Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Bachelor Thesis

EVOLUTIONARY COMPUTATION IN CONTINUOUS OPTIMIZATION AND MACHINE LEARNING

Leslie Dahlberg
ldg14001@student.mdh.se

Examiner: Peter Funk

Mälardalen University
Västerås, Sweden

Supervisor: Ning Xiong

Mälardalen University
Västerås, Sweden

April 7, 2017

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 can be formulated in such a way that we seek to minimize the output value of the model or the output of some function which transforms the models output into a fitness score. Because of this the process is often referred to as minimization. It becomes obvious that this is useful when considering optimizing the layout of a circuit in order to minimize the power consumption. To achieve this the optimizer looks for combinations of parameters which let the model produce the best output to a given input [1].

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. Additionally, the analytical model is not always known, which makes it impossible to use such methods. 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 [2, 3].

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 [4].

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 and 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 the aforementioned ones in some aspect.

Research has been conducted on improving various evolutionary algorithms by hybridizing and extending them, which has resulted in a wide array of algorithms for both specific and general purposes. Since these algorithms accepts parameters which modify their efficiency, studies have been carried out which compare different combinations of parameters. My thesis will draw upon this work by comparing three algorithms both generally and on a very specific problem.

2 Background

This section will aim to provide a general overview of the field of evolutionary computation. General terms and procedures which are often utilized in EC will be explained and the most well known traditional approaches will be presented. The necessary concepts from machine learning will be presented.

2.1 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 initialized by some procedure to contain a random set of parameter vectors, these should cover the whole parameter range of the model uniformly. The initial 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 characteristics are selected to live on into the next generation of the population. They are then altered randomly to create diversity in the population and evaluated. This process creates a new generation of the population on each iteration and continues until a solution is found or some other restriction is encountered [5]. The concept is demonstrated in figure 1 with $P(t)$ representing the population at generation t.

Algorithm 1 Basic evolutionary algorithm

```
t ← 0
initialize P(t)
evaluate P(t)
while termination-condition not fulfilled do
    t ← t + 1
    select P(t) from P(t − 1)
    alter P(t)
    evaluate P(t)
end while
```

Here the fundamental building 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.

2.1.1 Representation

The first step in using evolutionary algorithms is creating a representation which can encode all possible solutions to the problem. 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 interchangeably 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 [6].

Binary representation Genetic algorithms (GA) traditionally use a binary representation to store individual genotypes. The representations is a string of fixed length over the alphabet $\{0, 1\}$. The problem thus becomes a boolean optimization problem of the form $f : \{0, 1\}^l \to \mathbb{R}$, where the mappings $h : M \to \{0, 1\}^l$ and $h' : \{0, 1\}^l \to M$ are used to encode and decode the parameters and solutions [7].

Integer representation Integer representations have been proposed by some researches [8]. This approach is useful when dealing with problems where we need to select certain elements in a particular order, e.g. graph-problems, path-finding problems, the knapsack problem, etc. [9].

Real-valued representation Real-valued or floating-point representations were originally used in evolutionary programming and evolution strategies and work well for problems located in continuous search-spaces. The problems take the form $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ [7].

Tree representation Tree representations are mainly used in genetic programming to capture the structure of programs. The encoding varies but S-expressions are generally used. The tree structure is defined by a function-set and a terminal-set. The function-set defines the types of nodes in the tree, while the terminal-set contains the types of leaves the tree can contain [9].

2.1.2 Evaluation function

Evaluation function is responsible for improvement in the population. It is a function which assigns a fitness or cost value to every genotype and thus enables us to compare the quality of the genotypes in the population. It is also the only information about the problem that is available to the evolutionary algorithm and should therefore include all domain knowledge which is available about the problem [6]. The evaluation is also the process which takes up the most computational resources, 99% of the total computational cost in real-world problems [5].

2.1.3 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 allowed into the next generation of the population. The size of the population almost always remains constant during the lifetime of the algorithm. This in turn creates selection pressure which pushes the population towards improvement. A population's diversity is the measure of difference among the genotypes, phenotypes and fitness values [6].

Steady-state model In the steady-state model the entire population isn't replaced at once. Only a fraction of the population is replaced at one time.

