CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Differential Evolution Crossover with Dependency Detection

Vojtěch Voráček

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Voráček Vojtěch**   Personal ID number: **483769**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Differential Evolution Crossover with Dependency Detection

Bachelor's thesis title in Czech:

Křížení pro diferenciální evoluci s detekcí závislostí

Guidelines:

Differential Evolution (DE) is a successfull black-box optimization algorithm for real-valued problems. Its weakness is a class of problems with dependent solution components, the exponential or binomial crossover operators conventionally used in DE are not suitable for them. The goal of this bachelor project is to design a new crossover operator based on the linkage tree known from the genetic algorithms with binary representation.
1. Familiarize yourself with the basic variants of differential evolution and the binomial and exponential crossover operators.
2. Learn about dependencies between solution components and about models used to describe them. Focus on the linkage tree model used to solve binary optimization problems.
3. Find or propose a new way of building a linkage tree-like model for real-valued representation. Propose a modified crossover operator for DE based on the model.
4. Evaluate the proposed algorithm on a set of benchmark functions and compare it with the basic DE, and other optimization algorithms.

Bibliography / sources:

[1] Storn, R., Price, K. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997). https://doi.org/10.1023/A:1008202821328
[2] C. Olieman, A. Bouter and P. A. N. Bosman, "Fitness-based Linkage Learning in the Real-Valued Gene-pool Optimal Mixing Evolutionary Algorithm," in IEEE Transactions on Evolutionary Computation, doi: 10.1109/TEVC.2020.3039698
[3] Reshef, David N et al. "Detecting novel associations in large data sets." Science (New York, N.Y.) vol. 334,6062 (2011): 1518-24. doi:10.1126/science.1205438

Name and workplace of bachelor's thesis supervisor:

Ing. Petr Pošík, Ph.D., Department of Cybernetics, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **07.01.2021**   Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

_____   _____   _____
Ing. Petr Pošík, Ph.D.   prof. Ing. Tomáš Svoboda, Ph.D.   prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature   Head of department's signature   Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

_____   _____
Date of assignment receipt   Student's signature

Acknowledgement / Declaration

First of all, I would like to express my gratitude to my supervisor Ing. Petr Pošík, Ph.D. for his guidance, his helpful comments, and his patience with me.

Thank you!

Many thanks belong also to my family, current and future, and to all my friends for their support.

Love you!

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 20. May 2020

....

Abstrakt / Abstract

Diferenciální evoluce je považována za jeden z nejlepších evolučních algoritmů pro spojité black-box optimalizační problémy. Originální verze diferenciální evoluce použí náhodný uniformí operátor křížení, který nebere v potaz potencialní závislosti mezi částmi řešení a může tyto vazby narušovat.

Cílem této práce je poskytnout nový operátor křížení pro diferenciální evoluci, který bude vhodnější pro třídu problémů obsahující závislé komponenty řešení.

V této práci jsou prezentovány dvě možnosti, jak nalézt závislosti mezi proměnnými problému a dvě možnosti, jak modelovat strukturu vazeb. S užitím těchto metod byly navženy čtyři nové operátory křížení.

Nově navržené algoritmy jsou vyhodnoceny na množině referenčních funkcí a jsou porovnány s dalšími optimalizačními algoritmy, včetně original diferenciální evoluce.

Výsledky ukazují, že nově navžené algoritmy dosahují výrazně lepšího výkonu a škálovatelnosti než originální diferenciální evoluce ve smyslu potřebného počtu vyhodnocení účelové funkce k nalezení globálního optima. Něktére z nich podobné škálovatelnosti jako CMA-ES, jeden z nejmodernějších evolučních algoritmů.

Klíčová slova: Evoluční algoritmus, diferenciální evoluce, vazebný strom, mezní produkt, kontrola nelinearity, maximální informační koeficient, učení se závislostí.

Překlad titulu: Křížení pro diferenciální evoluci s detekcí závislostí

The differential evolution is considered as one of the best evolutionary algorithms for continuous black-box optimization problems. The original version of differential evolution uses random uniform crossover operator which do not take possible dependencies between parts of solution into account and may disrupts these linkages.

The goal of this work is to propose new a crossover operator for differential evolution more suitable for the class of problems containing dependent solution components of a solution.

In this work are presented two options to find dependencies between problem variables and two possibilities to model the linkage structure. Finally, with use of those methods four new crossover operators are designed.

Newly proposed algorithms are evaluated on a set of benchmark functions and compared with other optimization algorithms including the original differential evolution.

The results show that all newly proposed algorithms achieve significantly better performace and scalability than original differential evolution in terms of fitness function evaluations needed to find a global optimum. Some of them achieve comparable scalability to state-of-the-art evolutionary algorithm CMA-ES.

Keywords: Evolutionary algorithm, differential evolution, linkage tree, marginal product, non-linearity check, maximal information coefficient, linkage learning.

Contents /

Tables / Figures

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" "takto" 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.

■ 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.

■ 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.

■ 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 2.1. 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 2.2. General scheme of the differential evolution

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). [10] 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. [10]

To put it more formally, within crossover for each individual $\vec{x}_i$ each block $\mathcal{F}_j$ is iteratively considered in random order (crossover probability CR = 1). 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).

"*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.*" [11]

