutionary algorithms to theoretically span the whole continuum of the search space by giving a non-zero probability that a genotype mutates into any other other genotype. This has been used to formally prove that evolutionary algorithms will always reach the desired optimum given enough time. Recombination tries to create new superior individuals by combining the genes of two good parent genotypes [**?**, **?**].

2.2.6 Survivor selection mechanism

Survivor selection takes placa after new offspring have been generated determines which individuals are allowed to live on into the next generation. It is often refered to as the replacement strategy and contrary to parent selection it is usually deterministic. Two popular mechanisms are fitness-based selection and age-based selection. Fitness-based selection determines the next generation by selecting the individuals with the highest fitness while age-based selection allows only the offspring to survive [**?**].

2.2.7 Initatilization

Initialization is the process during which the intial population is generater. The genotypes are usually generated randomly from a uniform distribution based on some range of acceptable input values. If a good solution is known before hand variations of it can be include in the initial population as a bias, but this can sometimes cause more problems than it solves [**?**].

2.2.8 Termination condition

The terminatin condidion determines for how long the algorithm is run. Four criteria are used to determine when to stop [**?**]:

1. If a maximum number CPU-cycles or iterations is reached
2. If a known optimum is reached
3. If the fitness value does not imrove for a considerable amount of time
4. If the diversity of the polation drops below a given threshold

2.3 Main paradighms of evolutionary computation

Below the main paradighms of evolutionary computation will be discussed. They include genetic algorithms, evolution strategy, evolutionary programming and genetic programming.

2.3.1 Genetic algorithms

Genetic algorithms (GA) were introduced by John Holland in the 1960s as an attempt to apply biological adaptation to computational problems. GAs are multidemensional search algorithms which use populations of binary strings called chromosomes to evolve a solution to a problem. GAs use a selection operator, a mutation operator and a cross-over operator. The selection operator select individuals which are subjected to cross-over based on their fitness and cross-over combines their genetic material to form new individuals which are then randomly mutated [**?**]. See figure 2 for a simple GA.

Initialize a population of N binary chromosome with L bits
while termination-condition not fulfilled **do**
  Evaluate the fitness $F(x)$ of each chromosome x
  repeat
    Select two chromosomes probabilistally from the population
    based on their fitness
    With the cross-over probability P_c create two new offspring
    from the two selected chromosomes using the crossover operator.
    Otherwise create two new offspring identical to their parent chromosomes.
    Mutate the two chromosomes using the mutation probability P_m
    and place the resulting chromosomes into the new population
  until N offspring have been created
  Replace the old population with the new population
end while

Figure 2: Basic genetic algorithm

2.3.2 Evolution strategy

Evolution strategies (ES) were first developed to solve parameter optimization tasks. They differ from GAs by representing individuals using a pair of vectors $\vec{v} = (\vec{x}, \vec{\sigma})$. The earliest versions of ES used a population of only one individual and only utilized the mutation operator. New individuals were only introduced into the population if the performed better than their parents. The vector $\vec{x}$ represents the position in the search space and $\vec{\sigma}$ represents a vector standard deviations used to generate new individuals. Mutation occurs according to equation 1 where $N(0, \vec{\sigma})$ is a vector containing random Gaussian numbers with the mean 0 and a standard deviation of $\vec{\sigma}$ [**?**].

$$\vec{x}^{t+1} = \vec{x}^t + N(0, \vec{\sigma}) \tag{1}$$

Newer versions of the algorithm include $(\mu + \lambda) - \text{ES}$ and $(\mu, \lambda) - \text{ES}$. The main point of these it that their parameters like mutation variance adapt automatically to the problem. In $(\mu + \lambda) - \text{ES}$ μ parents generate λ offspring and the new generation is selected from μ and λ while in $(\mu, \lambda) - \text{ES}$ μ parents generate λ offspring $(\lambda > \mu)$ and the new generation is only selected from λ. These algorithms produce offspring by first applying cross-over to combine two parent chromosomes (including their deviation vectors $\vec{\sigma}$) and then mutating $\vec{x}$ and $\vec{\sigma}$ [**?**].

2.3.3 Evolutionary programming

Evolutionary programming (EP) was created as an alternative approach to artificial intelligence. The idea was to evolve finite state machines (FSM) which observe the environment and elicit appropriate responses [**?**]. The environmet is modeled as a sequence of input characters select from an alphabet and the role of the FSM is to produce the next character in sequence. The fitness of an FSM is measured by function which tests the FSM an a sequence of input characters, starting with the first character and the progressing to include on more addition character on each iteration. The function measures the correct prediction rate of the FSM and determines it's score [**?**].

