



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Bachelor's Thesis

Differential Evolution Crossover with Dependency Detection

Vojtěch Voráček

May 2021

Supervisor: Ing. Petr Pošík, Ph.D.



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 závislosti

Guidelines:

Differential Evolution (DE) is a successful 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.
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
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 also belong to my family, current and future, and 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 spojitě black-box optimalizační problémy. Originální verze diferenciální evoluce použije náhodný uniformní operátor křížení, který nebere v potaz potenciální 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 navrženy 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 navržené algoritmy dosahují výrazně lepšího výkonu a škálovatelnosti než originální diferenciální evoluce ve smyslu potřebného počtu vyhodnocení účelové funkce k nalezení globálního optima. Některé 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 one of the best evolutionary algorithms for continuous black-box optimization problems. The original version of differential evolution uses a random uniform crossover operator, which does not take possible dependencies between parts of the solution into account and may disrupt these linkages.

This work aims to propose a new crossover operator for differential evolution more suitable for the class of problems containing dependent solution components.

This work presents two options to find dependencies between problem variables and two possibilities to model the linkage structure. Finally, with the 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 performance 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 /

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	3
2.1.6 Mutation operator	3
2.1.7 Replacement strategy	3
2.1.8 Initialization	3
2.1.9 Termination condition	4
2.2 General scheme	4
2.3 Differential evolution	4
2.3.1 Representation	5
2.3.2 Mutation	5
2.3.3 Crossover	5
2.3.4 Replacement strategy	5
3 Linkage information modeling	7
3.1 Family Of Subsets	7
3.2 Linkage tree	7
3.3 Marginal product	9
4 Identification of the linkage structure	10
4.1 Fitness-based method	10
4.2 Distribution-based method	11
5 Experiments	13
5.1 Algorithms	13
5.1.1 Differential evolution variants	14
5.1.2 Other algorithms	15
5.2 Test problems	15
5.2.1 Sphere	15
5.2.2 Levy	16
5.2.3 Rastrigin	16
5.2.4 Rosenbrock	16
5.2.5 SoREB	17
5.2.6 OSoREB	17
5.3 Black Box Optimization Benchmarking problems	17
5.4 Setup	18
5.4.1 Test problems specifics ..	18
5.4.2 BBOB problems specifics	18
6 Results	19
6.1 Test problems	19
6.1.1 Separable problems	19
6.1.2 Block-separable problems	20
6.1.3 Non-separable problems ..	21
6.2 BBOB problems	23
7 Conclusion	25
References	27
A Abbreviations	31
B Complete BBOB results	32
C Implementation and contents of CD	33

Tables / Figures

5.1. Differential evolution variants .	14
5.2. MICE grid resolution B	15
2.1. Evolutionary algorithm pseudo-code	4
2.2. Differential evolution pseudo-code	6
3.1. Linkage tree	8
3.2. Differential evolution with known FOS	9
5.1. Differential evolution with dependency detection	13
6.1. Separable problems graphs	21
6.2. Block-separable problems graphs	22
6.3. Non-separable problems graphs	23
6.4. Ellipsoid separable function graphs	24
6.5. Bent function graphs.....	24
6.6. Griewank function graphs	24

Chapter 1

Introduction

1.1 Motivation

Striving for the best solution to a particular problem is an essential part of many fields of human interest. The process of finding the best solution according to some criteria is called optimization.

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. However, it is 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 that 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 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 weakness mentioned above of DE.

Chapter 2

Evolutionary algorithms

Evolutionary algorithms (EAs) [5–7] is a set of stochastic metaheuristic optimization algorithms inspired by Darwin’s theory of evolution by natural selection [8]. The theory describes the process of developing organisms over time as a result of changes in heritable traits. Changes that allow an organism to better adapt to its environment will help it survive and reproduce more offspring. This phenomenon is commonly called “*Survival of the fittest*”, first used by Herbert Spencer [9].

In analogy to the natural environment, EA maintains a *population* of potential solutions (*individuals*) for the given problem. The 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 modifying an individual (mutation). The algorithm runs until a candidate solution with sufficient quality is found, or a certain user-defined 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 a working algorithm.

2.1.1 Representation of individuals

Each individual is encoded in so-called *chromosomes*. The representation of chromosomes is called *genotype*. *Phenotype* refers to the interpretation of the genotype, in other words, how the objective function treats the genotype. The Representation also involves genotype-phenotype mapping. For instance, given an optimization problem on integers, if one decides to represent them by their binary code, 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. The objective function defines how quality an individual is with respect to the problem in consideration. Technically, it is a function that takes an individual as input and produces the measure of the 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 example mentioned above, if the problem was to minimize 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 within an evolutionary algorithm means a set of individuals. A population can be specified only by setting the population size. In other words, how many individuals are in the population. This parameter is usually specified by the user.