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 represents certain block of variables $\mathcal{F}_j$. The key property of the LT 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 $\mathcal{M}$, 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. [11]
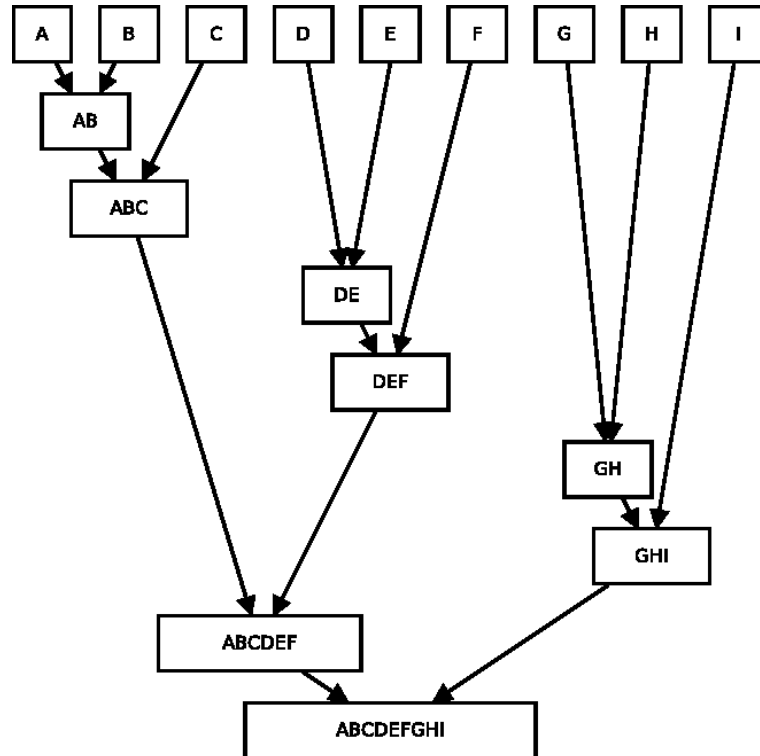

Figure 3.1. Example Linkage tree [12]

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

3.3 **Marginal product**

Second introduced FOS structure is *marginal product* (MP) [13]. The MP 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 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 pairwise dependent, MP is called compact FOS.

Before introduction of the *MP 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 building procedures starts by initializing MP $\mathcal{F}$ 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) $\mathcal{S}_{\mathcal{M}}(\mathcal{F}_n) \geq Kmax(\mathcal{S}_{\mathcal{M}}(\mathcal{F}_a), \mathcal{S}_{\mathcal{M}}(\mathcal{F}_b)),$
3) $|\mathcal{F}_n| \leq \theta_2, \theta_2 \in \mathbb{N},$

where thresholds $\theta_1 > 0$, $\theta_2 \in [1, D]$ and factor $K \in (0, 1]$ are defined by user. 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 has became the compact FOS.

The *DE using the MP FOS structure* (MP DE) build the MP in every generation. After building the MP, MP 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 [11]:

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

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

The dependency matrix

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

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

There are several methods how to contruct the dependency matrix $\mathcal{P}$ 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 $x_{i,k}, k \neq j$. [14–15] 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:

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

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 $x_{i,k}$. [16] 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 [16], 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 $p_{j,k}$ lie within $[0, 1)$ with 0 indicating independent variables.

4.2 Distribution-based mathod

The second method to construct the dependency matrix $\mathcal{P}$ is called the *maximal information coefficient* (MIC) [17]. There exists more methods used to identify dependencies between a pair of variables based on the distribution of the population [13, 16], however MIC achieved better accurancy in comparison to other methods. [17–18]

MIC is based on the idea, that if a relationship between a pair of variables exists, then it is posible to draw a grid on the scatterplot of the two variables that partitions the data to encapsulate that relationship.

To calculate MIC, all possible grids up to maximal grid resolution are considered. Note that maximal grid resolution depends on the sample size. For each pair of integers (x, y) the largest possible mutual information (MI) [19] achieveable by any x-by-y grid applied to the data is computed. Those mutual information values are then normalized by the logarithm of the minimum x and y. Finally, MIC is defined as maximum of thos highest normalized mutual information values. [17] Formally:

$$MIC_{i,j} = \max_{(x,y):x \leq B, y \leq B} \left(\max_{g:G_{x,y}} \left(\frac{MI_{i,j}|_g}{\log \min(x, y)} \right) \right),$$

where B is user-specified value defining maximal grid resolution, $G_{x,y}$ denotes a set of all possible x-by-y grids and $MI_{i,j}|_g$ stands for mutual information of i-th and j-th variables achieved by application of grid g.

As was mentioned above achieved good results in various comparisons, however a big limitation of MIC is its high computational cost. Therefore several optimized algorithms for approximating the MIC have been published. [17, 20–21]. In this work MICE *minepy* implementation [22–23] is used.

In this wrok MICE is not calculated from the whole population, but only subset C of all individuals in population is considered. The dependency matrix $\mathcal{P}$ is then formally calculated as following:

$$p_{i,j} = \text{MICE}_{i,j}|_C.$$

Chapter 5
Experiments

In 2.3 section the standard differential evolution was introduced. It was also noted that DE does not have any tool to recognize or model the linkage information between certain parts of solution. In chapter 3 two possible representations of dependency structure were introduced assuming known pairwise dependencies and in the chapter 4 two ways to find pairwise dependencies and thereby build the dependency matrix $\mathcal{P}$ were presented.

By combination of above it is possible to propose adjusted DE with dependency detection. The adjusted version differs from original in two factors. Firstly, in every generation the dependency matrix $\mathcal{P}$ and FOS structure based on it is built. Secondly, crossover is modified to respect dependent blocks.

Algorithm 4: DE with dependency detection

Result: best individual found so far
Generate initial population of size NP;
Evaluate population by fitness function;
while *termination_condition not met* **do**
  Create dependency matrix $\mathcal{N}$;
  Build FOS $\mathcal{F}$ based on $\mathcal{N}$;
  for *each $\vec{x}_i$ individual in population* **do**
    for *each block $\mathcal{F}_j \in \mathcal{F}$ in opposite order of merging* **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 5.1. Pseudocode of the differential evolution with dependency detection

The main goal of experiments is to study performance of various types of DE with dependency detection differing in creating the matrix $\mathcal{P}$, or in the building FOS $\mathcal{F}$. Compare them between each other and with standard DE and other optimization algorithms.

5.1 Algorithms

■ 5.1.1 **Differential evolution variants**

Within the experiments seven types of differential evolution were compared. The original DE as was introduced in 2.3 section. (DE_UNIFORM). Remaining six variants of DE are divided into three pairs according to how they create the distance matrix $\mathcal{P}$. Within each pair, one variant uses linkage tree FOS (LT) and second marginal product FOS (MP). The first pair uses nonlinearity check to create $\mathcal{P}$ (DE_LT_NC and DE_MP_NC). The second pair take advantage of maximal information coefficient (DE_LT_MIC and DE_MP_MIC). The third pair are DE variants with full prior knowledge of pairwise dependencies, therefore they build optimal $\mathcal{P}$, which is $(0, 1)$-matrix with zeros for independent pairs and ones for dependent (DE_LT+ and DE_MP+).