Each FSM creates one offspring which is mutated by one or more of the following operators: change of input symbol, change of state transition, addition of state, deletion of state and change of initial state. The next generation is then selected from the pool of parents and offspring, selecting the best 50% of all available solutions. A general form of EP has recently been devised which can tackle continuous optimization tasks [**?**].

2.3.4 Genetic programming

Genetic programming (GP) differs from traditional genetic algorithms by evolving computer programs which solve problems instead of directly finding the solution to a problem. The individuals in the population are therefore data-structures which encode computer programs, usually rooted trees representing expressions.

At it's most basic the programs are defines as functions which take a set of input parameters and produce an output. The programs are constructed from building block such as variables, numbers and operators. The initial population contains a set of such programs which have been as random trees.

The evolution process is similar to GAs in that the programs are evaluated using a function which runs a set of test cases and the programs undergo cross-over and other mutation. Cross-over is defines as the exchange of subtrees between programs and produces two offspring from two parents.

More advanced versions of GP include function calls which enable the programs to remember useful symmetries and regularities and facilitate code reuse [**?**].

2.4 Current research

Ideas around evolutionary computation began emerging in 1950s. Several researchers independently from each-other created algorithms which were inspired by natural Darwinian principles, these include Hollands Genetic Algorithms, Schwefel's and Rechenberg's Evolution Strategies and Fogel's Evolutionary Programming. These pioneering algorithms shared the concepts of populations, individuals, offspring and fitness and compared to natural systems they were quite simplistic, lacking gender, maturation processes, migration, etc.

Research has shown that no single algorithm can perform better than all other algorithms on average. This has been referred to as the 'no-free-lunch' and current solutions instead aim at finding better solutions to specific problems by exploiting inherent biases in the problem. This has led to the desire to classify different algorithms in order to decide which algorithms should be used which situations, a problem which is not easy.

Recent research has focused, among others, on parallelism, multi-population models, multi-objective optimization, dynamic environments and evolving executable code. Parallelism can easily be exploited in EC because of it's inherently parallel nature, e.g each individual in a population can be evaluated, mutated and crossed-over independently. Multi-core CPUs, massively parallel GPUs, clusters and networks can be used to achieve this. Multi-populations models mimic the way species depend on each-other in nature. Examples of this include host-parasite and predator-prey relationships where the the individual's fitness is connected to the fitness of another individual. Multi-objective optimization aims to solve problems where conflicting interests exist, a good example would be optimizing for power and fuel-consumption simultaneously. In such problems the optimization algorithm has to keep two or more interests in mind simultaneously and find intersections points which offer the best trade-offs between them. Dynamic environments include things

like the stock markets and traffic systems. Traditional EAs perform badly in these situations but they can perform well when slightly modified to fit the task. Evolving executable code, as in Genetic Programming and Evolutionary Programming, is a hard problem with very interesting potential applications. Most often low-level code such as assembly, lisp or generic rules are evolved [**?**].

2.5 Applications and use-cases

Evolutionary computing has found uses in a variety of fields. Some of them include include

1. Power-Systems (planning, voltage-control, load-forecasting, maintenance scheduling, fault-diagnosis, control and configuration) [**?**]
2. Cloud-Computing (resource management) [**?**]
3. Finance (portfolio optimization) [**?**, **?**]
4. Biology (protein structure prediction) [**?**, **?**]
5. Economics (evolutionary economic models) [**?**]
6. Disaster planning (planning, scheduling) [**?**]
7. Medicine (image processing) [**?**]
8. Intrusion Detection Systems [**?**]

3 Method

The initial research phase consisted of a literature study of evolutionary computing in general and then focused on differential evolution, particle swarm optimization and estimation of distribution algorithms, the purpose of which was to understand the state of the art in evolutionary computing and gain the necessary knowledge to evaluate different evolutionary methods and create a new improved algorithm.

3.1 Tools and environments

Matlab was used to program all algorithms, benchmark-functions and test-cases. Matlab's neural network package facilitated the testing of neural networks. All graphs we're drawn using Matlab's plotting functions.

3.2 Benchmarking functions

The functions for the benchmark are taken from the 2005 CEC conference on continuous evolutionary optimization [**?**]. The functions are losely based on the popular optimization benchmark suite created by DeJong [**?**].

For all functions $x = [x_1, x_2, x_3, ..., x_D]$ are the input parameters, $o = [o_1, o_2, o_3, ..., o_D]$ is the global optimum, D is the dimension and M is an orthogonal matrix with parameters unique to each function. The matrices for o and M are available in appendix A. Illustrations of the functions can be found in figures 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12.

