CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Differential Evolution Crossover with Dependency Detection

Vojtěch Voráček

March 2021

Acknowledgement / Declaration

Chtěl bych poděkovat své manželce Ludmile za podporu nejen finanční. Díky tomu mohu na svém pracovišti dělat, co mě baví, a nejsem stresován výplatní páskou.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 13. 13. 2013

....

Abstrakt / Abstract

Abstrakt stručně a přesně reprezentuje obsah práce, shrnuje cíl, metody, výsledky a závěry.

Klíčová slova: Klíčová slova jsou odborné termíny vyjadřující obsah práce.

Překlad titulu: Křížení pro diferenciální evoluci s detekcí závislostí

This document shows and tests an usage of the plainTEX officially (may be) recommended design style CTUstyle for bachelor (Bsc.), master (Ing.), or doctoral (Ph.D.) theses at the Czech Technical University in Prague. The template defines all thesis mandatory structural elements and typesets their content to fulfil the university formal rules.

This is version 2 of this template which implements the Technika font recommended by CTU graphics identity reference since 2016.

Keywords: document design template; bachelor, master, Ph.D. thesis; TEX.

/ Contents

1 Introduction 1
1.1 Motivation 1
2 Evolutionary algorithms 2
2.1 Components of evolutionary algorithms 2
2.1.1 Representation of individuals 2
2.1.2 Objective function 3
2.1.3 Population 3
2.1.4 Parent selection 3
2.1.5 Crossover operator 4
2.1.6 Mutation operator 4
2.1.7 Replacement strategy 4
2.1.8 Initialization 4
2.1.9 Termination condition 5
2.2 General scheme 5
2.3 Differential evolution 5
2.3.1 Representation 6
2.3.2 Mutation 6
2.3.3 Crossover 6
2.3.4 Replacement strategy 7
3 Linkage information modeling 9
3.1 Family Of Subsets 9
3.2 Linkage tree 9
3.3 Marginal product 11
4 Identification of the linkage structure 12
4.1 Fitness-based method 12
References 14

Chapter 1
Introduction

1.1 Motivation

Striving for the best solution of a certain problem is an important part of many fields of human interest. The process of finding the best solution according to some criteria is called optimization. Optimization in mathematical notation:

In the real world, there are a large number of engineering optimization problems whose input-output relationships are noisy and indistinct, so it cannot be assumed anything about the optimized function but it's possible to observe its outputs on given inputs. In these cases, the function is called a black box function and an optimization as a black box optimization.

Due to these limited capabilities, all black box optimization algorithms are allowed to perform just these three steps:

- Create a candidate solution
- Check if a candidate is feasible or not
- Evaluate its fitness by using the objective function

From the mid-1950s, a new family of optimization algorithms called Evolutionary algorithms has started to be developed. [1–2] Evolutionary algorithms have proven to be very effective while optimizing black box functions. [3] Among evolutionary algorithms, *Differential Evolution* (DE) has achieved excellent results on real-valued black box functions. [4].

However, there exists a class of functions containing dependent solution components and the recognition of those components may be a crucial task which could lead to significantly enhanced performance. Nevertheless, DE does not provide any tool capable of recognizing the dependent components of a solution. Thus it can be seen that this particular class of functions is the weakness of DE.

This work aims to find a way how to find dependencies between parts of solutions and how to represent a dependency structure. It would lead to the proposal of a new crossover operator for DE well suited for functions with dependent solution components. This new operator should partially eliminate the above-mentioned weakness of DE.

Chapter 2
Evolutionary algorithms

This chapter is mainly based on these references: [5–7]

Evolutionary algorithms (EAs) is a set of stochastic metaheuristic optimization algorithms inspired by Darvin's theory of evolution by natural selection. [8] The theory describes the process of development of organisms over time as a result of changes in heritable traits. Changes which allow an organism to better adapt to its environment will help it survive and reproduce more offspring. This phenomenon is commonly called as "Survival of the fittest" first used by Herbert Spencer. [9]

In analogy, EA maintains a "population" of potential solutions (*individuals*) for the given problem. Population is iteratively evolved by encouraging the reproduction of fitter individuals. The fitness is usually the value of the objective function in the optimization problem being solved. New candidate solutions are created either by combining existing individuals (crossover) or by modification of an individual (mutation). The algorithm runs until a candidate solution with sufficient quality is found or a user-defined computational limit is reached.

2.1 Components of evolutionary algorithms

In this section, certain parts of evolutionary algorithms are discussed in detail. In general, EAs can be divided into various components, procedures, or operators, which are:

- representation of individuals
- objective function
- population
- parent selection
- crossover operator
- mutation operator
- replacement strategy

To define a particular EA, it is necessary to specify these components. In addition, the initialization procedure and the termination condition must be defined to obtain working algorithm.