	Nonlinearity check	Max. inf. coeff.	Optimal
Linkage tree	DE_LT_NC	DE_LT_MIC	DE_LT+
Marginal product	DE_MP_NC	DE_MP_MIC	DE_MP+

Table 5.1. Overview of newly proposed variants of the differential evolution.

All above-mentioned types shares the following:

- Initialization of individuals $\sim \mathcal{N}(\mathbf{0}, 100 \cdot \mathbf{I}_D)$, where $\mathcal{N}$ is multivariate normal distribution, $\mathbf{0}$ stands for the zero vector and $\mathbf{I}_D$ represents $D \times D$ identity matrix.
- The differential weight $F = 0.7$

Other parameters:

- The crossover probability for DE_UNIFORM: $CR = 0.9$
- The degree of strength of univariate blocks:

$$C = \begin{cases} 1 & \text{for DE\_MP+,} \\ 2 & \text{otherwise.} \end{cases}$$

- Threshold θ_1 defining the minimal strength of block to be accepted:

$$\theta_1 = \begin{cases} 10^{-1} & \text{for DE\_MP\_MIC,} \\ 10^{-8} & \text{otherwise.} \end{cases}$$

- Maximal size of block: $\theta_2 = 6$
- Maximum potential degree of strength of block reduction during merging

$$K = \begin{cases} 0.8 & \text{for DE\_MP+,} \\ 0.4 & \text{for DE\_MP\_NC,} \\ 0.7 & \text{for DE\_MP\_MIC.} \end{cases}$$

- Number of checked individuals within non-linearity check method: $m = \lceil 0.15 \cdot NP \rceil$
- The subset used to calculate MICE $C = C_b \cup C_r$, where C_b is set of $\lceil 0.3 \cdot NP \rceil$ best individuals in population and C_r is a set of $\lceil 0.1 \cdot NP \rceil$ randomly chosen individuals from the remaining.
- The maximal MICE grid resolution B is set according to the table 5.2 (rounded to the nearest integer in an upward direction).
- For the MICE parameter c which determines how many more clumps there will be than columns in every partition was set default value 15 [22]

All above-mentioned values were found empirically unless otherwise stated.

Number of samples	B parameter
$|C| < 25$	$|C|^{0.85}$
$25 \le |C| < 50$	$|C|^{0.8}$
$50 \le |C| < 250$	$|C|^{0.75}$
$250 \le |C| < 500$	$|C|^{0.7}$
$500 \le |C| < 1000$	$|C|^{0.65}$
$1000 \le |C| < 2500$	$|C|^{0.60}$
$2500 \le |C| < 5000$	$|C|^{0.55}$

Table 5.2. The dependence of the cardinality of C on the parameter B, taken from [22].

5.1.2 Other algorithms

Covariance matrix adaptation evolution strategy (CMA-ES) belongs to the class of evolutionary algorithms. CMA-ES is considered state-of-the-art in evolutionary computation and has very quickly become the standard tool for continuous optimisation. [24–26] In this work the Hanses's implementation of CMA-ES with default parameters is used. [27]

Last considered algorithm is Nelder-Mead simplex algorithm [28], the optimization algorithm which is not an evolutionary algorithm nevertheless, it uses only function values, therefore may be used for blackbox optimization. The Scipy implementation called FMIN is used [29]. Minimal absolute difference in candidate solution between iterations (*xtol*) as well as minimal absolute difference in fitness function values between iterations (*ftol*) is set to 10^{-12}. Independent restarts are allowed.

5.2 Test problems

The first set of benchmarking problems is called Test problems. Those problems are in a sence suitable for above-introduced algorithms (unconstrained, noiseless, ...). These six optimization problems to minimise are considered to study the impact of various types of linkage learning on the performace of DE and to benchmark considered algorithms.

Before the introduction of the Test problems, state the important property of functions, the *additive separability*. Define additively separable function F as:

$$F(x_0, x_1, ..., x_{D-1}) = f_0(x_0) + f_1(x_1) + ... + f_{D-1}(x_{D-1}),$$

where $f_0, f_1, ..., f_{D-1}$ are functions of one variable. It is crucial that the optimum of D-dimensional additively separable function may be obtained by performing D independent one-dimensional optimizations along each dimension, formally:

$$\min_{[x_0,x_1,...,x_{D-1}]\in\mathbb{R}^{\mathbb{D}},} F(x_0, x_0, ..., x_{D-1}) = \min_{x_0\in\mathbb{R}} f_0(x_0) + \min_{x_1\in\mathbb{R}} f_1(x_1) + ... + \min_{x_{D-1}\in\mathbb{R}} f_{D-1}(x_{D-1})$$

It can be seen that standard DE which optimize each dimension independently would be suitable for optimizing additively separable functions because additively separable function are exactly those function without dependencies between variables.

5.2.1 Sphere

The first benchmark function is sphere funcion also known as De Jong F1 [30]. It is presumable the easiest continous domain optimization problem. It is convex, separable and has one local minimum.

Definition of the sphere function:

$$f_{sphere}(\vec{x}) = \sum_{i=0}^{D-1} x_i^2$$

Global minimum:

$$f_{sphere}(\vec{x}_{min}) = 0$$
$$\vec{x}_{min} = [0, 0, ..., 0]$$

■ 5.2.2 Levy

The second considered benchmark problem is Levy function [31]. Like Sphere, it is separable function with one local minimum nevertheless, Levy function is considered more difficult to optimize.

The Levy function is defined as follows:

$$f_{Levy}(\vec{x}) = \sin^2(\pi v_0) + \sum_{i=0}^{D-2} \left[(v_i-1)^2(1+10\sin^2(\pi v_i+1))\right] + (v_{D-1}-1)^2(1+sin^2(2\pi v_{D-1})),$$

where $v_i = 1 + \frac{x_i-1}{4}$, for all $i = 0, 1, ..., D-1$.