3.2.1 F_1: Shifted Sphere Function

$$F_1(x) = \sum_{i=1}^{D} z_i^2 \tag{2}$$
$$z = x - o$$
$$x \in [-100, 100]^D$$

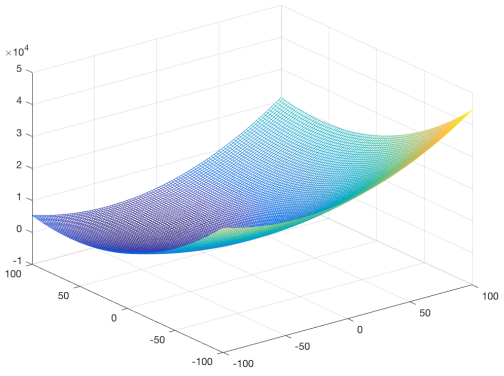

Figure 3: 3-D map for 2-D function

3.2.2 F_2: Shifted Schwefel's Problem

$$F_2(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} z_j)^2 \tag{3}$$
$$z = x - o$$
$$x \in [-100, 100]^D$$

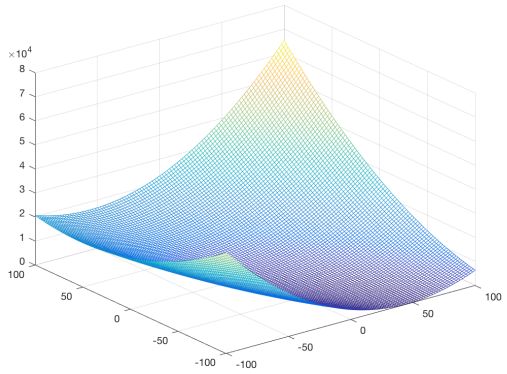

Figure 4: 3-D map for 2-D function

3.2.3 F_3: Shifted Rotated High Conditioned Elliptic Function

$$F_3(x) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} z_i^2 \tag{4}$$
$$z = (x - o) * M$$
$$x \in [-100, 100]^D$$

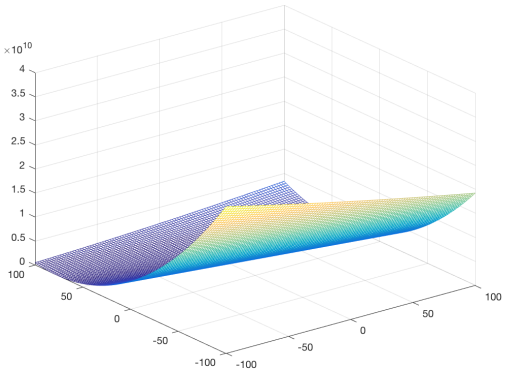

Figure 5: 3-D map for 2-D function

3.2.4 F_4: Shifted Schwefel's Problem with Noise in Fitness

$$F_4(x) = (\sum_{i=1}^{D} (\sum_{j=1}^{i} z_j)^2) * (1 + 0.4|N(0,1)|) \tag{5}$$
$$z = x - o$$
$$x \in [-100, 100]^D$$

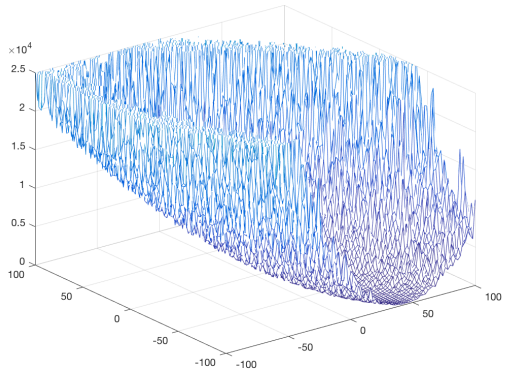

Figure 6: 3-D map for 2-D function

3.2.5 F_5: Schwefel's Problem with Global Optimum on Bounds

$$F_5(x) = max\{|A_i x - B_i|\} \tag{6}$$

$$i = 1, ..., D, x \in [-100, 100]^D$$

$$A \text{ is a } D * D \text{ matrix}, a_{ij} = \text{ random numbers in } [-500, 500], det(A) \neq 0$$

$$B_i = A_i * o, o_i = \text{ random numbers in } [-100, 100]$$

$$o_i = -100, \text{ for } i = 1, 2, ..., [D/4], o_i = 100, \text{ for } i = [3D/4], ..., D$$