2.1.1 Representation of individuals

Each individual is encoded in so called *chromosomes*. Representation of chromosome is called *genotype* . While *phenotype* refers to the interpretation of the genotype, in other words, how the objective function treats the genotype. The Representation also involves a genotype-phenotype mapping. For instance, given an optimization problem on integers, if one decide to represent them by their binary code, then 20 would be seen as a phenotype and 10100 as a genotype representing it.

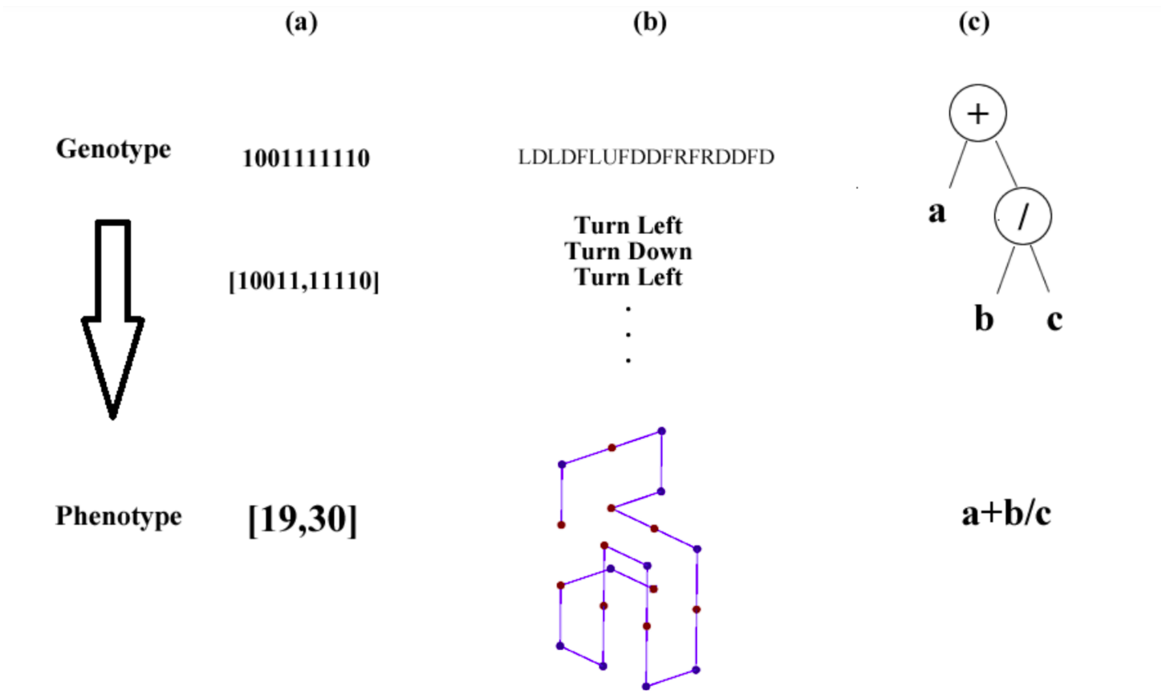


Figure 2.1. Examples of genotype-phenotype mapping (a) Integere representation (b) Protein structure representation on a lattice model (c) Tree representation for a mathematical expression [6]

2.1.2 Objective function

The role of the objective function, is to represent the requirement to adapt to. Objective function defines how quality individual is with respect to the problem in consideration. Technically, it is a function which takes an individual as an input and produces a the measure of quality of a given individual as an output. The measure of quality is called *fitness* and the objective function is called *fitness function*.

To remain with the above-mentioned example, the problem was to minimise x^2 on integers. The fitness of the individual represented by the genotype 10100 would be defined as a square of its corresponding phenotype: $20^2 = 400$

2.1.3 Population

Population in a evolutionary algorithm means a set of individuals. Population can be specified only by setting the population size, in other words, how many individuals are in population. This parameter is usually specified by user.

2.1.4 Parent selection

During each *generation* (one iteration of algorithm), a certain part of population is selected to breed offspring. The choice is made similarly to natural selection, in other words, fitter individuals are preferred, nevertheless, low quality individuals are given a small, but positive change to be selected. Otherwise, the EA could become too greedy and get stuck in the local optimum. Parent selection along with the replacement strategy pushes quality improvements. Parent selection as well as other EA procedures are usually stochastic.

Individuals selected by parent selection are called *parents*.

2.1.5 Crossover operator

Crossover is a genetic operator used to combine typically two parents to generate new offsprings. The idea behind crossover is that by mating two individuals with different but desirable features, it is possible to produce offsprings which combines both of those features. Similarly to other genetic operators, crossover is stochastic.