Global minimum:

$$f_{Levy}(\vec{x}_{min}) = 0$$
$$\vec{x}_{min} = [1, 1, ..., 1]$$

■ 5.2.3 Rastrigin

The Rastrigin function [32–33] is the third benchamrk problem. Like the previous functions, this one is also separable. It it difficult function to optimize, due to regular "noise" it has many regularly distributed local minimum. The Rastrigin function is defined as:

$$f_{Rastrigin}(\vec{x}) = 10 \cdot D + \sum_{i=0}^{D-1} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Global minimum:

$$f_{Rastrigin}(\vec{x}_{min}) = 0$$
$$\vec{x}_{min} = [0, 0, ..., 0]$$

■ 5.2.4 Rosenbrock

The Rosenbrock function [34], also known as Banana function is the first considered unseparable function, because it has overlapping dependencies. Each pair of consecutive variables is dependent. The Rocenbrock function contains narrow, parabolic valley, where the global minimum is located. However, even though this valley is easy to find, convergence to the minimum is difficult [35]. The definition of Rosenbrock function is as follows:

$$f_{Rosenbrock}(\vec{x}) = \sum_{i=0}^{D-2} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right]$$

Global minimum:

$$f_{Rosenbrock}(\vec{x}_{min}) = 0$$
$$\vec{x}_{min} = [1, 1, ..., 1]$$

5.2.5 SoREB

The Sum of Rotated Ellipsoid Blocks or abbreviated SoREB [13] is defined as follows:

$$f_{Ellipsoid}(\vec{x}) = \sum_{i=0}^{l-1} \left[10^{\frac{6i}{l-1}} x_i^2 \right]$$

$$f_{SoREB}(\vec{x}, k) = \sum_{i=0}^{D/k-1} \left[f_{Ellipsoid}\big(R_\theta([x_{ki}, ..., x_{k(i+1)-1}])\big) \right]$$

Where R_θ defines the rotation of a vector around the origin by the angle of θ and k is size of block, defined by user. Rotated blocks of variables which enter to $f_{ellipsoid}$ as an input creates strongly connected components. Variables within the block have strong dependencies but are completely independent of any variables outside their block. This feature is called block-separability.

Within comparison four types of the SoREB function differing in the size of blocks (2, 3, 4, 5) were considered. The rotation of $\theta = \pi/8$ was used with one exception, if $D = 5$, k is set to 4.

Global minimum:

$$f_{SoREB}(\vec{x}_{min}, k) = 0; k \in \mathbb{N}$$

$$\vec{x}_{min} = [0, 0, ..., 0]$$

5.2.6 OSoreb

The SoREB function conations only non-overlapping non-decomposable block of size k. In [13] the overlapping version of this problem was defined as OSoREB. In addition to the original SoREB problem a SoREB blocks of length 2 for every pair of successive variables in belonging to other original blocks are used. For OSoREB is used $k = 5$ and $\theta = \pi/8$. Definition of OSoREB:

$$f_{OSoREB}(\vec{x}, k) = f_{SoREB}(\vec{x}, k) + \sum_{i=1}^{D/k-1} \left[f_{Ellipsoid}\big(R_\theta([x_{ki-1}, x_{ki}])\big) \right]$$

Global minimum:

$$f_{OSoREB}(\vec{x}_{min}, k) = 0; k \in \mathbb{N}$$

$$\vec{x}_{min} = [0, 0, ..., 0]$$

5.3 Black Box Optimization Benchmarking problems

Black Box Optimization Benchmarking (BBOB) problems is the second consedered set of benchmarking problems. It is a set of 24 noise-less real-parameter single-objective benchmark functions defined in [36] as Real-Parameter Black-Box Optimization Bench- marking 2009 Noiseless Functions. Those function were used for BBOB workshop 2009. The BBOB problems were selected with intention to evaluate the performance of algo- rithms with regard to standard difficulties which occurs in continuous domain search. So they definitely should, at least to a certain extent, reflect problems that are dealt with in practice. All BBOB problems are to be minimised.

It is important to note that since BBOB problems cover wide range of possible optimization problems, proposed DE variants may be unsuitable for some of them. For instance, proposed DE variants lack a tool to handle constrains.

5.4   Setup

All experimental results described in this work 6 measures the first time a global optimum was hit, in other words, the number of fitness function calls needed to reach the small enough neigbourhood of the global optimum for the first time within the run. The toleration is 10^{-8}.

5.4.1   Test problems specifics

For each problem, each algorithm and each dimension 25 independent runs are performed. The performance is considered successful if at least 24 runs converged to the global optimum or to a predefined sufficiently close approximation within $300000 \cdot D$ calls of the fitness function.

The associate population size of evolutionary algorithms is the smallest possible size so that the algorithms performance is considered successful. It is determined by starting from smallest possible population and letting the algorithm runs 25 times. If the performance have not been successful, the population size for the next trial will increase by k. This procudere is repeated until the successful population size is found ot until population size reaches upper limit T. In the first case optimal population size is searched for by performing bisection search between the current poopulation size and the previous size. In the second case is considered unsuccessful for certain problem and dimension.

5.4.2   BBOB problems specifics

The setup for BBOB problems is partially determined by authors of BBOB problems in [37]. For each algorithm, dimension and optimized function five different function *instances* are used, each of them three times. The number of fitness functon calls is limited to $1000000 \cdot D$. The population size is set to 25, except for algorithms using MIC, for which it is increased to 50. For BBOB problems algorithm DE_LT+ and DE_MP+ are not considered, because the dependency structure of the problems may be unclear.

BBOB problems evaluation as well as results visualization is provided by COCO (COmparing Continuous Optimizers) platform [38].

Chapter 6
Results

In this chapter the results of experiments introduced in previous chapter 5 are presented. The results are divided into two sections.

6.1 Test problems

Firstly, results of the performance of the algorithms introduced in section 5.1 on the Test problems. Results are visualized in the form of graphs, which shows the dependence of the number of fitness function calls on the dimension of certain problem. These graphs are called scalability graphs. They shows the most important facets of algorithm's performance whatsmore, they provides a prediction regarding the performance on higher-dimensional problems.

Each data point is the median of successful runs. To display data over a very wide range of values in a compact way and to get clearer results, a base-10 logarithmic scale is used for both axis of graphs. It is also worth noting that the y-axis does not start at zero.