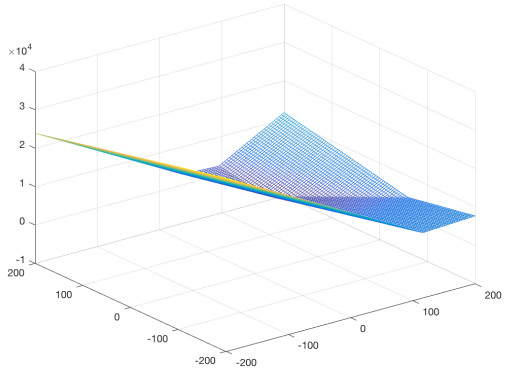

Figure 7: 3-D map for 2-D function

3.2.6 F_6: Shifted Rosenbrock's Function

$$F_6(x) = \sum_{i=1}^{D-1} (100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) \tag{7}$$

$$z = x - o + 1$$

$$x \in [-100, 100]^D$$

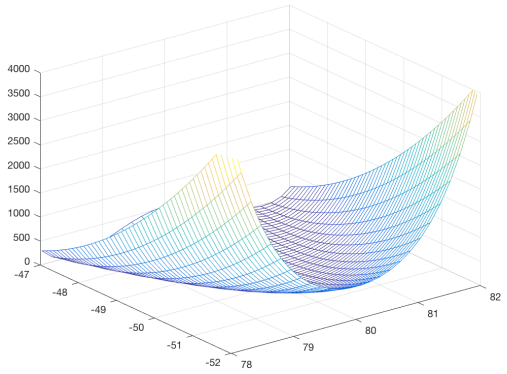

Figure 8: 3-D map for 2-D function

3.2.7 F_7: Shifted Rotated Griewank's Function without Bounds

$$F_7(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos \frac{z_i}{\sqrt{i}} + 1 \tag{8}$$

$$z = (x - o) * M$$

$$x \in [0, 600]^D$$

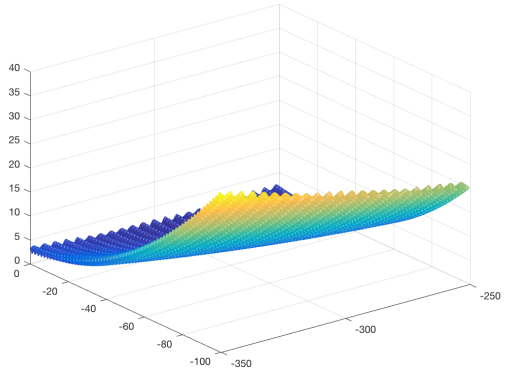

Figure 9: 3-D map for 2-D function

3.2.8 F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds

$$F_8(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{D} z_i^2}\right) - \exp\left(\frac{1}{D} \sum_{i=1}^{D} \cos\left(2\pi z_i\right)\right) + 20 + e \tag{9}$$

$$z = (x - o) * M$$

$$x \in [-32, 32]^D$$

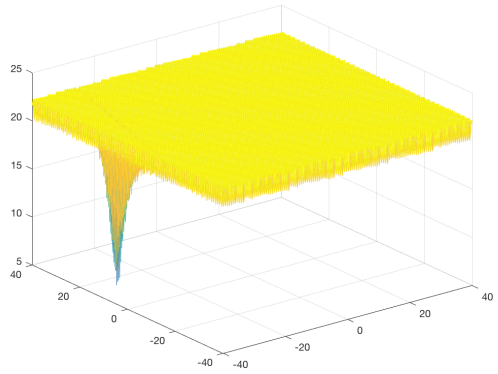

Figure 10: 3-D map for 2-D function

3.2.9 F_9: Shifted Rastrigin's Function

$$F_9(x) = \sum_{i=1}^{D} z_i^2 - 10\cos(2\pi z_i) + 10 \tag{10}$$
$$z = x - o$$
$$x \in [-5, 5]^D$$

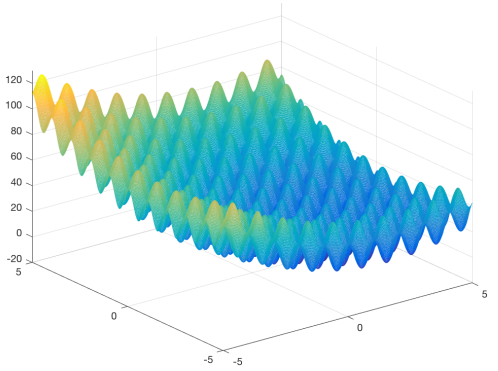

Figure 11: 3-D map for 2-D function

3.2.10 F_{10}: Shifted Rotated Rastrigin's Function

$$F_{10}(x) = \sum_{i=1}^{D} z_i^2 - 10\cos(2\pi z_i) + 10 \tag{11}$$
$$z = (x - o) * M$$
$$x \in [-5, 5]^D$$

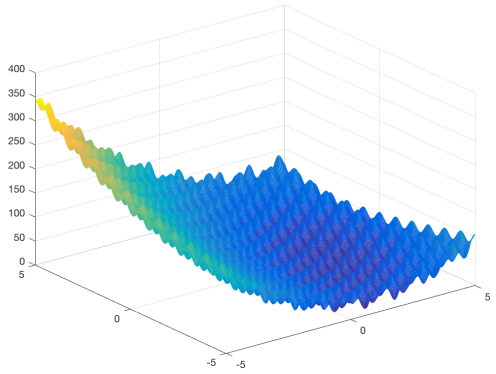

Figure 12: 3-D map for 2-D function

3.3 Machine learning problem set

To evaluate the optimization algorithms further I set have set out to apply them on a variety of machine learning problems. These will predominantly be applications of feed-forward neural networks (FFNN).