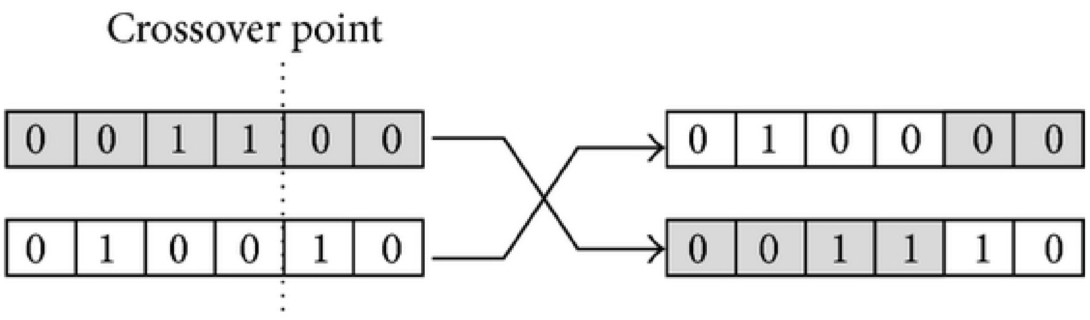


Figure 2.2. Example of One-point crossover (part of chromosome right to the Crossover point are swapped between two parents chromosomes) [10]

2.1.6 Mutation operator

Mutation is an unary genetic operator which changes parts of the chromosome of an individual, typically randomly. In mutation, the mutated individual may change entirely from the original individual. Mutation is used to maintain and introduce diversity in the genetic population.

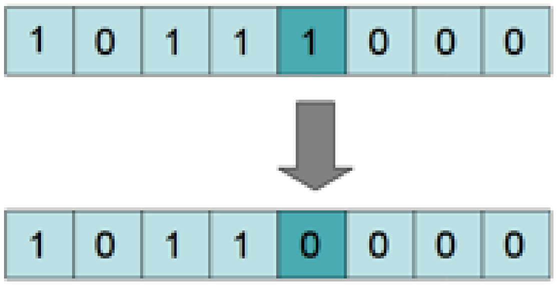


Figure 2.3. Example of mutation [11]

2.1.7 Replacement strategy

Replacement strategy defines which individuals survive and become members of subsequent generation. Typically, the decision is based on the quality of individuals, prefering those with higher fitness. The replacement strategy is similar to parent selection, as both are responsible for promoting quality improvement. However, parent selection is usually stochastic while the replacement strategy is often deterministic.

2.1.8 Initialization

It generates a defined number of individuals of the given representation, thereby creating the initial population. *Initialization* is often done randomly due to lack of knowledge when optimizing black box functions.

2.1.9 Termination condition

The algorithm runs until the *termination condition* has been reached. If we know the optimum of the optimized problem, then reaching the optimum (with a given precision $\epsilon \geq 0$) is a natural termination condition. However, since EAs are stochastic, there is usually no guarantee to reach an optimum and the condition would be never satisfied. Therefore, this condition is extended to a condition which certainly stops the algorithm, such as the limited number of fitness function calls.

2.2 General scheme

In the previous section, the main parts of an EA were introduced individually. By merging all above-mentioned components, the evolutionary algorithm is formed. This section describes the way an EA works as a whole.

```
Algorithm 1: Evolutionary algorithm
  Result: best individual in $Population(t)$
  $t \leftarrow 0$;
  $Population(t) \leftarrow initialization()$;
  $fitnessFunctionEvalutation(Population(t))$;
  while $termination\_condition$ not met do
      $Parents \leftarrow parentSelection(Population(t))$;
      $Offspring \leftarrow crossover(Parents)$;
      $Offspring \leftarrow mutation(Offspring)$;
      $evaluate(Offspring)$;
      $Population(t+1) \leftarrow replacementStrategy(Population(t), Offspring)$;
      $t \leftarrow t+1$;
  end
```

Figure 2.4. General scheme of an evolutionary algorithm

Firstly, an initial population is generated by an initialization procedure. The population is subsequently evaluated by a fitness function. Then starts a generational process.

The generational process is repeated until a terminal condition is not satisfied. Generational process starts by parent selection, usually based on a fitness, some individuals are chosen to seed the new generation. The chosen individuals are combined by a crossover operator to produce offsprings, which are then modified by the mutation operator. Offspring are then evaluated by a fitness function and the generational process ends by creating a new population. Creating a new population is performed with respect to the replacement strategy that selects some newly created offsprings to replace some members of the old population.

The algorithm returns the best individual found so far, eventually some statistics concerning the run of algorithm.

2.3 Differential evolution

This chapter is mainly based on these references: [4, 7]

Differential evolution was introduced by Storn and Price [4] as an efficient evolutionary algorithm initially designed for multidimensional real-valued spaces.