The Test results may be divided into three groups according to separability into separable problems (sphere, Levy, Rastrigin), block-separable problems (SoREB) and non-separable problems (Rosenbrock, OSoREB).

1) Separable problems

For all three separable problems both algorithms with full prior knowledge (DE_LT+, DE_MP+) perform very similarly. For the easiest function (sphere) is DE_LT+ a bit better, nevertheless for more difficult problems, the DE_MP+ achieves a little better results than DE_LT+ (for Levy and Rastrigin).

Since separable problems does not conatin any dependencies between variables, it is not surprising that there is almost no difference in performance of DE_LT_MIC and DE_LT+, because for separable problems, all possible linkage trees should be equally good. Therefore is does not matter what dependency matrix $\mathcal{P}$ DE_LT_MIC find and subsequently what linkage tree it build, the linkage tree would be just as good as the one built by DE_LT+.

The DE_MP_MIC is slightly worse than DE_LT+, DE_MP+ and DE_LT_MIC especially for higher dimensions. It is very hard to recognize separability by MIC because the DE selection operator alignt individuals with the fitness contours. Therefore DE_MP_MIC may determine some variables as dependent and build suboptimal MP-FOS which results in worse performace

Although both algorithms usign non-linearity check (DE_LT_NC, DE_MP_NC) correctly recognize the separability of the problem and build optimal FOS, they have achieved significantly worse results than other newly-introduced variants of DE. The difference is mainly caused by the fact that DE_LT_NC and DE_MP_NC uses fitness function evaluations to find dependencies, in contrast with DE_LT+, DE_MP+ and

DE_LT_MIC which also build optimal FOS but do not waste fitness function evaluations.

It also provides an explanation of why the DE_LT_NC outperforms DE_MP_NC. Since DE_LT_NC, DE_MP_NC build optimal FOS, the DE_LT_NC performes comparably to DE_LT+ within crossover. Similarly to DE_MP_NC and DE_MP+. And since DE_LT+ and DE_MP+ perform similarly, DE_LT_NC and DE_MP_NC use comparable number of fitness function evaluations within the crossover. It is a fact that LT-FOS has necessarily higher cardinality than MP-FOS for same dimension, therefore DE_LT_NC find the optimimum within less number of generations than DE_MP_NC. Since dependency matrix is built in each generation, DE_MP_NC uses more fitness function calls to find dependencies which results in worse performance then DE_LT_NC.

CMA-ES has achieved interesting results. For the sphere function, CMA-ES is a constant factor better than DE_LT+, DE_MP+, DE_LT_MIC, DE_MP_MIC, nevertheless as the difficulty of optimization of functions grows, the performance of CMA-ES is getting worse. For Levy function CMA-ES perform similarly to four mention algorithms and for the hardest function (Rastrigin) CMA-ES is significantly worse and achieves results comparable to DE_LT_NC and DE_MP_NC. But it is worth notting that CMA-ES scales better than NC variants of DE.

Finally, FMIN outperforms all algorithms for low dimensions of sphere, nevertheless scales very bad and get outperformed by all algorithms in higher dimensions. FMIN is unable to find optimum, even for low dimensions, of harder functions as Levy and Rastrigin.
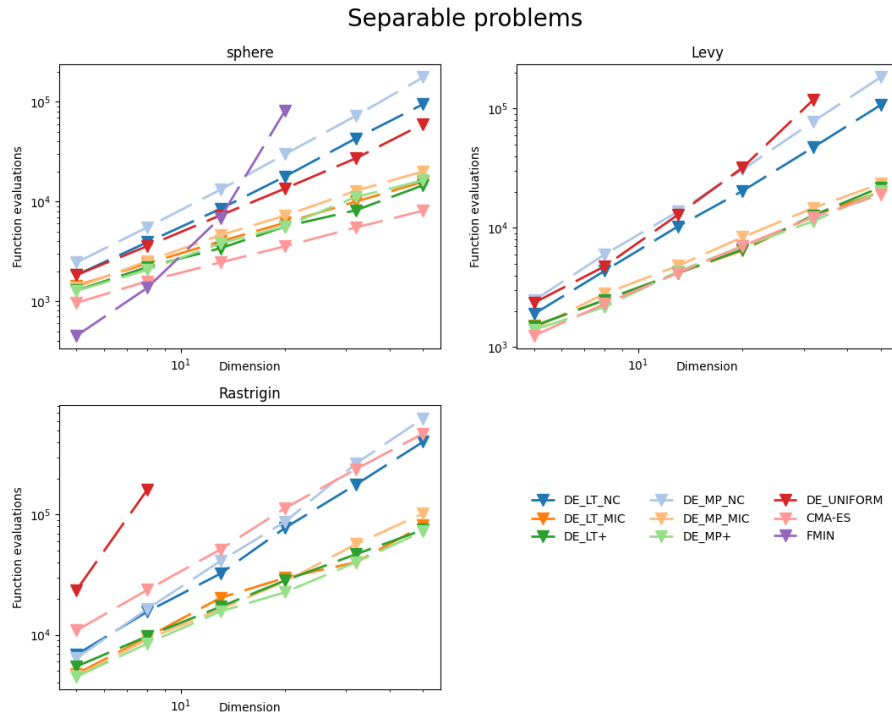

Figure 6.1. Scalability graphs of separable problems. Each point is the median of successful runs.

2) Block-separable problems

The separable problems are represented by SoREB function with variable size of block (2, 3, 4 , 5). It is worth notting the relationship between particular pair of DE variants which use same technique build dependency matrix.

Firstly, the DE_MP+ is, a constant factor better than DE_LT+. It is not surprising, since block-separable structure of a problem may be represented by MP FOS very well.

However for the second pair, which uses non-linear check, it can be seen that LT variant (DE_LT_NC) outperforms MP variant (DE_MP_NC). Although DE_MP_NC build same FOS as DE_MP+, which perfcetly capture the structure of the problem, it cannot achieve better results than DE_LT_NC. It points to the fact that, worse performance within the crossover is compensated by lower number of fitness function evaluations used to find dependencies.