■ 2.1.4 Parent selection

During each *generation* (one iteration of an algorithm), a specific part of the population is selected to breed offspring. The choice is made similar to natural selection. In other words, fitter individuals are preferred. Nevertheless, low quality individuals are given a small but positive chance 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

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

■ 2.1.6 Mutation operator

The *mutation* is a unary genetic operator that changes parts of an individual's chromosome, typically randomly. In mutation, the mutated individual may change entirely from the original individual. The mutation is used to maintain and introduce diversity in the population.

■ 2.1.7 Replacement strategy

Replacement strategy defines which individuals survive and become members of the subsequent generation. Typically, the decision is based on the quality of individuals, preferring 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

The *Initialization* procedure generates a defined number of individuals of the given representation, thereby creating the initial population. Initialization is often done randomly due to a 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 never be satisfied. Therefore, this condition is extended with the condition that 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 the above-mentioned components, the evolutionary algorithm is formed. This section describes the way the EA works as a whole.

Algorithm 1: Evolutionary algorithm

Result: best individual in $Population(t)$
 $t \leftarrow 0$;
 $Population(t) \leftarrow initialization()$;
 $fitnessFunctionEvaluation(Population(t))$;
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 fitness function subsequently evaluates the population. Then starts a generational process.

The generational process is repeated until a terminal condition is not satisfied. The generational process starts with parent selection, usually based on 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. A fitness function subsequently evaluates offsprings, and the generational process ends by creating a new population. Creating a new population is done 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 the algorithm.

2.3 Differential evolution

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

DE [7] 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 *mutant*

vector. The mutant 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 exclusive. The offspring \vec{o}_i is then created by crossing over the mutant vector with \vec{x}_i .

Note that the impact of the mutant 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, the mutant vector will be small if the population is condensed in a particular region. Thus DE belongs to the family of adaptive mutation algorithms.

Lastly, the newly created offspring is compared to his parent using the greedy criteria. If the offspring is better than his parent, it replaces 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:

$$\vec{x}_i = \{x_{i,0}, x_{i,1}, \dots, x_{i,D-1}\}, \forall j : x_{i,j} \in \mathbb{R},$$

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

$$P = \{\vec{x}_0, \vec{x}_1, \dots, \vec{x}_{NP-1}\}, NP \geq 4,$$

where NP is the size of population.

2.3.2 Mutation

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

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

with random, mutually exclusive indexes $r_1, r_2, r_3 \in \{0, 1, \dots, NP - 1\}$, 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 the mutation, the mutant vector \vec{m}_i undergoes a crossover with its relevant individual \vec{x}_i to generate the offspring \vec{o}_i . Standard DE us binomial crossover, where offspring is generated as follows:

$$\vec{o}_i = \begin{cases} \vec{m}_i & \text{if } i = R \vee \text{rand}(0, 1) < CR, \\ \vec{x}_i & \text{otherwise.} \end{cases} \quad (1)$$

where: $R \in [0, D]$ is a random integer, $\text{rand}(0, 1)$ represents a random number between zero and one, and CR denotes the probability of crossover. Since \vec{x}_i is the parent of \vec{o}_i , it can be seen that each individual from the population generates offspring. In other words, the parent selection chooses all individuals from the population.

2.3.4 Replacement strategy

To decide which individuals become members of the 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. Formally, the new population is determined as follows:

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

Algorithm 2: Differential evolution**Result:** best individual found so farGenerate 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**

$$\vec{o}_{i,d} = \begin{cases} \vec{x}_{r_1,d} + F * (\vec{x}_{r_2,d} - \vec{x}_{r_3,d}), & \text{if } d = R \text{ or } \text{rand}(0, 1) < CR \\ \vec{x}_{i,d}, & \text{otherwise} \end{cases}$$

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 noting that DE, as was described in the previous chapter, uses random, uniform crossover. The crossover has no assumptions about the structure of the optimized function. Specially DE does not take possible dependencies between specific parts of the solution into account.

However, there exists a whole class of problems with dependent solution components. DE using uniform crossover does not take possible dependencies into consideration and often disrupts linkage between strongly connected components.

The aim of this work is to propose a new crossover operator capable of finding dependencies and taking them into account when generating new offsprings. This chapter proposes two possible representations of dependency structure and introduces an adapted crossover operator.