Generational model In the generational model the entire population is replaced at once.

2.1.4 Parent selection mechanism

Parent selection serves to improve the quality of a population by selecting which individuals will survive into the next generation. The selected individuals are called parents as they usually undergo some form of alteration or combination with other individuals before progressing to the next generation. The selection method is usually probabilistic and gives better solutions a higher probability and worse solutions a lower probability to survive. It's important that bad solutions still receive a positive probability since the population might otherwise lose diversity and coalesce around a false local optimum [6].

Fitness proportional selection Fitness proportional selection (FPS) assigns a selection probability to an individual based on it's absolute fitness. This results in very good individuals overtaking the population quickly and premature convergence. If individuals have very similar fitness values the selection pressure becomes low. This mechanism is also very dependent on the exact form of the fitness function [10].

Ranking selection In ranking selection the population is sorted according to the individual's fitness values and the selection pressure is kept constant. A constant number of the best individuals is then selected from the sorted list [10].

Tournament selection Tournament selection enables selection without global knowledge of the population's fitness. A number of individuals are chosen at random and the best one is selected. This process is repeated until the desired number of individuals is selected [10].

2.1.5 Variation operators

Variation operators 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 are called mutation operators while n-ary operators are called cross-over or recombination operators. The biological equivalents to these 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 [6, 5].

Binary mutation The most commonly used mutation scheme for binary representations consists of randomly flipping bits (genes) in a chromosome with a certain probability. The number of alterations is not fixed with this approach, but the common procedures used can often be set to change a certain number of bits on average [9].

Binary recombination Three approached are normally used to recombine binary chromosomes. One-point crossover divides the chromosome into two sections, picking a random intersection point, and swaps the tails of the two chromosomes creating two offspring. N-point crossover generalizes this behavior by picking n random splitting points and assembling new chromosome by taking alternate sections of the parent chromosomes. Uniform crossover creates an array of uniform numbers from a probability distribution and chooses which parent to take a gene from by comparing the random value to a probability threshold [9].

Integer mutation Random resetting and creep mutation are used to mutate integer chromosomes. In random resetting integer values are changed with a certain probability. The new values are chosen at random from the pool of permissible values. Creep mutation samples small numbers from distributions and adds or subtracts them from genes at random [9].

Integer recombination For integer representations the same techniques that are used for binary representations apply [9].

Real-valued mutation For real-valued representations mutation is similar to integer mutation with the exception that new random values are drawn from continuous distributions and creep mutation uses a gaussian distribution to sample values. A lower and upper bound is used to limit the span of the generated random numbers [9].

Real-valued recombination There are three common ways to recombine real-valued chromosomes. Discreet recombination works like n-point crossover and thus does not alter the values in the offspring chromosomes. Arithmetic recombination chooses values which fall in-between the values of the parent chromosomes for it's offspring. Blend recombination works like arithmetic recombination but allow for values which lie slightly outside of the interval defines by the parent genes [9].

Tree mutation Trees are usually mutated by selecting a node at random and re-generating it's subtree using the same approach which was used to generate the initial population [9].

Tree recombination Subtree crossover is commonly used to recombine trees. It randomly selects a node in each tree and then swaps the respective subtrees creating two new offspring [9].

2.1.6 Survivor selection mechanism

Survivor selection takes place after new offspring have been generated and determines which individuals are allowed to live on into the next generation. It is often referred 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 [6].

2.1.7 Initatilization

Initialization is the process during which the initial population is generated. 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 [5].

2.1.8 Termination condition

The termination condidion determines for how long the algorithm is run. Four criteria are used to determine when to stop [6]:

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 improve for a considerable amount of time
4. If the diversity of the population drops below a given threshold

2.2 Traditional Evolutionary Algorithms

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

2.2.1 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 multidimensional 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 [11]. See algorithm 2 and figure 1 for a simple GA.