Lastly, MIC variants (DE_LT_MIC, DE_MP_MIC) perform almost similarly for block size two and three. For $k = 4$ and $k = 5$ DE_LT_MIC achieve better result than DE_MP_MIC, nevertheless for the higher dimensions dependent blocks the dependent blocks are easier to recognize and the DE_MP_MIC outperforms DE_LT_MIC.

The relationship between MIC variants and NC variants is also worth notting. It can be seen that, for $k = 2$ both MIC variants perform similarly to DE_LT+ and better than NC variants, however as the size of block increases and more dependencies occur, MIC variants are getting outperformed by NC variants. It is causes by weaker ability of MIC to recognition dependencies, especially for low number of samples.

The original DE_UNIFORM performace shows up as the worst of all considered algorithms for block-separable problems. For bigger sizes of blocks the DE_UNIFORM is even worse.

CMA-ES perform similarly for all sizes of blocks in terms of the required number of evaluations. Since other algorithms need more evaluations for bigger blocks, CMA-ES outperforms all algorithm except DE_MP+ for $k = 5$. Nevertheless DE_LT+ and DE_MP+ outscale CMA-ES, which scalability is comparable to DE_LT_NC and DE_MP_NC.

Last considered algorithm FMIN shows best performace for small dimension, but the worst scalability of all algorithm and inability to find optimum for higher dimensions.

Lastly note that DE_LT+ and DE_LT_NC need on average more evaluations to find optimum for $D = 5$ than for $D = 5$ if $k = 5$, which shows certain limitations of LT when al variables are pairwise dependent.

3) Non-separable problems

Firstly, it is worth notting that no DE variant which build MP-FOS is able to find optimum of presented non-separable problems, even for small dimension. It is probably caused by overlapping dependency structure of both problems, which is impossible to model by MP-FOS.

DE_LT+ and DE_LT_MIC perform almost identicaly, the same trend may be observed in figure 6.1 representing results on separable functions. For Rosenbrock DE_LT+ and DE_LT_MIC scale better than DE_LT_NC, nevertheless for OSoREB DE_LT+ and DE_LT_MIC are only a constant factor better than DE_LT_NC. The slightly better relative scalability towards DE_LT+ and DE_LT_MIC corresponds to the results obtained in ?? because OSoREB is in a sense closer to the block-separable problem than Rosenbrock and the dependency structure of OSoREB may be captured better within LT-FOS than the structure of Rosenbrock. Therefore the impact of correct recognition of dependencies, which DE_LT_NC does, increases.

The same phenomenon may be seen in performance of DE_UNIFORM which is a constant better than DE_LT_NC for Rosenbrock, with relative small number of de-
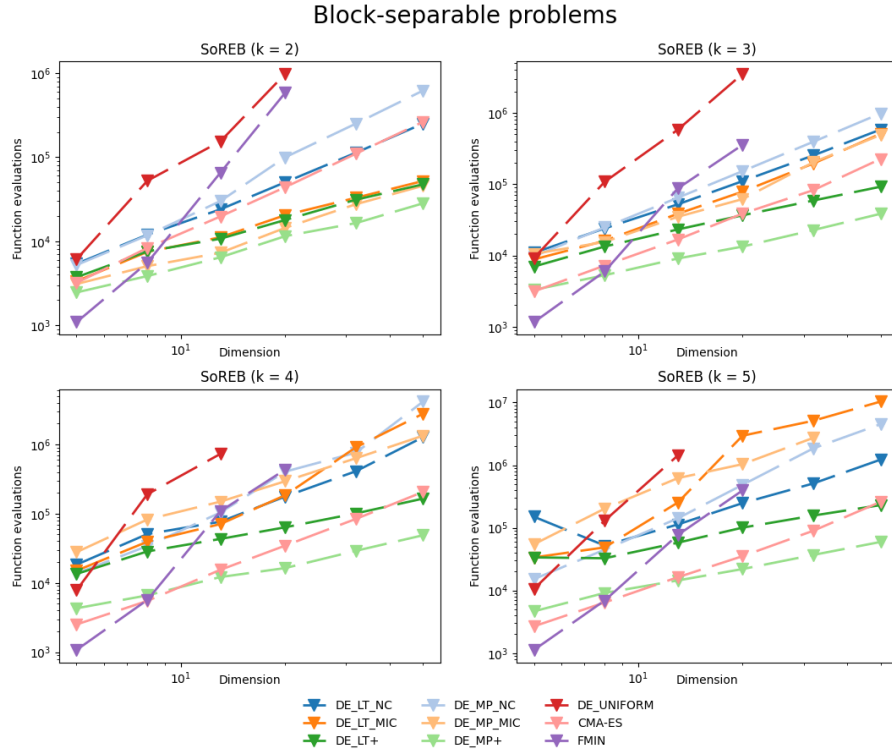
Figure 6.2. Scalability graphs of block-separable problems. Each point is the median of successful runs.

pendencies. Nevertheless as the number of dependencies increase from Rosenbrock to OSoREB, the scalability of DE_UNIFORM is getting worse and DE_UNIFORM is unable to find optimum of OSoREB in higher dimensions.

On the other hand, the opposite trend can be viewed on performance of CMA-ES, which outperforms all other algorithms nevertheless, for Rosenbrock the difference is less significant and DE_LT+ and DE_LT_MIC seemed to scale better than CMA-ES.

For non-separable problems, similarly as for other FMIN show up as the best algorithm for low dimansions, similarly to sphere function, but scale the worst. Therefore is almost useless for higher dimensions.

6.2 BBOB problems

Results on the BBOB problems are presented by graphs of Empirical cumulative distribution functions (ECDFs) [39]. These ECDFs show on the y-axis the proportion of cases for which the number of fitness function evaluations needed to find the optimum was smaller than the value given on the x-axis. For x-axis a base-10 logarithmic scale is used and the total number of fitness function evaluations is diveded by dimension. Each graph also shows performance of the best algorithm of BBOB workshop 2009 noted as *best 2009* .