3.1 Family Of Subsets

Both representations of dependency structure are based on the *Family Of Subsets* (FOS) [10]. FOS is a way how to model linkage information that describes presumed dependencies between variables. FOS $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots\}$ represents a subset of powerset $\mathcal{P}(\mathcal{I})$ of \mathcal{I} , where $\mathcal{I} = \{0, 1, \dots, D - 1\}$ stands for 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].

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 contained in its parent \vec{x}_i , then these values 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 accepted if it has better or equal fitness value than the original \vec{x}_i . Changes in DE pseudocode are captured in figure 3.2.

3.2 Linkage tree

Many FOS structures exist, and any of them can be used to model linkage structure. However, this work focuses on two of them. The first of them is the *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 the following chapter(4).

The linkage tree is a tree with D leaf nodes and $D - 1$ inner nodes, where D is the number of problem variables. Each node of the LT represents a specific block of variables \mathcal{F}_j . The key property of the LT is that each \mathcal{F}_j , which contains 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}_j , 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 procedure proceeds bottom-up. Therefore the tree is initialized with these univariate blocks as leaves. In each step, a new node is created by merging two nodes of the tree 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 merging process stops when no more merges are possible. In other words, the root node has been created. Due to the way the procedure works, the root node has to be a set of all problem variables. The tree itself contains multiple levels of dependencies. From univariate level at the height of zero to complete dependency between all variables at a depth of zero. [11]

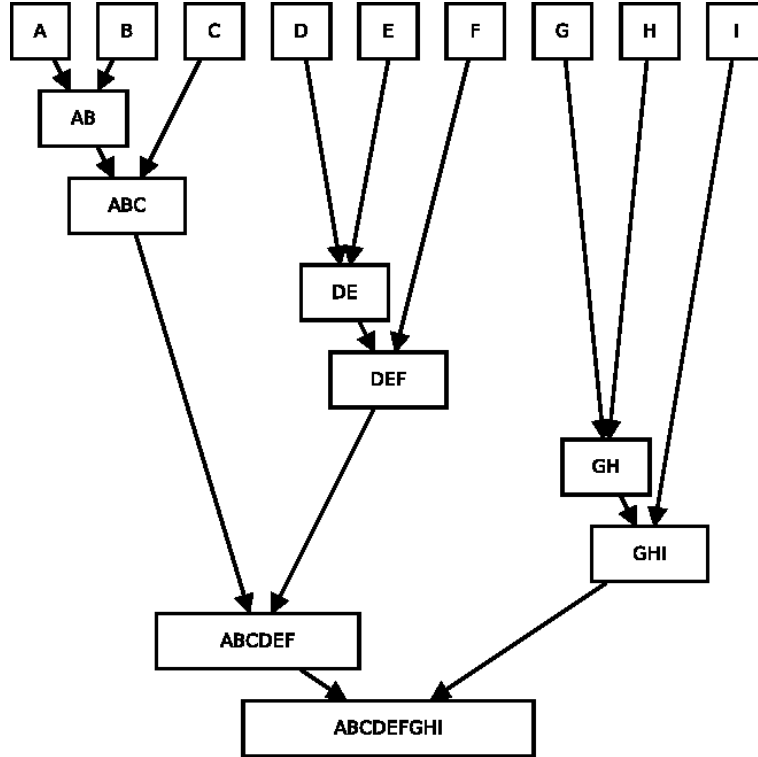


Figure 3.1. Example Linkage tree [12]

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

3.3 Marginal product

The 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 $\mathcal{F} = \{\{0\}, \{1\}, \dots, \{D-1\}\}$, where D is number of problem variables. On the contrary, when all variables are considered mutually dependent, MP is called compact FOS.

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

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

where $G \geq 0$ is user-selected factor defining the degree of strength of univariate blocks.

The MP building procedure starts by initializing MP \mathcal{F} as univariate 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 as the most dependent by the given distance measure \mathcal{M} . 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 $\theta_1 > 0, \theta_2 \in [1, D]$ and factor $K \in (0, 1]$ are defined by user. Then \mathcal{F}_n is inserted into 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 become the compact FOS.

The DE using the MP FOS structure (DE_MP) build the MP in every generation. After building the MP, DE_MP 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. Pseudocode of the differential evolution with known FOS

Chapter 4

Identification of the linkage structure

In the 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 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]:

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

where X_i and X_j are sets of variables and $p_{u,v}$ is an 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 symmetric and positive semi-definite matrix. The dependency matrix gives the dependency between each pair of variables. The element $p_{i,j}$ denotes the pairwise dependency strength between i -th and j -th problem variables. Consequently \mathcal{P} contains zeros on diagonal i.e. $\forall i = 0, 1, \dots, D-1 : p_{i,i} = 0$.