3.3.1 ML_1: Training FFNNs to do function approximation

The optimization algorithm were used to train FFNNs to perform function approximation and curve-fitting on seven sample data-sets which are available in Matlab's Neural Network Toolbox. The data-sets used are the following

1. simplefit_dataset (Simple fitting dataset)
2. abalone_dataset (Abalone shell rings dataset)
3. bodyfat_dataset (Body fat percentage dataset)
4. building_dataset (Building energy dataset)
5. chemical_dataset (Chemical sensor dataset)
6. cho_dataset (Cholesterol dataset)
7. engine_dataset (Engine behavior dataset)
8. house_dataset (House value dataset)

The data sets contain two matrices. One with sample input vectors to the neural network and one with the expected correct output vectors. The function which the optimization algorithm directly optimize is the sum of squared errors as defined by

$$sse(x,t) = \frac{1}{2n}\sum_{i=1}^{n}(y(x)-t)^2 \tag{12}$$

where x is the input vector to neural network y, t is the correct expected output which $y(x)$ should produce and n is the length of vector x.

3.3.2 ML_2: Training FFNNs to do classification

The optimization algorithm were used to train FFNNs to perform classification tasks on eight sample data-sets which are available in Matlab's Neural Network Toolbox. The data-sets used are the following

1. simpleclass_dataset(Simple pattern recognition dataset)
2. cancer_dataset (Breast cancer dataset)
3. crab_dataset (Crab gender dataset)
4. glass_dataset (Glass chemical dataset)
5. iris_dataset (Iris flower dataset)
6. thyroid_dataset (Thyroid function dataset)
7. wine_dataset (Italian wines dataset)

The evaluation procedure is the same as for ML_1.

3.3.3 ML_3: Training FFNNs to do clustering

The optimization algorithm were used to train FFNNs to perform clustering tasks on one sample data-sets which is available in Matlab's Neural Network Toolbox. The data-sets used is the following

1. simplecluster_dataset (Simple clustering dataset)

The evaluation procedure is the same as for ML_1 and ML_2.

3.3.4 ML_4: Training a FFNN to play a simple snake game

A simple "Snake" game was designed for testing the evolutionary algorithms on neural networks as applied to games. The game takes it's width and height as input parameters and constructs a square-grid which the snake will be able to move in. The snake is initialized to the center of the grid and is able to move left, right, up and down. A food square is initialized to a random square and replaced with a new random food square when it is consumed by the snake. After consuming a food square the snake gain an additional tail square. The snake dies if it runs into the edge of the grid or into itself.

The neural network controlling the snake takes a 12-dimensional representation of the snake, food and playing field and outputs a 4-dimensional vector which helps the snake decide in which direction it should move next.

Once the snake dies and the game is over a fitness score will be returned to the optimizing algorithm. The score is combination of how long the snake managed to live and how much food it ate. It is described by the equation below:

$$fitness = food^{1.5} + \frac{1}{1 - e^{-lifetime}} - 0.5 \tag{13}$$

The equation was constructed to force the snake to first learn how to survive but then not rely on moving in circles instead of looking for food.

4 Theory

This section describes the three algorithms which were benchmarked together with my own algortihm contribution.

4.1 Differential Evolution

Differential evolution is a stochastic optimization algorithm which works on populations of parameter vectors. The problem to minimize will be denoted by $f(x)$ where $X = [x_1, x_2, x_3, ...x_D]$ and D is equal to the number of variables taken as input parameters by $f(x)$. The algorithm consists of multiple steps which will be described in detail below, the flow of the algorithm is illustrated in figure 13.

Initialize population
repeat
    Cross-over
    Selection
    Create new generation
until Solution is found

Figure 13: Overview of differential evolution algorithm.

The first step in DE is to create an initial population, the size of the population is N and it will be represented by a matrix x where g is the generation and $n = 1, 2, 3, ..., N$:

$$x^g_{n,i} = [x^g_{n,1}, x^g_{n,2}, x^g_{n,3}, ..., x^g_{n,D}] \tag{14}$$

The population is randomly generated to uniformly fill the entire parameter space ($x^U_{n,i}$ is the upper bound for parameter x_i and $x^L_{n,i}$ is the lower bound for parameter x_i):

Mutation is the first step when creating a new generation from the population. Mutation is performed individually for every vector x in the population. The mutation procedure is as follows: select three random vectors for each parameter vector (this requires that the population has a size of $N > 3$) and create a set of new vectors v called mutant vectors according to the formula below where $n = 1, 2, 3, ..., N$:

$$v^{g+1}_n = [x^g_{r1n} + F(x^g_{r2n} - x^g_{r3n}) \tag{15}$$

The value of F can be chosen from the interval $[0, 2]$ and determines the influence of the differential weight $(x^g_{r2n} - x^g_{r3n})$.