All results in this section were obtained by COCO platform. Complete results contains 25 function for various dimensions (2, 3, 5, 10, 20, 40). Complete results are shown in Appendix A. In this section, only selected functions of dimensions 5 and 20
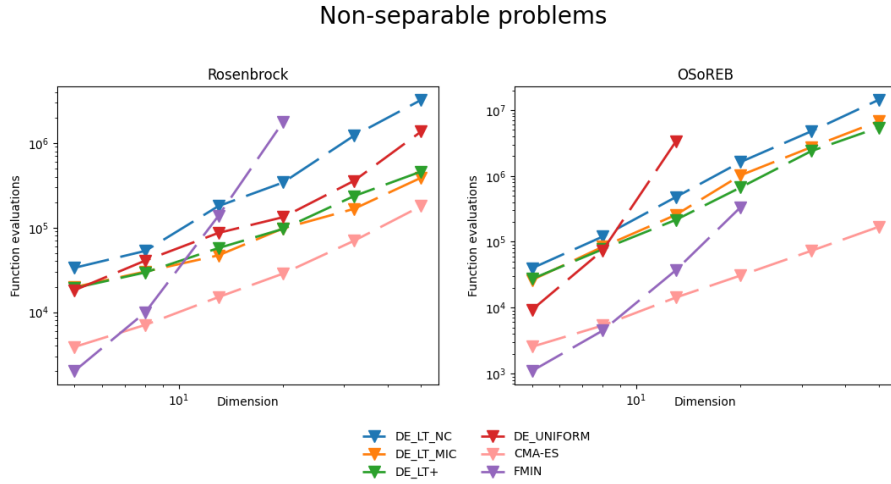
Non-separable problems

Figure 6.3. Scalability graphs of non-separable problems. Each point is the median of successful runs.

occurs. These functions were chosen to represent characteristic trend seen for more functions.

Firstly, the set of functions on which all DE variants perform similarly but worse than CMA-ES and best 2009. This set is represented by so called Ellipsoid separable function on figure 6.4.
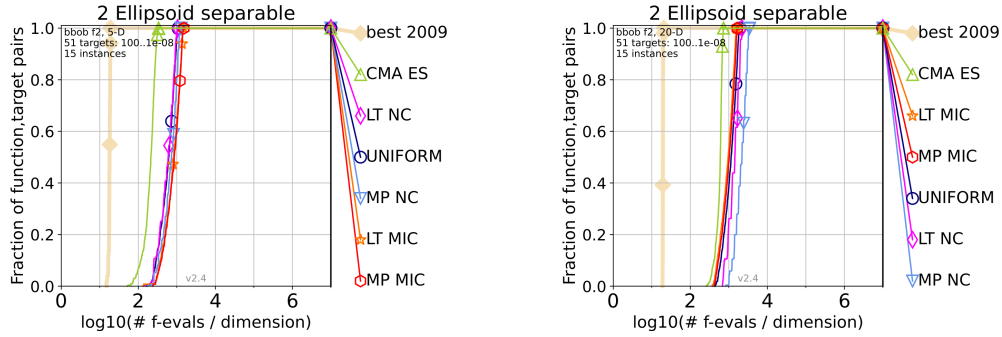

Figure 6.4. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Ellipsoid separable function for dimensions five and twenty.

Secondly, for a number of functions, MP varints of DE are outperform by other DE variants, especially for higher dimensions, nevertheless LT and UNIFORM variants are outperformed by CMA-ES and by best 2009. The example of such a function may be seen on figure 6.5.

The third observed trend within BBOB functions is significantly worse performace of DE variants againts CMA-ES and best 2009. All DE variants are unable to find global optimum for majority of these functions. Note that these are functions are mainly Multi-modal functions with weak global structure. This trend is caprured on figure 6.6.
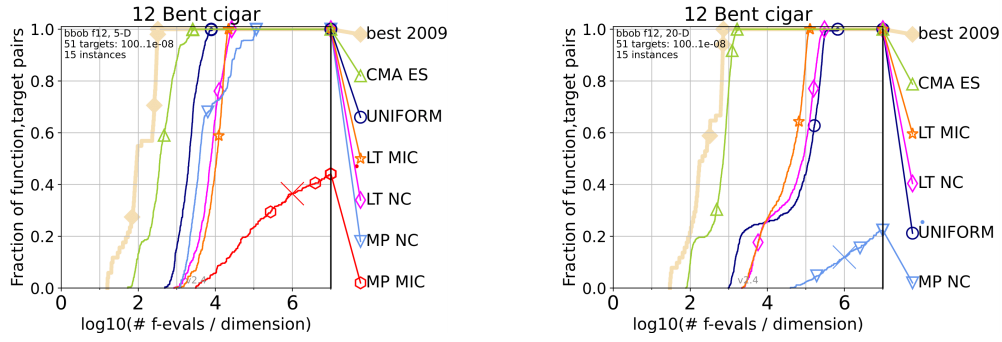
23

Figure 6.5. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Bent function for dimensions five and twenty.
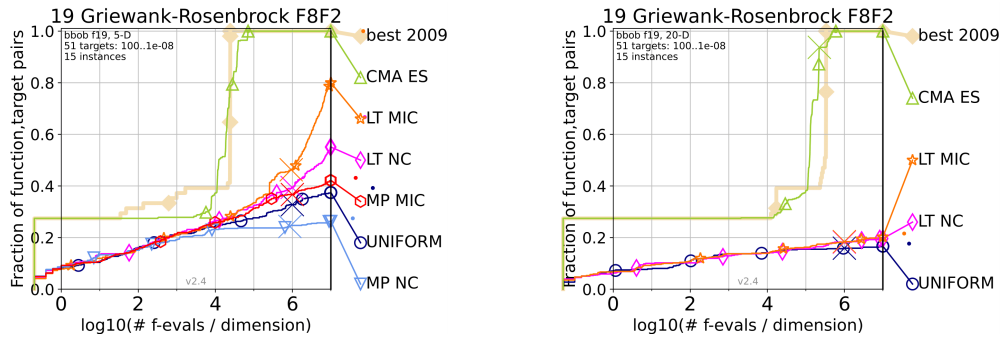

Figure 6.6. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Ellipsoid separable function for dimensions five and twenty.

Chapter 7
Conclusion

The main goal of this work was to propose a crossover operatior with dependency detection for DE.