There are several methods how to construct 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), defines whether two variables interact directly based on fitness values. The method works under the assumption that non-linear interactions may exist only between dependent variables. It classifies a pair of variables, either separable or non-separable, by comparing the difference in overall fitness while making the exact same change for a particular pair of chromosomes 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. The set of indices of m best individuals in the population is denoted as C .

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

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

Note that $d_{i,j,k}$ as well as $p_{j,k}$ lie within $[0, 1]$ with zero indicating independent variables.

4.2 Distribution-based method

The second method to construct the dependency matrix \mathcal{P} is called the *maximal information coefficient* (MIC) [17]. There exist more methods used to identify dependencies between a pair of variables based on the distribution of the population [13, 16]. However, MIC achieved better accuracy 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 possible to draw a grid on the scatterplot of the two variables that partitions the data to encapsulate that relationship.

In order 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] achievable 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 the maximum of those 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 MIC has achieved good results in various comparisons. However, a big limitation of MIC is its high computational cost. Therefore several algorithms for approximating the MIC have been published. [17, 20–21]. In this work MICE minepy implementation [22–23] is used.

In this work, MICE is not calculated from the whole population, but only subset C of all individuals from the current 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 the solution. In chapter 3, two possible representations of dependency structure were introduced assuming known pairwise dependencies, and in chapter 4, two ways to find pairwise dependencies and thereby build the dependency matrix \mathcal{P} were presented.

Based on those methods, it is possible to propose adjusted DE with dependency detection. The adjusted version differs from the original in two factors. Firstly, in every generation, the dependency matrix \mathcal{P} and FOS structure based on it is built. Secondly, the 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 the performance of various types of DE with dependency detection differing in creating the matrix \mathcal{P} , or in the building of 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 are compared. The original DE, as was introduced in 2.3 section. (DE_UNIFORM). The remaining six variants of DE are divided into three pairs according to how they create the distance matrix \mathcal{P} . Within each pair, the first variant uses the linkage tree (LT) and the second the marginal product (MP). The first pair uses the non-linearity 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 pairs (DE_LT+ and DE_MP+).

	Non-linearity 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.

It is important to note that all newly proposed variants of DE create \mathcal{P} and build the FOS structure in every second generation instead of every generation in order to speed up computing.

All above-mentioned DE variants share 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:

$$G = \begin{cases} 1 & \text{for DE_MP+}, \\ 2 & \text{otherwise.} \end{cases}$$

- 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 table 5.2 (rounded to the nearest integer in an upward direction).
- Th MICE parameter c , which determines how many more clumps there will be than columns in every partition, was set default value 15 [22]

All values mentioned above were found empirically unless otherwise stated.

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

The 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 optimization. [24–26] In this work the Hanses’s implementation of CMA-ES with default parameters is used. [27]

The last considered algorithm is the Nelder-Mead simplex algorithm [28], the optimization algorithm, which is not an evolutionary algorithm. Nevertheless, it uses only function values to find the optimum. Therefore may be used for black box 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. These six optimization problems to minimize are considered to study the impact of various types of linkage learning on the performance 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:

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

where f_0, f_1, \dots, 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:

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

It can be seen that standard DE, which optimizes each dimension independently, would be suitable for optimizing additively separable functions.

5.2.1 Sphere

The first benchmark function is the sphere function, also known as De Jong F1 [30]. It is presumably the easiest continuous 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:

$$\begin{aligned} f_{sphere}(\vec{x}_{min}) &= 0 \\ \vec{x}_{min} &= [0, 0, \dots, 0] \end{aligned}$$

■ 5.2.2 Levy

The second considered benchmark problem is the Levy function [31]. Like the sphere, it is a separable function with one local minimum. Nevertheless, the 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 $i = 0, 1, \dots, D - 1$.

Global minimum:

$$\begin{aligned} f_{Levy}(\vec{x}_{min}) &= 0 \\ \vec{x}_{min} &= [1, 1, \dots, 1] \end{aligned}$$

■ 5.2.3 Rastrigin

The Rastrigin function [32–33] is the third benchmark problem. Like the previous functions, this one is also separable. It is a 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:

$$\begin{aligned} f_{Rastrigin}(\vec{x}_{min}) &= 0 \\ \vec{x}_{min} &= [0, 0, \dots, 0] \end{aligned}$$

■ 5.2.4 Rosenbrock

The Rosenbrock function [34], also known as the Banana function, is the first considered non-separable function because it has overlapping dependencies. Each pair of consecutive variables is dependent. The Rosenbrock function contains a 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:

$$\begin{aligned} f_{Rosenbrock}(\vec{x}_{min}) &= 0 \\ \vec{x}_{min} &= [1, 1, \dots, 1] \end{aligned}$$

5.2.5 SoREB

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

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

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

R_{θ} defines the rotation of a vector around the origin by the angle of θ , and k is the size of block. Rotated blocks of variables that enter to $f_{\text{ellipsoid}}$ as an input creates strongly connected components. Variables within the block have strong dependencies but are entirely 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 $\theta = \pi/8$ is used.