DE utilizes a population of real vectors. The initial population is chosen randomly. After initialization, for each member $\vec{x}_i$ of a population P is generated a so called *mutatant vector*. A mutatant vector is generated by adding the weighted difference between two individuals ($\vec{x}_{r_2}$, $\vec{x}_{r_3}$) to a third individual ($\vec{x}_{r_1}$). These three individuals are mutually exlusive. The mutant vector is then crossed over with $\vec{x}_i$ The offspring $\vec{o}_i$ is then created by crossing over the mutatnt vector with $\vec{x}_i$.

Note that a size of mutatant vector is largely based on the actual variance in the population. The mutant vector will make major changes if the population is spread, on the other hand mutant vecotr will be small if the population is condensed in a particular region. Thus DE belongs to the family of adaptive mutation algorithms.

Lastly, newly created offspring is compared to his parent using the greedy criteria. If the offspring is better, it replace its parent in the population.

To put it more formally, standard DE is defined by specifying of following components of EA:

2.3.1 Representation

Individuals are represented by real-valued vectors:

$$\vec{x_i} = \{x_{i,0}, x_{i,1}, ..., x_{i,D-1}\}, \forall j : x_{i,j} \in \mathbb{R},$$

where i represents the index of the individual in the population P and D stands for dimension of the optimized function. Population is represented as following:

$$P = \{\vec{x}_0, \vec{x}_0, ..., \vec{x}_{NP-1}\}, NP \geq 4,$$

where NP is the size of population.

2.3.2 Mutation

For each individual in the population $\vec{x}_i, i = 0, 1, ..., NP - 1$, DE generates mutant vector $\vec{m}_i$ as following:

$$\vec{m}_i = \vec{x}_{r_1} + F * (\vec{x}_{r_2} - \vec{x}_{r_3})$$

with random, mutually exclusive indexes $r_1, r_2, r_3 \in \{0, 1, ..., NP - 1\}$, which are also chosen to be different from the running index i. F, called *differential weight*, is a constant factor $\in [0, 2]$, representing the amplification of the random deviation $(\vec{x}_{r_2} - \vec{x}_{r_3})$.

2.3.3 Crossover

After mutation the mutant vector $\vec{m}_i$ undergoes a crossover with the its relevant individual $\vec{x}_i$ to generate the offspring $\vec{o}_i$. Standard DE us binomial crossover, where offspring is generated as following:

$$f(x) = \begin{cases} -1 & \text{for } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where: *description of symbols ....*$\vec{x}_i$ is the parent of $\vec{o}_i$. Therefore, it can be seen that each individual from the population generates offspring. In other words, the parent selection chooses all individuals from the population.

The probability of crossover is signed as CR.

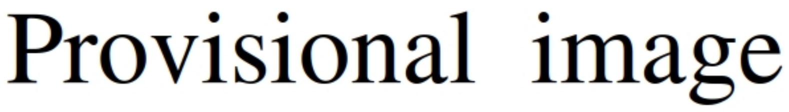
2.3.4 Replacement strategy

To decide which individuals become members of subsequent generation, DE compares offspring $\vec{o}_i$ to its relevant parent $\vec{x}_i$ using the greedy criteria. Thus, if $\vec{o}_i$ is better than $\vec{x}_i$, the offspring $\vec{o}_i$ will replace the parent $\vec{x}_i$ and enter the population of the next generation. To put it more formally, new population is determined as follows:

$$P = \{\vec{p}_i | \vec{p}_i = argmax(\vec{x}_i, \vec{o}_i); i = 0, 1, ..., NP - 1\}$$

```
Algorithm 2: Differential evolution
  Result: best individual found so far
  Generate initial population of size $NP$;
  Evaluate population by fitness function;
  while termination_condition not met do
      for each i individual in population do
          Generate integers $r_1, r_2, r_3 \in [1, NP]$, with $r_1 \neq r_2 \neq r_3 \neq i$ (transitively);
          Generate integer $R \in [0, D]$;
          for each d dimension do
              $\vec{o}_{i,d} = \begin{cases} \vec{x}_{r_1,d} + F * (\vec{x}_{r_2,d} - \vec{x}_{r_3,d}), & \text{if } d = R \text{ or } rand(0,1) < CR \\ \vec{x}_{i,d}, & \text{otherwise} \end{cases}$
          end
          if $\vec{o}_i$ is better than $\vec{x}_i$ then
              Replace $\vec{x}_i$ with $\vec{o}_i$;
          end
      end
  end
```

Figure 2.5. General scheme of the differential evolution



Provisional image

Figure 2.6. DE recombination

Chapter 3
Linkage information modeling

It is worth nothing, that DE, as was described in previous chapter, uses random, uniform crossover. The crossover has no assumptions about the structure of optimized function, specially DE do not take possible dependencies between specific parts of solution into account.