Crossover occurs after mutation and is applied individually to every vector x. A new vector u called the trial vector is constructed from the mutant vector v and the original vector x. The trial vector is produces according to the formula below with $i = 1, 2, 3, ..., D$ and $n = 1, 2, 3, ..., N$:

$$u^{g+1}_{n,i} = \begin{cases} v^{g+1}_{n,i}, & \text{if } rand() \leq CR \wedge i = I_{\text{rand}} \\ x^g_{n,i}, & \text{otherwise} \end{cases} \tag{16}$$

I_{rand} is a randomly selected index from the interval $[1, D]$ and CR is the crossover constant which determines the probability that an element is selected from the mutant vector.

Selection is the last step in creating a new generation. The trial vector u is compared with the original vector x for fitness and the vector with the lost cost is selected for the generation according to the formula below where $n = 1, 2, 3, ..., N$:

$$x^{g+1}_n = \begin{cases} u^{g+1}_n, & \text{if} f(u^{g+1}_n) < f(x^g_n) \\ x^g_{n,i}, & \text{otherwise} \end{cases} \tag{17}$$

After selection is performed for every vector in the population the population is evaluated to determine if an acceptable solution has been generated. If a solution has been found the algorithm terminates, otherwise the mutation, crossover and selection is performed again until a solution is found or a maximum number of iterations has been reached [**?**].

4.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was introduced in 19XX by Kenneth and Ebenhart [reference needed]. The optimization problem is represented by an n-dimensional function

$$f(x_1, x_2, x_3, ...x_n) = f(\vec{X}) \tag{18}$$

where $\vec{X}$ is a vector which represents the real parameters given to the function. The intent is to find a point in the n-dimensional parameter hyperspace that minimizes the function.

PSO is a parallel search technique where a set of particles fly through the n-dimensional search space and probe solutions along the way. Each particle P has a current position $\vec{x}(t)$, a current velocity $\vec{v}(t)$, a personal best position $\vec{p}(t)$ and the neighborhoods best position $\vec{g}(t)$. A neighborhood N is a collection of particles, the neighborhood is often set to be identical to the whole swarm of particles, denoted S.

The algorithm has a set of general properties: v_{max} restricts the velocity of each particle to the interval $[-v_{max}, v_{max}]$, an inertial factor ω, two random numbers ϕ_1 and ϕ_2 which affect the velocity update formula by modulating the influence of $\vec{p}(t)$ and $\vec{g}(t)$, and two constants C^2 and C^1 which are termed particle "self-confidence" and "swarm confidence".

The initial values of $\vec{p}(t)$ and $\vec{g}(t)$ are equal to $\vec{x}(0)$ for each particle. After the particle have been initialized an iterative update process is started which modifies the positions and velocities of the particles. The formula below describes the process (d is the dimension of the position and velocity and i is the index of the particle):

$$v_{id}(t+1) = \omega v_{id}(t) + C_1\phi_1(p_{id}(t) - x_{id}(t)) + C_2\phi_2(g_{id}(t) - x_{id}(t)) \tag{19}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{20}$$

The "self-confidence" constant affects how much self-exploration a particle is allowed to do while "swarm-confidence" affects how much a particle follows the swarm. ϕ_1 and ϕ_2 are random numbers which push the particle in a new direction while ω keeps a particle on the path it's currently on [**?**]. The PSO algorithm is described in figure 14.

Initialize particles with random positions $\vec{x}(0)$ and velocities $\vec{v}(0)$
repeat
  for all Particles i **do**
    Evaluate fitness $f(\vec{x_i})$
    Update $\vec{p}(t)$ and $\vec{g}(t)$
    Adapt the velocity of the particle using the above-mentioned equation
    Update the position of the particle
  end for
until $\vec{g}(t)$ is a suitable solution

Figure 14: PSO algorithm

4.3 Estimation of Distribution Algorithm

Estimation of distribution algorithms are stochastic search algorithms which try to find the optimal value of a function by creating and sampling probability distributions repeatedly. The first step is creating population $P(0)$ and filling it with solution parameter vectors created from a probability distribution which covers the whole search space uniformly. Then all solutions in $P(g)$ are evaluated and the best solutions $S(g)$ are selected (a threshold variable t is used to determine how many solutions are selected, setting $t = 50\%$ means that the best of solutions are selected). After selection a probabilistic model $M(g)$ is constructed from $S(g)$ and new solutions $O(g)$ are sampled from $M(g)$. Finally $O(g)$ is incorporated into $P(g)$ The generation counter is incremented $g = g+1$ and the selection, model and sampling stages are repeated until a suitable solution is found [**?**].