Global minimum:

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

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

5.2.6 OSoREB

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

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

Global minimum:

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

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

5.3 Black Box Optimization Benchmarking problems

Black Box Optimization Benchmarking (BBOB) problems are the second considered set of benchmark problems. It is a set of 24 noise-less real-parameter single-objective benchmark functions defined in [36] as Real-Parameter Black-Box Optimization Benchmarking 2009 Noiseless Functions. Those functions were used for the BBOB workshop 2009. The BBOB problems were selected with the intention to evaluate the performance of algorithms with regard to standard difficulties that occur 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 minimized.

It is important to note that since BBOB problems cover a wide range of possible optimization problems, proposed DE variants may be unsuitable for some of them. BBOB problems consist of separable and non-separable ones.

5.4 Setup

All experimental results described in this work measures the first time a global optimum was hit. In other words, the number of fitness function calls needed to reach the small enough neighborhood 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, twenty-five 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 algorithm's performance is considered successful. It is determined by starting from the smallest possible population and letting the algorithm runs 25 times. If the performance has not been successful, the population size for the next trial will increase by s . This procedure is repeated until the successful population size is found or a population size reaches the upper limit T . If the successful population size is found, the optimal population size is searched for by performing a bisection search between the current population size and the previous size. Otherwise, the algorithm is considered unable to optimize a certain problem and dimension. Parameters s and T are set as follows:

$$s = \begin{cases} 10 & \text{for DE_LT_MIC and DE_MP_MIC,} \\ 4 & \text{for otherwise.} \end{cases}$$

$$T = \begin{cases} 50 + 6 \cdot D & \text{for DE_LT_MIC and DE_MP_MIC,} \\ 50 & \text{for otherwise.} \end{cases}$$

5.4.2 BBOB problems specifics

The setup for BBOB problems is partially determined by the 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 function 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, algorithms DE_LT+ and DE_MP+ are not considered.

BBOB problems evaluation and results visualization are provided by COCO (Comparing Continuous Optimizers) platform [38].

Chapter 6

Results

In this chapter, the results of experiments introduced in the previous chapter 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 show the dependence of the number of fitness function calls on the dimension of a certain problem. These graphs are called scalability graphs. They show the most important facets of the algorithm's performance. Moreover, they provide 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 axes 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).

6.1.1 Separable problems

The results on separable problems are shown in figure 6.1.

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

Since separable problems do not contain any dependencies between variables, it is not surprising that there is almost no difference in DE_LT_MIC and DE_LT+ performance because, for separable problems, all possible linkage trees should be equally good. Therefore it does not matter what dependency matrix \mathcal{P} DE_LT_MIC finds and subsequently what linkage tree it builds. 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 aligns individuals with the fitness contours. Therefore DE_MP_MIC may determine some variables as dependent and build suboptimal MP-FOS, which results in worse performance.

Although both algorithms using 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 performs comparably to DE_LT+ within the crossover. Similar to DE_MP_NC and DE_MP+. Moreover, since DE_LT+ and DE_MP+ perform similarly, DE_LT_NC and DE_MP_NC use a comparable number of fitness function evaluations within the crossover. It is a fact that LT FOS has a necessarily higher cardinality than MP FOS for the same dimension. Therefore DE_LT_NC finds the optimum within a fewer number of generations than DE_MP_NC. Since the dependency matrix is built in every second generation, DE_MP_NC uses more fitness function calls to find dependencies, resulting in worse performance than 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 the Levy function, CMA-ES performs similarly to the 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. However, it is worth noting that CMA-ES scales better than NC variants of DE.

Finally, FMIN outperforms all algorithms for low dimensions of the sphere. Nevertheless, it scales very badly and gets outperformed by all algorithms in higher dimensions. FMIN is unable to find optimum, even for low dimensions, of harder functions as Levy and Rastrigin.

6.1.2 Block-separable problems

The results on block-separable problems are shown in figure 6.2.

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

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

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

Lastly, MIC variants (DE_LT_MIC, DE_MP_MIC) perform almost similarly for block sizes two and three. For $k = 4$ and $k = 5$, DE_LT_MIC achieves better results than DE_MP_MIC for lower dimensions, nevertheless, for the higher dimensions, 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 noting. 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 caused by the weaker ability of MIC to recognition dependencies.