However, there exists a whole class of problems with dependent solution components. De using uniform crossover not only take posible dependencies into consideration, but even very often disrupts linkage between strongly connected components. Therefore it can be seen that mentioned class is a significant weakness of standard DE.

The aim of this work is to propose new crossover operator capable of finding dependencies and taking them into account when generating new offspring. This chapter proposes two possible representations of dependency structure and how to adapt crossover operator.

3.1 Family Of Subsets

Both representations od dependency structure are based on the *Family Of Subsets* (FOS). [12] FOS is a way how to model linkage information that describe presumed dependencies between variables. FOS $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, ...\}$ represents a subset of powerset $\mathcal{P}(\mathcal{I})$ of $\mathcal{I}$, where $\mathcal{I}= \{1, 2, ..., d-1\}$ stands fos a set of indices and d is a number of problem variables (dimension of the fitness function). Each block $\mathcal{F}_j \in \mathcal{F}$ contains the indices of those variables that are considered dependent. The block $\mathcal{F}_j$ divides the set of all variables into two mutually exclusive subsets of variables $\mathcal{F}_j$ and $\mathcal{F} \setminus \mathcal{F}_j$. Variable within those subsets are crossed over together. [12]

To put it more formally, within crossover for each individual $\vec{x}_i$ each block $\mathcal{F}_j$ is iteratively considered in random order. For each block $\mathcal{F}_j$ is randomly generated new mutant vector $\vec{m}_i$ in the same way as standard mutation. If the mutants values for variables contained in $\mathcal{F}_j$ are different from those in parent $\vec{x}_i$, then these value are overwritten in the parent $\vec{x}_i$, this produces $\vec{x}_{i,new}$ which is then evaluated by the fitness function. New individual $\vec{x}_{i,new}$ is only accented if it has better or equal fitness value than the original $\vec{x}_i$. How the DE changes is captured in the figure 3.2.

3.2 Linkage tree

There exist many FOS structures and any of them can be used to model linkage structure, however this work focus on two of them. First of them is *linkage tree* (LT-FOS).

“*The Linkage Tree is the hierarchical cluster tree of the problem variables using an agglomerative hierarchical clustering algorithm with a distance measure $\mathcal{M}$. The distance measure $\mathcal{M}(X_1, X_2)$ measures the degree of dependency between two sets of variables X_1 and X_2.*” [13]

There exist more potential distance measures $\mathcal{M}(X_1, X_2)$, however, in this work, two distance measures are used. They are described in detail in following chapter 4.

The linkage tree is a tree with D leaf nodes and $D-1$ inner nodes, where D is number of problem variables. Each node of the LT-FOS represents certain block of variables $\mathcal{F}_j$. The key property of the LT-FOS is that each $\mathcal{F}_j$ which containts more than one variable is the union of two other sets $\mathcal{F}_k$, $\mathcal{F}_l \in \mathcal{F}$, where $j \neq k \neq l$ (transitively). To put it more formally, for any subset $\mathcal{F}_i$, where $|\mathcal{F}_j| > 1$, there exist subsets $\mathcal{F}_k, \mathcal{F}_l$ for which the following applies:

1) $\mathcal{F}_k, \mathcal{F}_l \neq \emptyset$
2) $\mathcal{F}_k \cap \mathcal{F}_l = \emptyset$
3) $\mathcal{F}_k \cup \mathcal{F}_l = \mathcal{F}_j$

The hierarchical clustering procedure starts by assigning each problem variable to a separate block in random order. The procedu proccedes bottom-up, therefore the tree is initialized with these univariete blocks as leaves. In each step new node is created by merging two nodes of the tree which were determined, by given distance measure $\mathcal{D}$, as the most dependent. It is important to mention that each node can be merged only once. The process of merging stops when no more merges are possible to put it another way, root node has been created. Due to the way the procedure works, root node has to be a set of all problem variables. The tree itself contains multiple levels of dependencies. From univariate level at a height of zero to complete dependency between all variables at a depth of zero. [13]