In chapter 3 were introduced two representations of the dependency structure, linkage tree (LT) which contains itself multiple levels of dependencies, from univariate level to complete dependency, and marginal product (MP) which contains every problem variable exactly once.

In chapter 4 two approaches that can find pairwise dependencies between variables were introduced. First, fitness-based approach called non-linearity check (NC) determining the possible dependence between a pair of variables according to the fitnes values. Second approach called maximal information coefficient (MIC) indentifies dependencies based on the distribution of the population.

Subsequently six new variants of DE differing in the crossover operator were proposed in section 5.1. Except four regular variants using above-mention methods DE_LT_NC, DE_MP_NC, DE_LT_MIC, DE_MP_MIC, two artifical variants with full prior knowledge of dependency structure were also proposed.

According to results presented in section 6.1, newly proposed methods show to achieve a fair scalability, all of them outscale the original DE (DE_UNIFORM) for almost all test problems, independent of the dependency structure. They also showed better scalability than FMIN algorithm and for some problems even better scalability than state-of-the-art evolutionary algorithm CMA-ES. So usege any of four propsoed regular variants of DE instead of original one would bring better performace.

According to the results from previous chapter it may be concluded that linkage tree representation, thanks to its robustness, is seemed to be better choice than marginal product representation.

It is worth noting that the results obtained by our NC methods should be compared to the MIC results with careful consideration. For instance, while NC methods can easy recognize separability of two variables, for MIC ones the same task is very hard because the DE selection operator alignt individuals with the fitness contours. On the other hand, MIC variants, in contrast with NC ones, do not need any fitness function evaluations to find dependencies. The disadvantage of MIC is that DE usually take advantage of relative small populations and since MIC is a statistical method, it achieves better results with more samples. On the other hand MIC would probably be more noise resistant than NC. However, no experiments to substantiate this assumption have not been presented so far, so there is a space for future work to prove or refute this hypothesis.

Although all four regular variants of DE enhance the performance of the original DE, they have some limitations which would be be reduced by further adjustments.

For instance, *incremental dependency updating* described in [16] would propably significantly reduce the amount of fitness function evaluations used by NC methods to find dependencies and decrease the difference in performance between NC methods and methods with full prior knowledge.

The last unaswered question is, how to deal with problems containing overlapping sub-components. Not LT nor MP are able to represent these types of structures very well, whatsmore the optimal linkage structure of these problems is unknown. So to find the optimal structure for these problems would be also very interesting.

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] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] David Reshef, Yakir Reshef, Hilary Finucane, Sharon Grossman, Gilean McVean, Peter Turnbaugh, Eric Lander, Michael Mitzenmacher, and Pardis Sabeti. Detecting Novel Associations in Large Data Sets. *Science (New York, N.Y.).* 2011, 334 1518-24. DOI 10.1126/science.1205438.

[18] Yakir A. Reshef, David N. Reshef, Hilary K. Finucane, Pardis C. Sabeti, and Michael Mitzenmacher. Measuring Dependence Powerfully and Equitably. *Journal of Machine Learning Research.* 2016, 17 (211), 1-63.

[19] R. Steuer, J. Kurths, C. O. Daub, J. Weise, and J. Selbig. "The mutual information: Detecting and evaluating dependencies between variables". *Bioinformatics.* 18

[20] Yakir A. Reshef, David N. Reshef, Hilary K. Finucane, Pardis C. Sabeti, and Michael M. Mitzenmacher. *Measuring dependence powerfully and equitably.* 2016.

[21] Alexander Luedtke, and Linh Tran. *The Generalized Mean Information Coefficient.* 2013.

[22] Davide Albanese, Samantha Riccadonna, Claudio Donati, and Pietro Franceschi. "A practical tool for maximal information coefficient analysis". *GigaScience.* 2018, 7 (4), DOI 10.1093/gigascience/giy032. giy032.

[23] Davide Albanese, Samantha Riccadonna, Claudio Donati, and Pietro Franceschi. *minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers.* Bioinformatics. 2012.

[24] Anne Auger, Steffen Finck, Nikolaus Hansen, and Raymond Ros. *BBOB 2009: Comparison Tables of All Algorithms on All Noiseless Functions.* . INRIA.

[25] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. *PSO Facing Non-Separable and Ill-Conditioned Problems.* . INRIA.

[26] Nikolaus Hansen. The CMA Evolution Strategy: A Comparing Review. *Towards a new evolutionary computation. Studies in Fuzziness and Soft Computing.* 2007, 192 75-102. DOI 10.1007/3-540-32494-1_4.

[27] Nikolaus Hansen. *CMA-ES written in ANSI C (yet fairly object-oriented).* `https://github.com/CMA-ES/c-cmaes`. 2014.

[28] J. Nelder, and R. Mead. A Simplex Method for Function Minimization. *Comput. J..* 1965, 7 308-313.

[29] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, .Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods.* 2020, 17 261–272. DOI 10.1038/s41592-019-0686-2.

[30] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems.* Ph.D. Thesis, The University of Michigan. 1975.

[31] Hartmut Pohlheim. *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB.* 1997.
`http://www.geatbx.com/ea_matlab.html`.

[32] L.A. Rastrigin. Systems of Extreme Control. *Nauka*. 1974,

[33] G. Rudolph. *Globale Optimierung mit parallelen Evolutionsstrategien*. 1990.

[34] H.H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*. 1960, 3 175-184. DOI 10.1093/comjnl/3.3.175.

[35] Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*. 2013, 48 DOI 10.1007/s00158-013-0919-4.

[36] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noisy Functions Definitions*. .

[37] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup. 2009,

[38] Nikolaus Hansen, Dimo Brockhoff, Olaf Mersmann, Tea Tusar, Dejan Tusar, Oussaim Ait ElHara, Phillipe R. Sampaio, Asma Atamna, Konstantinos Varelas, Umut Batu, Duc Manh Nguyen, Filip Matzner, and Auger Anne. *COmparing Continuous Optimizers: numbbo/COCO on Github*. https://github.com/numbbo/coco. 2019.

[39] G. R. Shorack, and J. A. Wellner. Empirical processes with applications to statistics of Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics. *John Wiley & Sons, New York*. 1986,

Appendix A
Complete BBOB results