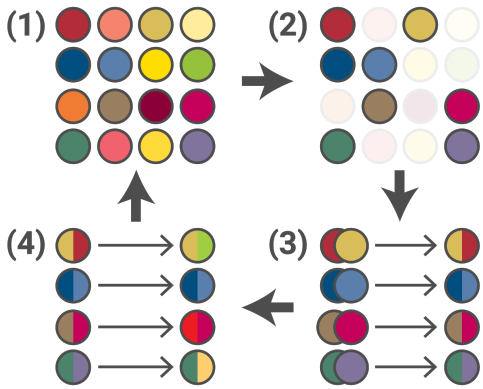

Figure 1: Stages in GA (1) Evaluation (2) Selection (3) Crossover (4) Mutation

Algorithm 2 Basic genetic algorithm

```
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 probabilistically 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
```

2.2.2 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 they performed better than their parents. The vector $\vec{x}$ represents the position in the search space and $\vec{\sigma}$ represents a vector of 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 $\vec{\sigma}$ [2].

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

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].

2.2.3 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 [12]. The environment is modeled as a sequence of input characters selected 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 a function which tests the FSM on a sequence of input characters, starting with the first character and then progressing to include one more addition character on each iteration. The function measures the correct prediction rate of the FSM and determines it's score [2].

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].

2.2.4 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 [2].

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 blocks such as variables, numbers and operators. The initial population contains a set of such programs which have been initialized as random trees [2].

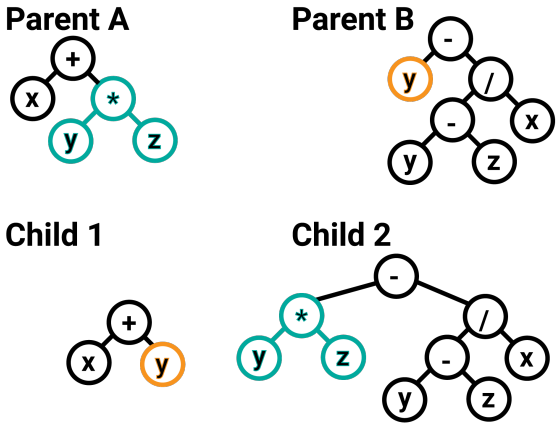

Figure 2: Crossover in GP with two parents producing two offspring

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 mutations. Cross-over is defined as the exchange of subtrees between programs and produces two offspring from two parents [2], see figure 2.

More advanced versions of GP include function calls which enable the programs to remember useful symmetries and regularities and facilitate code reuse [2].

2.3 Emerging Evolutionary Algorithms

This section describes the three algorithms which were benchmarked together with my own algorithm contribution. The Matlab code can be viewed in appendix **??**.

2.3.1 Differential Evolution

Differential evolution [13] 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. See algorithm 3

Algorithm 3 DE algorithm

```
Initialize population within bounds
Evaluate fitness of population
repeat
    for all Individuals x do
        Select three other random individuals a, b and c
        Create mutant vector v = a + F(b − c)
        Create trial vector u by blending x with v (taking at least one gene from u)
        Replace x with u if fitness(u) > fitness(x)
    end for
until End condition
```

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 $n = 1, 2, 3, ..., N$:

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

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 $n = 1, 2, 3, ..., N$:

$$v^{g+1}_n = [x^g_{r1n} + F(x^g_{r2n} - x^g_{r3n}) \tag{3}$$

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 produced according to the formula below with $i = 1, 2, 3, ..., D$ and $n = 1, 2, 3, ..., N$:

$$u^{g+1}_{n,i} = \begin{cases} v^{g+1}_{n,i}, & \text{if } rand() \leq CR \wedge i = I_{\text{rand}} \\ x^g_{n,i}, & \text{otherwise} \end{cases} \tag{4}$$

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 $n = 1, 2, 3, ..., N$:

$$x^{g+1}_n = \begin{cases} u^{g+1}_n, & \text{if} f(u^{g+1}_n) < f(x^g_n) \\ x^g_{n,i}, & \text{otherwise} \end{cases} \tag{5}$$

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.

In the benchmark the parameters for DE have been set to $F = 0.6$ and $CR = 0.9$ as recommended by Gamperle et al. [14].