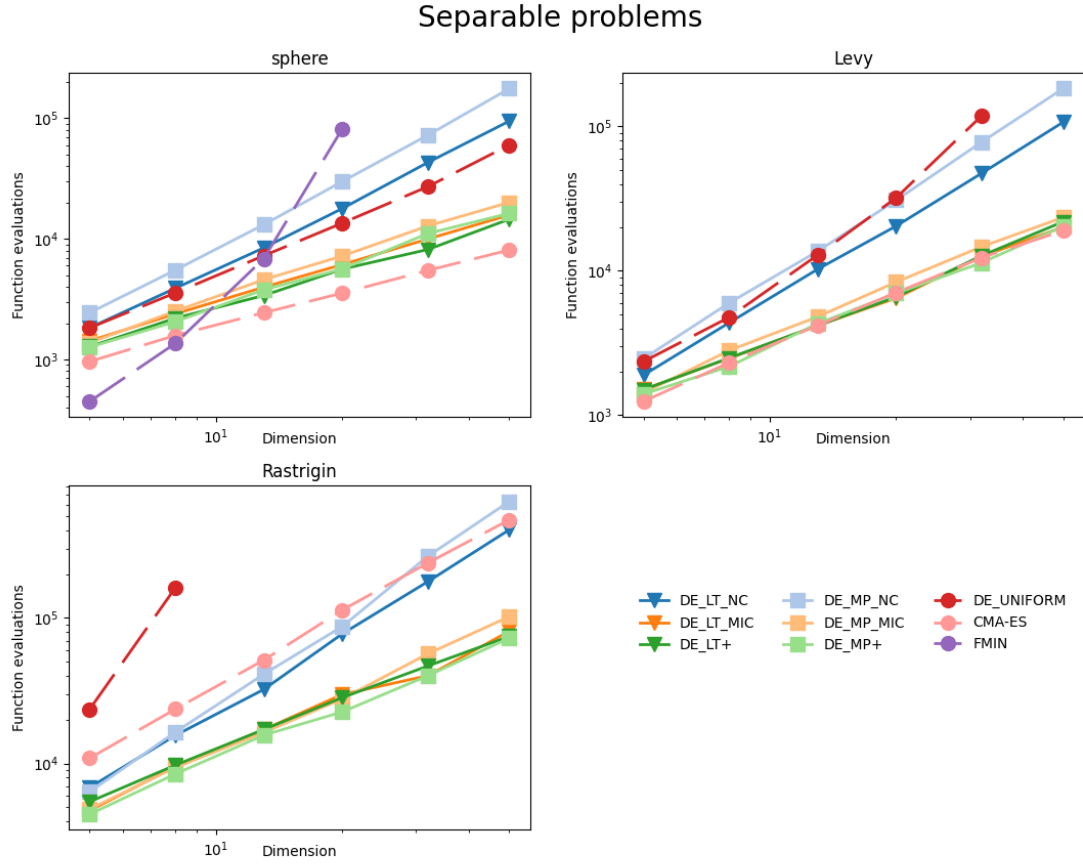


Figure 6.1. Scalability graphs of separable problems. Each point is the median of successful runs.

The original DE.UNIFORM performance shows up as the worst of all considered algorithms for block-separable problems. For bigger sizes of block the DE.UNIFORM is even worse.

CMA-ES perform similarly for all SoREB variants, regardless of the size of blocks, in terms of the required number of evaluations. Since other algorithms need more evaluations for SoREB with bigger blocks, CMA-ES outperforms all algorithms 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 results 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 $D = 8$ if $k = 5$, which shows certain limitations of LT when all variables are pairwise dependent.

6.1.3 Non-separable problems

The results on non-separable problems are shown in figure 6.3.

Firstly, it is worth noting that no DE variant that builds MP-FOS is capable of finding the optimum of presented non-separable problems, even for a small dimension.