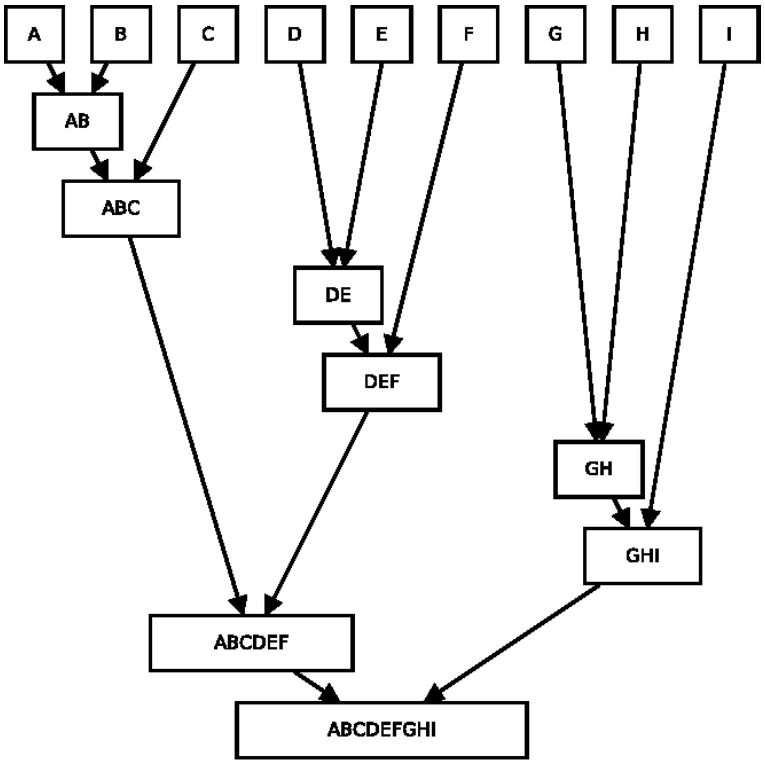


Figure 3.1. Example Linkage tree [14]

The *DE using the LT FOS structure* (LT-FOS DE) build the LT-FOS in every generation. Once the tree is built, LT-FOS DE traverses the tree in the opposite order of the merging.

3.3 Marginal product

Second introduced FOS structure is *marginal product* (MP-FOS) [15]. The MP-FOS is defined as set $\mathcal{F}$, where for each $\mathcal{F}_k, \mathcal{F}_l \in \mathcal{F}$ holds that $\mathcal{F}_k \cap \mathcal{F}_l = \emptyset$. When all variables are independent, MP-FOS is called univariate FOS and $\mathcal{F} = \{\{0\}, \{1\}, ..., \{D-1\}\}$, where D is number of problem variables. On the contrary, when all variables are considered dependent, MP-FOS is called compact FOS.

Before introduction of the *MP-FOS building procedure* it is necessary to define *strength of block* $\mathcal{S}_{\mathcal{M}}(\mathcal{F}_j)$ which determines dependency rate within certain block $\mathcal{F}_j$ according to given distance measure $\mathcal{M}$. The strength of block is defined as following:

$$\mathcal{S}_{\mathcal{M}}(\mathcal{F}_i) = \begin{cases} \frac{C}{D-1} \sum_{v \in \mathcal{I}} \mathcal{M}(\mathcal{F}_i, \{v\}) & \text{if } |\mathcal{F}_i| = 1, \\ \frac{1}{|\mathcal{F}_i|(|\mathcal{F}_i|-1)} \sum_{u \in \mathcal{F}_i} \sum_{v \in \mathcal{F}_i} \mathcal{M}(\{u\}, \{v\}) & \text{otherwise,} \end{cases} \tag{1}$$

where $C \geq 0$ is user selected factor defining the degree of strength of univariete blocks.

The MP-FOS building procedures starts by initializing MP-FOS $\mathcal{F}$ as univariete FOS and by assigning the strength of block to each block. In each step new block $\mathcal{F}_n$ is created by merging two blocks $\mathcal{F}_a, \mathcal{F}_b \in \mathcal{F}$, which are determined, by given distance measure $\mathcal{M}$, as the most dependent. Then, $\mathcal{F}_n$ is assigned its strength of block. If newly created block meets the following conditions:

1) $\mathcal{S}_{\mathcal{M}}(\mathcal{F}_n) \geq \theta_1, \theta_1 \in \mathbb{R}$
2) $\mathcal{S}_{\mathcal{M}}(\mathcal{F}_n) \geq K max(\mathcal{S}_{\mathcal{M}}(\mathcal{F}_a), \mathcal{S}_{\mathcal{M}}(\mathcal{F}_b))$,
3) $|\mathcal{F}_n| \leq \theta_2, \theta_2 \in \mathbb{N}$,

where thresholds θ_1 and θ_2 are defined by user, usually chosen as $\theta_1 > 0$ and $\theta_2 < D$. Then $\mathcal{F}_n$ is inserted to the FOS $\mathcal{F}$ and $\mathcal{F}_a, \mathcal{F}_b$ are removed from $\mathcal{F}$. The procedure runs until a newly created block $\mathcal{F}_n$ has not met mentioned conditions or until MP-FOS has became the compact FOS.