The hardest part is constructing the probabilistic model, this will differ for continuous and discreet optimization and a model of appropriate complexity has to be chosen depending on the

nature of the problem. The simplest method for continuous EDAs is using a continuous Univariate Marginal Density Algorithm (UMDA). However depending on the complexity of the problem other methods such as Gaussian nets (GN) must be used [**?**].

The UMDA is an EDA algorithm which uses a set of independent probability distributions to sample new solution vectors. The probability model can be expressed as a product of the individual probabilities

$$p(x) = \prod_{d=1}^{D} p_d(x_d) \tag{21}$$

where is the global multivariate density, D is the vector length and are the individual univariate marginal densities. Equi-width histograms (EHW) can be used to capture the probability spread [**?**].

4.4 An improved evolutionary algorithm

5 Results

This section presents benchmarks and measurements of the four main algorithms presented in this thesis. It is divided into parts: the function optimization benchmark and the machine learning benchmark.

5.1 Function optimization

The function test-bed was tested with dimension size $D = 2$, $D = 10$, $D = 30$ and $D = 50$. The parameters for different dimension sizes are presented in figure 15.

Dimension	Max. Generations	Population size
2	500	100
10	1500	150
30	2000	200
50	2500	250

Figure 15: Parameters for different dimension size in benchmark

The benchmark ran each test-case 50 times and recorded the average minimum value of the function and the standard deviation. The results are presented in figure 16, 17, 18 and 19. The lowest value if marked in bold font.

F	DE		PSO		EDA		—	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Figure 16: Benchmark results for $D = 2$

F	DE		PSO		EDA		—	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Figure 17: Benchmark results for $D = 10$

F	DE		PSO		EDA		—	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Figure 18: Benchmark results for $D = 30$

F	DE		PSO		EDA		—	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Figure 19: Benchmark results for $D = 50$

5.2 Machine learning

The algorithms were tested on four machine learning benchmarks: ML_1, ML_2, ML_3 and ML_4 as described in the section "Method". The dimensionality of the problem depends on the size of the input and output vectors of the data-sets and the size of the hidden layer of the neural networks. The hidden layer size was defines as the mean of the input- and output-layer sizes. The population size was set to 50, the max. number of generations was set to 2000 and every measurement was made 50 times. The figures 20, 21, 22 and 23 show the results obtained for problems (average minimum value and standard deviation).

F	DE		PSO		EDA		—	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
simplefit_dataset	0	0	0	0	0	0	0	0
abalone_dataset	0	0	0	0	0	0	0	0
bodyfat_dataset	0	0	0	0	0	0	0	0
building_dataset	0	0	0	0	0	0	0	0
chemical_dataset	0	0	0	0	0	0	0	0
cho_dataset	0	0	0	0	0	0	0	0
engine_dataset	0	0	0	0	0	0	0	0
house_dataset	0	0	0	0	0	0	0	0

Figure 20: Benchmark results for ML_1

F	DE		PSO		EDA		QSA	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
simpleclass_dataset	0	0	0	0	0	0	0	0
cancer_dataset	0	0	0	0	0	0	0	0
crab_dataset	0	0	0	0	0	0	0	0
glass_dataset	0	0	0	0	0	0	0	0
iris_dataset	0	0	0	0	0	0	0	0
thyroid_dataset	0	0	0	0	0	0	0	0
wine_dataset	0	0	0	0	0	0	0	0

Figure 21: Benchmark results for ML_2

F	DE		PSO		EDA		QSA	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
simplecluster_dataset	0	0	0	0	0	0	0	0

Figure 22: Benchmark results for ML_3

F	DE		PSO		EDA		QSA	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
snake_dataset	0	0	0	0	0	0	0	0

Figure 23: Benchmark results for ML_4

6 Discussions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed suscipit eu ante id aliquet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris condimentum iaculis erat, eget pretium leo consequat eget. Donec eu aliquam purus. Quisque scelerisque feugiat ligula, non vehicula urna blandit nec. Etiam a dignissim purus, at molestie dui. Vestibulum a venenatis mi, id hendrerit sem. Cras purus elit, dapibus eget est id, lacinia tincidunt neque. Mauris eu arcu ipsum. Sed enim leo, iaculis vitae nisl tincidunt, ultricies accumsan diam. Donec neque urna, pellentesque sit amet interdum a, pretium ac enim. Proin quis ante nunc. Fusce vitae lorem condimentum, porta velit blandit, commodo nisi.