Variants Different DE schemes are classified as DE/x/y, where x symbolizes the vector which is mutated and y is the number of differential vectors used. The value of x can be "rand" for random vector or "best"' for the best vector in the population. The algorithm abose is therefore classified as DE/rand/1. The variant DE/best/2 is considered to be a good alternative to DE/rand/1 [15]. It's mutation equation is described below:

$$v^{g+1}_n = [x^g_{best} + F(x^g_{r1n} - x^g_{r2n}) + F(x^g_{r3n} - x^g_{r4n}) \tag{6}$$

2.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [16] was introduced in 1995 by Kenneth and Ebenhart [17]. The optimization problem is represented by an n-dimensional function

$$f(x_1, x_2, x_3, ...x_n) = f(\vec{X}) \tag{7}$$

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 which act as independent swarms. Neighbourhoods can be social or geographical. Social neighbourhoods do not change and contain the same particles during

the entire optimization process, while geographical neighborhoods are dynamic and consist only of particles which are near to one another. The neighborhood is often set to be identical to the whole swarm of particles, denoted S.

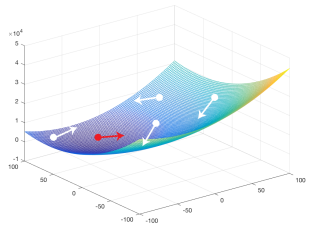

Figure 3: Illustration of particles in PSO

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):

$$v_{id}(t+1) = \omega v_{id}(t) + C_1\phi_1(p_{id}(t) - x_{id}(t)) + C_2\phi_2(g_{id}(t) - x_{id}(t)) \tag{8}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{9}$$

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 algorithm 4. See figure 3 for an illustration of PSO.

Algorithm 4 PSO algorithm

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

In the benchmark the parameters for PSO have been set to $\omega = 0.8$ [18], $c_1 = c_2 = 1.494$ [19] and v_{max} = parameter range size [16].

2.3.3 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 50% of the 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 [20].

The most difficult 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 Estimation of Baysian Networks (EBNA) can be used [21].

UMDA The UMDA algorithm 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

$$p(x) = \prod_{d=1}^{D} p_d(x_d) \tag{10}$$

where $p(x)$ is the global multivariate density, D is the vector length and $p_d(x_d)$ are the individual univariate marginal densities [22]. The algorithm is described in algorithm 5.

Algorithm 5 UMDA algorithm

```
Initialize population P
repeat
    Evaluate P
    Select the best t% of P into S
    Let m be the mean of S
    Let s be the standard deviation of S
    Sample S' from normal distribution using m and s
    Create new generation of P from S' and S
until Termination condition
```

2.4 Machine learning concepts

The machine learning concepts needed for this thesis are artificial neural networks (ANN), specifically feed-forward neural networks (FFNN). Artificial neural networks are mathematical models which are based on the functioning of biological neurons in the brain. They are useful for predicting future behavior and events, trends, function approximation and data-classification. FFNNs are a popular type of ANN, often referred to as multi-layer perceptrons. ANNs have to be trained on training data in order to function properly and the most widely used and most popular method for this is back-propagation (BP) [23].

Back-propagation is a local minimization algorithm which works in n-dimensions. It can therefore be replaced by other optimization algorithms such as genetic algorithms. This is what will be considered in the thesis [23].

2.4.1 Feed-Forward Neural Networks

ANNs usually consist of five components

1. A graph containing nodes (neurons) and links between nodes
2. A variable which holds the state of each node
3. A real-valued weight for each link between nodes
4. A real-valued threshold for each node

5. A transfer function which calculates the value of a node's state variable based on the states of nodes which precede the node and are connected to it through links

FFNNs consists of one input layer of neurons which accept a vector of input signals, one or more "hidden" layers of intermediary neurons which process the signal and one output layer which produces the final result in the form of an output vector. In FFNNs all neurons are interconnected with all other neurons in neighbouring layers. These connections have weights which modulate the strenghs of the respective connections [24]. See figur 4 for a simple FFNN with two neurons (z_1 and z_2) in the input layer, which are linked to three hidden neurons (y_1, y_2 and y_3), which are in turn linked to two output neurons (o_1 and o_2). The weights are represented by v_{ij} and w_{ij}.