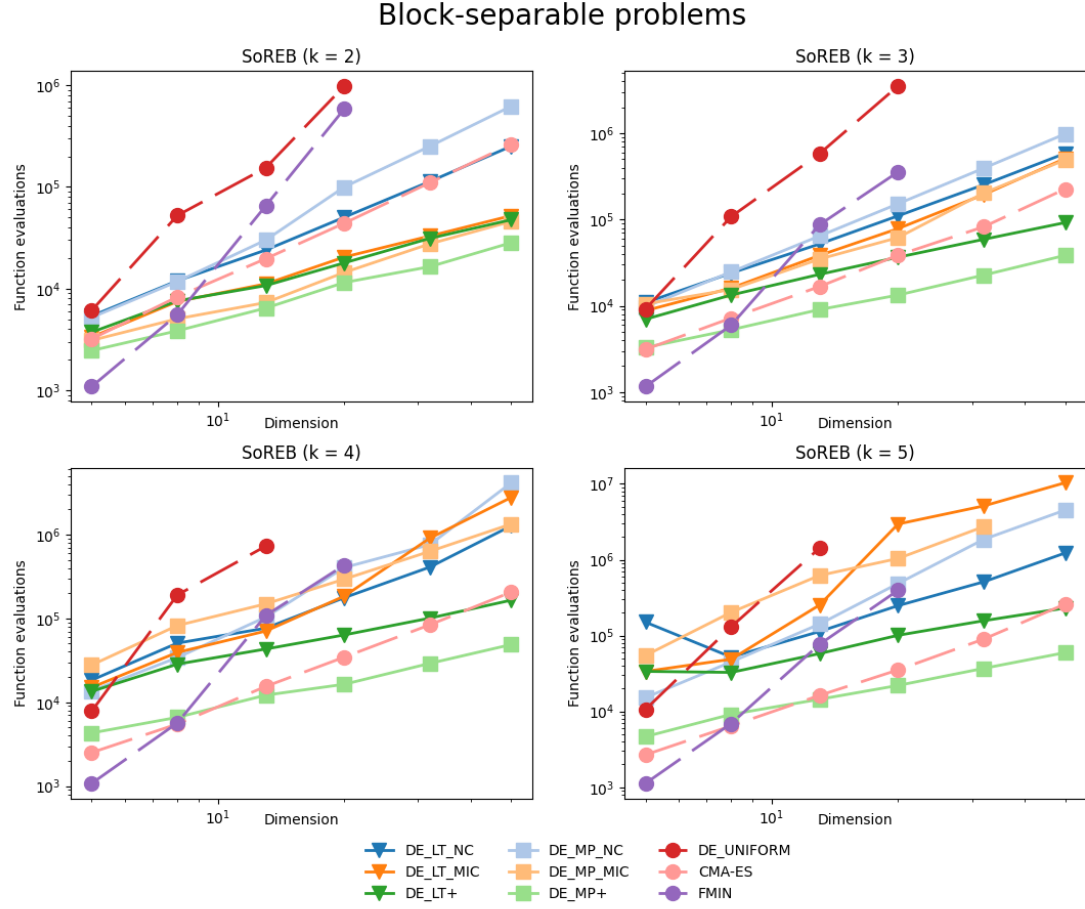


Figure 6.2. Scalability graphs of block-separable problems. Each point is the median of successful runs.

It is probably caused by the overlapping dependency structure of both problems, which is impossible to model by MP-FOS.

DE_LT+ and DE_LT_MIC perform almost identically. The same trend may be observed in figure 6.1. Representing results on separable functions. For the 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 of DE_LT_NC on OSoREB towards DE_LT+ and DE_LT_MIC corresponds to the results obtained in 6.2 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 the performance of DE_UNIFORM, which is a constant better than DE_LT_NC for Rosenbrock, the function with a relatively small number of dependencies. Nevertheless, as the number of dependencies increases from Rosenbrock to OSoREB, the scalability of DE_UNIFORM gets worse, and DE_UNIFORM is unable to find the optimum of OSoREB in higher dimensions.

On the other hand, the opposite trend can be viewed in the performance of CMA-ES, which outperforms all other algorithms. Nevertheless, for the 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 dimensions, but scale the worst. Therefore is almost useless for higher dimensions.