7 Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed suscipit eu ante id aliquet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris condimentum iaculis erat, eget pretium leo consequat eget. Donec eu aliquam purus. Quisque scelerisque feugiat ligula, non vehicula urna blandit nec. Etiam a dignissim purus, at molestie dui. Vestibulum a venenatis mi, id hendrerit sem. Cras purus elit, dapibus eget est id, lacinia tincidunt neque. Mauris eu arcu ipsum. Sed enim leo, iaculis vitae nisl tincidunt, ultricies accumsan diam. Donec neque urna, pellentesque sit amet interdum a, pretium ac enim. Proin quis ante nunc. Fusce vitae lorem condimentum, porta velit blandit, commodo nisi.

8 Acknowledgments

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed suscipit eu ante id aliquet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris condimentum iaculis erat, eget pretium leo consequat eget. Donec eu aliquam purus. Quisque scelerisque feugiat ligula, non vehicula urna blandit nec. Etiam a dignissim purus, at molestie dui. Vestibulum a venenatis mi, id hendrerit sem. Cras purus elit, dapibus eget est id, lacinia tincidunt neque. Mauris eu arcu ipsum. Sed enim leo, iaculis vitae nisl tincidunt, ultricies accumsan diam. Donec neque urna, pellentesque sit amet interdum a, pretium ac enim. Proin quis ante nunc. Fusce vitae lorem condimentum, porta velit blandit, commodo nisi.

References

[1] A. E. Eiben and J. E. Smith, *Evolutionary Computing: The Origins.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 13–24.

[2] Z. Michalewicz, R. Hinterding, and M. Michalewicz, *Evolutionary Algorithms.* Boston, MA: Springer US, 1997, pp. 3–31.

[3] Y. Zhang, S. Wang, and G. Ji, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[4] A. P. Engelbrecht, *Computational intelligence: an introduction.* John Wiley & Sons, 2007.

[5] A. Eiben and M. Schoenauer, "Evolutionary computing," *Information Processing Letters*, vol. 82, no. 1, pp. 1 – 6, 2002, evolutionary Computation.

[6] A. E. Eiben and J. E. Smith, *What Is an Evolutionary Algorithm?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 25–48.

[7] M. Mitchell and C. E. Taylor, "Evolutionary computation: An overview," *Annual Review of Ecology and Systematics*, vol. 30, pp. 593–616, 1999.

[8] D. B. Fogel and L. J. Fogel, *An introduction to evolutionary programming.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 21–33.

[9] K. D. Jong, "Evolutionary computation," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 52–56, 2009.

[10] R. Bansal, "A bibliographical survey of evolutionary computation applications in power systems (1994-2003)," *International journal of power & energy systems*, vol. 26, no. 3, p. 216, 2006.

[11] M. Guzek, P. Bouvry, and E.-G. Talbi, "A survey of evolutionary computation for resource management of processing in cloud computing [review article]," *IEEE Computational Intelligence Magazine*, vol. 10, no. 2, pp. 53–67, 2015.

[12] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 321–344, 2013.

[13] R. Aguilar-Rivera, M. Valenzuela-Rendón, and J. Rodríguez-Ortiz, "Genetic algorithms and darwinian approaches in financial applications: A survey," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7684–7697, 2015.

[14] N. Krasnogor, W. E. Hart, J. Smith, and D. A. Pelta, "Protein structure prediction with evolutionary algorithms," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, ser. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1596–1601.

[15] Z. Xiu-fen, P. Zi-shu, K. Li-shan, and Z. Chu-yu, "The evolutionary computation techniques for protein structure prediction: A survey," *Wuhan University Journal of Natural Sciences*, vol. 8, no. 1, pp. 297–302, 2003.

[16] K. Safarzyńska and J. C. van den Bergh, "Evolutionary models in economics: a survey of methods and building blocks," *Journal of Evolutionary Economics*, vol. 20, no. 3, pp. 329–373, 2010.

[17] Y.-J. Zheng, S.-Y. Chen, and H.-F. Ling, "Evolutionary optimization for disaster relief operations: a survey," *Applied Soft Computing*, vol. 27, pp. 553–566, 2015.

[18] S. Damas, O. Cordón, and J. Santamaría, "Medical image registration using evolutionary computation: An experimental survey," *IEEE Computational Intelligence Magazine*, vol. 6, no. 4, pp. 26–42, 2011.

[19] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[20] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," *KanGAL report*, vol. 2005005, p. 2005, 2005.

[21] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artificial Intelligence*, vol. 85, no. 1â2, pp. 245 – 276, 1996.

[22] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[23] S. Das, A. Abraham, and A. Konar, *Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–38.

[24] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111 – 128, 2011.

[25] P. Pošık, "Estimation of distribution algorithms," *Czech Technical University, Faculty of Electrical Engineering, Dept. of Cybernetics*, 2004.