Evaluating the neural network The value of a neuron x_i in the network is determined according to the equations

$$x_i = f(\xi_i) \tag{11}$$

$$\xi_i = v_i + \sum_{j=0}^{n} \omega_{ij} x_j \tag{12}$$

where n is the number of neurons in the preceding layer, x_j are the neurons in the preceding layer and v_i are the weights between x_i and x_j.

The function f is the transfer function and is defined as

$$f(\xi) = \frac{1}{1 + exp(-\xi)} \tag{13}$$

The threshold is added as an extra neuron to each layer except the output layer to simplify the calculations. The threshold neuron always has a value of -1. The actual threshold value is determined by the weights of the link going out of the threshold neuron. [25].

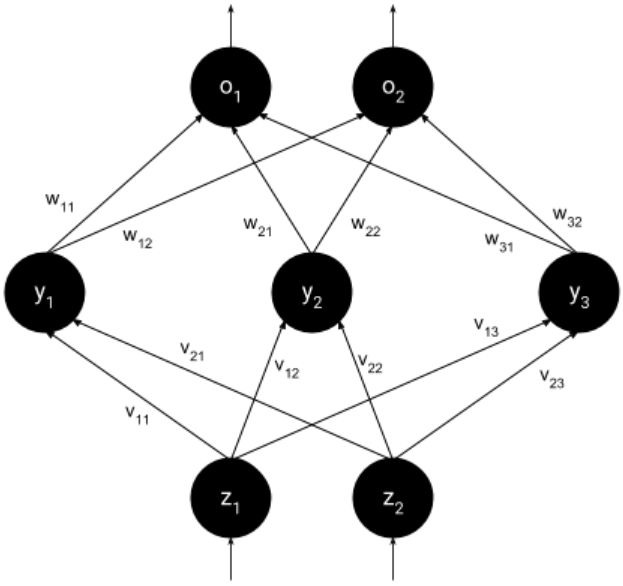

Figure 4: Feed-Forward Neural Network

Back-propagation Back-propagation (BP) is the most popular method for training neuron networks. Training data is send through the network and the output vector is subtracted from a result vector which contain the correct output values producing the error difference. An error value is then calculated by the means of the sum of squared errors from the individual error differences. Subsequently all weights in the network are slightly adjusted to accommodate the calculated error. After a number of iterations this process produces the correct combination of weights [4].

2.5 Method

A benchmark was constructed to perform the algorithm comparison. Both mathematical function optimization and machine learning test-sets were used. These are described in detail in the section **??**. Matlab was used to implement the algorithms and benchmarks because it provides easy access to important mathematical and scientific constructs which are usually not available in conventional programming languages. Matlab is also widely used in the field of evolutionary computation.

3 Related Work

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 Holland's 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 [26].

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 in which situations, a problem which is not trivial [26].

3.1 Recent Research

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-population 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 [26].

3.2 Emerging Evolutionary Algorithms

Differential Evolution Differential evolution (DE) was conceived in 1995 by Storn and Price [27] and soon gained wide acceptance as one of the best algorithms in continuous optimization [28]. This spawned many new papers describing variations and hybrids of the algorithm [29], such as self-adaptive DE (SaDE) [15], opposition-based DE (ODE) [30] and DE with global and local neighborhoods (DEGL) [30].

DE is very easy to implement and has been shown to outperform most other algorithms consistently, it has also been shown that it in general performs better than PSO [31, 32]. DE uses very few parameters and the effects of altering these parameters have been well studied [29]. DE comes in a total of 10 different varieties based on which mutation and cross-over operators are used [33]. Eight of these schemes have been tested and compared, showing that the version called DE/best/1/bin (which utilizes the best current individual in the cross-over process instead of random individuals) generally yields the best results [34]. [14] measured the performance of different combinations of parameters, producing general recommendations for DE.

The desire to find optimal parameters have led to the use of self-adjusting DE algorithms. Examples include the use of fuzzy systems to control the parameters [35] and the SaDE algorithm [15].