The *DE using the MP FOS structure* (MP-FOS DE) build the MP-FOS in every generation. After building the MP-FOS, MP-FOS DE traverses FOS in the opposite order of the merging, in other words, from the last one added to FOS to the first one.

```
Algorithm 3: Differential evolution with known FOS
  Result: best individual found so far
  Generate initial population of size $NP$;
  Evaluate population by fitness function;
  while termination_condition not met do
      for each i individual in population do
          for each block $\mathcal{F}_j \in \mathcal{F}$ do
              $\vec{x}_{new} = \vec{x}_i$;
              Generate integers $r_1, r_2, r_3 \in [1, NP]$, with $r_1 \neq r_2 \neq r_3 \neq i$ (transitively);
              for each variable $v \in \mathcal{F}_j$ do
                  $\vec{x}_{new,v} = \vec{x}_{r_1,v} + F * (\vec{x}_{r_2,v} - \vec{x}_{r_3,v})$;
              end
              if $\vec{x}_{new}$ is better than $\vec{x}_i$ then
                  Replace $\vec{x}_i$ with $\vec{x}_{new}$;
              end
          end
      end
  end
```

Figure 3.2. provisional pseudocode

Chapter 4
Identification of the linkage structure

In previous chapter two possible representations of dependency structure were introduced. In order to represent the dependency structure, it is necessary to determine the degree of dependence between each pair of variables and furthermore between each pair of sets of variables. The tool used to measure the degree is called the distance measure and is denoted as $\mathcal{M}$.

Formally, $\mathcal{M}$ is a function that takes two sets of variables as input and produces a real, positive number as output. $\mathcal{M}$ is defined as follows [13]:

$$\mathcal{M}(X_i, X_j) = \frac{1}{|X_i||X_j|} \sum_{u \in X_i} \sum_{v \in X_j} n_{u,v} \tag{1}$$

where X_i and X_j are sets of variables and $n_{u,v}$ is element of the *dependency matrix* $\mathcal{N}$ at position u, v.

The dependency matrix

$$\mathcal{N} = \begin{pmatrix} n_{0,0} & n_{0,1} & \cdots & \cdots & n_{0,D-1} \\ n_{1,0} & n_{1,1} & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ n_{D-1,0} & \cdots & \cdots & \cdots & n_{D-1,D-1} \end{pmatrix} \in \mathbb{R}^{D \times D}$$

is a symetric and positive semi-definite matrix. The dependency matrix gives the dependency between each pair of variables specifically, the element $n_{i,j}$ denotes he pairwise dependency strength between i-th and j-th problem variables. Consequently $\mathcal{N}$ contains zeros on diagonal i.e. $\forall i = 0, 1, ..., D-1 : n_{i,i} = 0$.

There are several methods how to contruct the dependency matrix $\mathcal{N}$ nevertheless, this work focus only on two of them.

4.1 Fitness-based method

The first method called *non-linearity check* (NC) is to define whether two variables interact is directly based on fitness values. The method works under the assumption that non-linear interactions may exist only between dependent variables. Specially, it classify a pair of variables either separable or non-separable by comparing the difference in overall fitness while making the exact same change for a certain pair of chromosome of given individual $x_{i,j}$ for different values of $x_{i,k}, k \neq j$. [16–17] Nevertheless, checking only one individual is not convincing enough, because there may exist linearity between a dependent pair of variables in some context, therefore more individuals must be checked. In this work, m best individuals from the population are checked, where $m = max\{2, \frac{3}{20}NP\}$. C denotes the set of indices of m best individuals in population.

For each of chosen individuals $\vec{x}_i, i \in C$ and for each pair of variables j, k a pairwise dependency $d_{i,j,k}$ is calculated. The overall pairwise dependency between those variables is determined by aggregating those values as following:

$$n_{j,k} = \frac{1}{m} \sum_{i \in C} d_{i,j,k}.$$

In order to calculate $d_{i,j,k}$, four individuals are picked by combining all possible points that can be created by picking two different values for each $x_{i,j}$ and $x_{i,k}$. [18] The absolute value of differences in overall fitness value for those point are used to calculate the potential dependence between *j-th* and *k-th* variables by determining whether the adjustment to $x_{i,k}$ affect the change in fitness caused by modification to $x_{i,j}$. Define:

$$\Delta_{i,j} = |(f(\vec{x}_i)|x_{i,j} = a_j, x_{i,k} = a_k) - (f(\vec{x}_i)|x_{i,j} = a_j + b_j, x_{i,k} = a_k)|,$$

$$\Delta_{i,j,k} = |(f(\vec{x}_i)|x_{i,j} = a_j, x_i, k = a_k + b_k) - (f(\vec{x}_i)|x_{i,j} = a_j + b_j, x_{i,k} = a_k + b_k)|,$$

where f denotes fitness function, a_j, a_k, b_j, b_k can be any real value, such that for every variable j, a_j and $a_j + b_j$ remains within the bound for $x_{i,j}$ inside the current population, to put it more formally:

$$\forall j : \max_{\vec{x}_i \in P}(x_{i,j}) \geq a_j \geq \min_{\vec{x}_i \in P}(x_{i,j}),$$

$$\forall j : \max_{\vec{x}_i \in P}(x_{i,j}) \geq a_j + b_j \geq \min_{\vec{x}_i \in P}(x_{i,j}),$$

nevertheless, in this work are used values that have been empirically found for [18], which are

$$a_j = \min_{\vec{x}_i \in P}(x_{i,j}) + (\max_{\vec{x}_i \in P}(x_{i,j}) - \min_{\vec{x}_i \in P}(x_{i,j})) * 0.35,$$

$$b_j = (\max_{\vec{x}_i \in P}(x_{i,j}) - \min_{\vec{x}_i \in P}(x_{i,j})) * 0.35.$$

Finally, *j-th* and *k-rh* are said to dependent when $|\Delta_{i,j} - \Delta_{i,j,k}| \geq 0$, the pairwise dependent $d_{i,j,k}$ is defined as:

$$d_{i,j,k} = \begin{cases} 1 - \frac{\Delta_{i,j,k}}{\Delta_{i,j}} & \text{if } \Delta_{i,j} \geq \Delta_{i,j,k}, \\ 1 - \frac{\Delta_{i,j}}{\Delta_{i,j,k}} & \text{otherwise.} \end{cases} \tag{2}$$

note that $d_{i,j,k}$ as well as $n_{j,k}$ lie within $[0, 1)$ with 0 indicating independent variables.

4.2 Distribution-based mathod

References

[1] R. M. Friedberg. A Learning Machine: Part I. *IBM J. Res. Dev.*. 1958, 2 2-13.

[2] R. M. Friedberg, B. Dunham, and J. H. North. A Learning Machine: Part II. *IBM J. Res. Dev.*. 1959, 3 282-287.

[3] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10 - Companion Publication*. 2010, 1689-1696. DOI 10.1145/1830761.1830790.

[4] Rainer Storn, and Kenneth Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*. 1997, 11 341-359. DOI 10.1023/A:1008202821328.

[5] A. E. Eiben, and James E. Smith. *Introduction to Evolutionary Computing*. 2nd edition. Springer Publishing Company, Incorporated, 2015. ISBN 3662448734.

[6] Ka-Chun Wong. *Evolutionary Algorithms: Concepts, Designs, and Applications in Bioinformatics: Evolutionary Algorithms for Bioinformatics*. 2015.

[7] Sean Luke. *Essentials of Metaheuristics* . 2009 .
`http://cs.gmu.edu/~sean/book/metaheuristics/` .

[8] Darwin Charles. *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859.

[9] Spencer Herbert. *The Principles of Biology*. London: William and Norgate, 1864.

[10] Marco Ferreira, J. Bagari■, Jose Lanza-Gutierrez, Sílvio Mendes, Jo■o Pereira, and Juan A. Gomez-Pulido. On the Use of Perfect Sequences and Genetic Algorithms for Estimating the Indoor Location of Wireless Sensors. *International Journal of Distributed Sensor Networks*. 2015, 2015 1-12. DOI 10.1155/2015/720574.

[11] Kendall Park. *0-1 Knapsack Approximation with Genetic Algorithms* . 2014 .
`https://github.com/KendallPark/genetic-algorithm` .

[12] Peter A.N. Bosman, and Dirk Thierens. More Concise and Robust Linkage Learning by Filtering and Combining Linkage Hierarchies. 2013, 359–366. DOI 10.1145/2463372.2463420.

[13] ”Thierens. ”Berlin: ”Springer Berlin Heidelberg”, ISBN ”978-3-642-15844-5” .

[14] Brian W. Goldman, and D. Tauritz. Linkage tree genetic algorithms: variants and analysis. 2012,

[15] Anton Bouter, Tanja Alderliesten, Cees Witteveen, and Peter A. N. Bosman. Exploiting Linkage Information in Real-Valued Optimization with the Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm. 2017, 705–712. DOI 10.1145/3071178.3071272.

[16] M. Munetomo, and D. E. Goldberg. A genetic algorithm using linkage identification by nonlinearity check. 1999, 1 595-600 vol.1. DOI 10.1109/ICSMC.1999.814159.

[17] Masaharu Munetomo, and David Goldberg. Linkage Identification by Non-monotonicity Detection for Overlapping Functions. *Evolutionary computation.* 1999, 7 377-98. DOI 10.1162/evco.1999.7.4.377.

[18] C. Olieman, A. Bouter, and P. A. N. Bosman. Fitness-Based Linkage Learning in the Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm. *IEEE Transactions on Evolutionary Computation.* 2021, 25 (2), 358-370. DOI 10.1109/TEVC.2020.3039698.