Particle Swarm Optimization Particle swarm optimization (PSO) is the most widely used swarm intelligence (SI) algorithm to date. Many modified versions of PSO have been proposed, among others quantum-behaved PSO (QPSO), bare-bones PSO (BBPSO), chaotic PSO, fuzzy PSO, PSOT-VAC and opposition-based PSO. PSO has also been hybridized with other evolutionary algorithm, for instance: genetic algorithms (GA), artificial immune systems (AIS), tabu

search (TS), ant colony optimization (ACO), simulated annealing (SA), differential evolution (DE), bio-geography based optimization (BBO) and harmonic search (HS) [3].

Wang et al. [36] have compared the performance of different PSO-parameters and proposed a set of criteria for improving the performance of PSO. Fuzzy logic controllers (FLC) have been used to continuously optimize the PSO-parameters by Kumar and Chaturvedi [37]. Zhang et al. found a simple way to use control theory in order to find good parameters [38]. Yang proposed modified velocity PSO (MVPSO) in which particles learn the best parameters from the other particles [39].

Estimation of Distribution Algorithms Estimation of distribution algorithms (EDA) use probabilistic models to solve complex optimization problems. They have been successful at many engineering problems which at which other algorithms have failed, for instance: military antenna design, protein structure prediction, clustering of genes, chemotherapy optimization, portfolio management, etc [20].

Several techniques have been proposed to make EDA more efficient. Parallelization of fitness evaluation and model building has proven effective [40]. Local optimization techniques such as deterministic hill climbing (DHC) has been shown to make EDA faster [41].

3.3 Evolutionary Algorithms and Machine Learning

Evolutionary algorithms (EC) and machine learning (ML) are two growing and promising field in computer science and many attempts have therefore been made to combine the two. ML has been used to improve EC optimization algorithms with so called MLEC-algorithms where various techniques from AI and ML, such as artificial neural networks (ANN), cluster analysis (CA), support vector machines (SVM), etc. help EC algorithms to learn important features of the search space [42].

The opposite use case has also been proposed, using EC to improve ML techniques. An example of this is using DE to optimize feed-forward neural networks (FFNN). Here DE seems to perform similarly to traditional gradient based techniques [43]. Hajare and Bawane showed that using PSO to initialize weights and biases in a neural network before training yields better results than using random weights and help avoid local minima which back-propagation (BP) algorithms often get stuck in [44]. Larrañaga and Lozano [45] tested various EC algorithms (GA, EDA and ES) agains BP and concluded that EC is a competitive alternative to traditional approaches.

3.4 Contributions

EC is a an interesting alternative to traditional approaches in machine learning and continuous optimization and while algorithms such as DE, PSO, etc. have been compared on mathematical benchmarks before [32, 28], the application of EC to machine learning has not been studied with as much detail. My primary contribution to the field will be to find what algorithms work best for different ML problems and based on this propose ML-specific improvements.

4 Problem Formulation

The purpose of this thesis is to explore the performance and usefulness of three emerging evolutionary algorithms: differential evolution (DE), particle swarm optimization (PSO) and estimation of distribution algorithms (EDA). The intention is to 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.

Research Questions The questions asked in the thesis are:

- How do DE, PSO and EDA perform comparatively when applied to mathematical optimization problems?
- How do DE, PSO and EDA perform comparatively when applied to machine learning problems such as neural network optimization and artificial intelligence in games
- How are DE, PSO and EDA suited to these different problems?
- Can an improved algorithm which draws inspiration from the design of DE, PSO and/or EDA outperform any of them in some/all of the aforementioned benchmarks and problem sets?

Motivation This research is interesting because it yield insight into the applicability of emerging evolutionary algorithms to currently popular machine learning methods such as artificial neural networks and also compares them more generally on generic mathematical optimization problems. The possibility of an improved novel algorithm which is better at handling machine learning problems also makes the work more interesting.

Outcomes The goals in this works are:

- Benchmark DE, PSO, EDA on mathematical optimization problems
- Benchmark DE, PSO, EDA on machine learning problems
- Develop a new algorithm inspired by DE, PSO and/or EDA
- Benchmark the new algorithm
- Compare the new algorithm with DE, PSO and EDA and draw conclusions from the results

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.

5 Time plan

The thesis work planing will be based upon the structure of the written report to ensure that the final product is delivered in time. The first two weeks will serve as an introduction to the topic and the introduction and background section of the report will be completed during this period. This will guarantee that they will be available in time for the status and planning seminary. After this, deeper inquiry into the main algorithms of the thesis will be conducted and the first implementation prototypes will be constructed. During this time a benchmark methodology will also be decided. This phase might occupy 1 - 3 weeks and will be followed by a benchmark and an investigation of how to apply the algorithms to machine learning problems. This will probably occupy one week and will be followed by the development of an improved algorithm and the implementation of the machine learning benchmarks. After this a complete benchmark of all four algorithms on both standard functions and machine learning problems will be conducted. This should leave enough time over to fine tune the new algorithm and finalize the report before the end of the course/project period.

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] T. Back, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, 1997.

[8] G. unter Rudolph, "Evolutionary algorithms for integer programming," *Davidor et al*, vol. 611, pp. 139–148.

[9] A. E. Eiben and J. E. Smith, *Representation, Mutation, and Recombination.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 49–78.

[10] ——, *Fitness, Selection, and Population Management.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 79–98.

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

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

[13] 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.

[14] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Advances in intelligent systems, fuzzy systems, evolutionary computation*, vol. 10, pp. 293–298, 2002.

[15] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[16] 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.

[17] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on.* IEEE, 1995, pp. 39–43.

[18] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on.* IEEE, 1998, pp. 69–73.

[19] J. Kennedy, "Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999, pp. 1931–1938.

[20] 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.

[21] P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana, "A review on probabilistic graphical models in evolutionary computation," *Journal of Heuristics*, vol. 18, no. 5, pp. 795–819, 2012.

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

[23] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14, pp. 2627–2636, 1998.

[24] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms." in *IJCAI*, vol. 89, 1989, pp. 762–767.

[25] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

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

[27] R. Storn and K. Price, "Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces, berkeley," *CA: International Computer Science Institute*, 1995.

[28] K. Price, "Differential evolution vs. the functions of the 2nd icec," in *Proceeding of 1997 IEEE International Conference on Evolutionary Computation*, 1997.

[29] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, Feb 2011.

[30] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary computation*, vol. 12, no. 1, pp. 64–79, 2008.

[31] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighborhood-based mutation operator," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 526–553, 2009.

[32] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2. IEEE, 2004, pp. 1980–1987.

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

[34] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 485–492.

[35] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing*, vol. 9, no. 6, pp. 448–462, 2005.

[36] W. Dang-she, W. Hai-jiao, and Z. Jian-ke, "Selection of the pso parameters for inverting of ellipsometry," in *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on*. IEEE, 2012, pp. 776–780.

[37] S. Kumar and D. Chaturvedi, "Tuning of particle swarm optimization parameter using fuzzy logic," in *Communication systems and network technologies (CSNT), 2011 international conference on*. IEEE, 2011, pp. 174–179.

[38] W. Zhang, Y. Jin, X. Li, and X. Zhang, "A simple way for parameter selection of standard particle swarm optimization," in *International Conference on Artificial Intelligence and Computational Intelligence*. Springer, 2011, pp. 436–443.

[39] H. Yang, "Particle swarm optimization with modified velocity strategy," *Energy Procedia*, no. 11, pp. 1074–1079, 2011.

[40] K. Sastry, D. E. Goldberg, and X. Llora, "Towards billion-bit optimization via a parallel estimation of distribution algorithm," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 577–584.

[41] W. E. Hart, "Adaptive global optimization with local search," Ph.D. dissertation, Citeseer, 1994.

[42] J. Zhang, Z. h. Zhan, Y. Lin, N. Chen, Y. j. Gong, J. h. Zhong, H. S. H. Chung, Y. Li, and Y. h. Shi, "Evolutionary computation meets machine learning: A survey," *IEEE Computational Intelligence Magazine*, vol. 6, no. 4, pp. 68–75, Nov 2011.

[43] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Processing Letters*, vol. 17, no. 1, pp. 93–105, 2003.

[44] P. R. Hajare and N. G. Bawane, "Feed forward neural network optimization by particle swarm intelligence," in *Emerging Trends in Engineering and Technology (ICETET), 2015 7th International Conference on*. IEEE, 2015, pp. 40–45.

[45] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001, vol. 2.