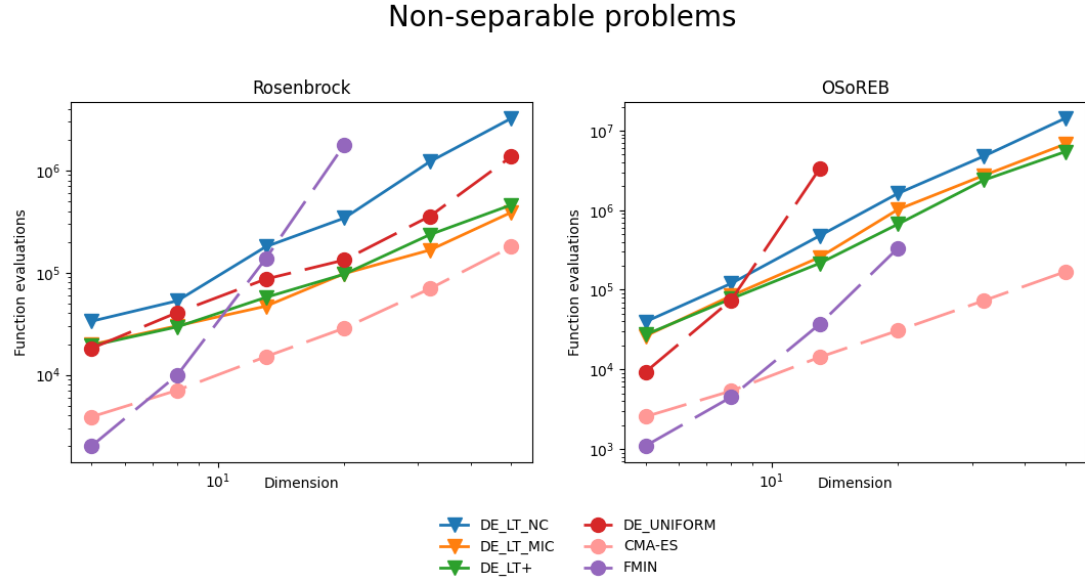


Figure 6.3. Scalability graphs of non-separable problems. Each point is the median of successful runs.

6.2 BBOB problems

The 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 the x-axis, a base-10 logarithmic scale is used, and the total number of fitness function evaluations is divided by dimension. Each graph also shows the performance of the best algorithm of BBOB workshop 2009 noted as *best 2009*.

All results in this section were obtained by the COCO platform. Complete results contains 25 function for various dimensions (2, 3, 5, 10, 20, 40). Complete results are shown in Appendix B. In this section, only selected functions of dimensions 5 and 20 occur. These functions were chosen to represent the characteristic trend seen for more functions.

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

Secondly, for a number of functions, MP variants of DE are outperformed by other DE variants, especially for higher dimensions. Nevertheless, LT and UNIFORM variants

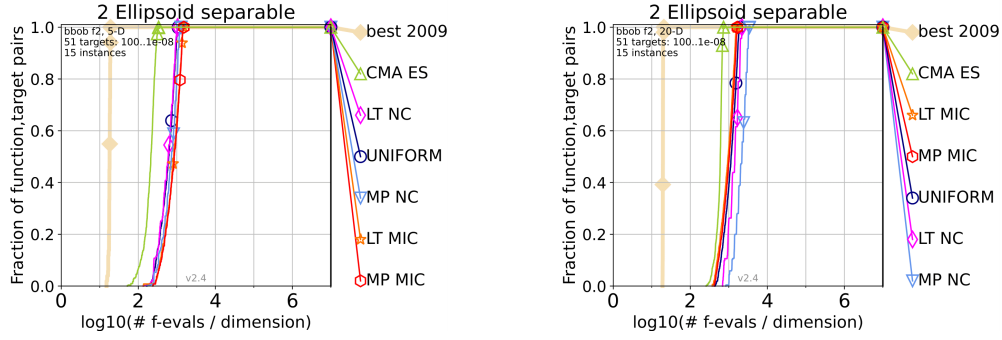


Figure 6.4. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Ellipsoid separable function for dimensions five and twenty.

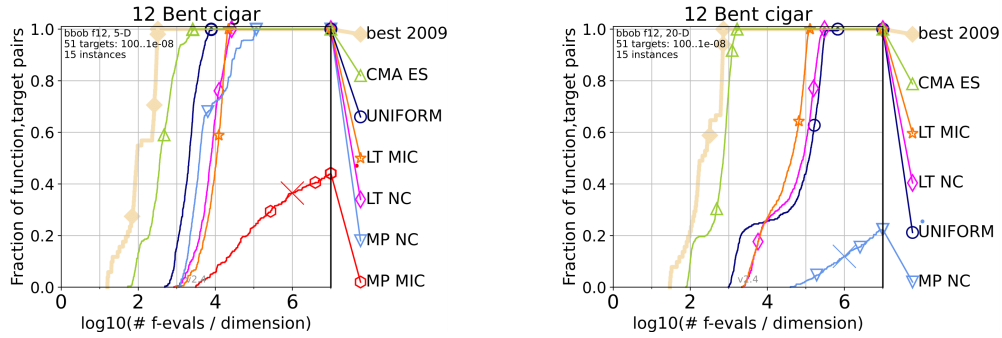


Figure 6.5. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Bent function for dimensions five and twenty.

are outperformed by CMA-ES and by best 2009. An example of such a function may be seen in figure 6.5.

The third observed trend within BBOB functions is a significantly worse performance of DE variants against CMA-ES and best 2009. DE variants are unable to find the global optimum for the majority of these functions, especially in higher dimensions. Note that these functions are mainly Multi-modal functions with a weak global structure. This trend is captured in figure 6.6.

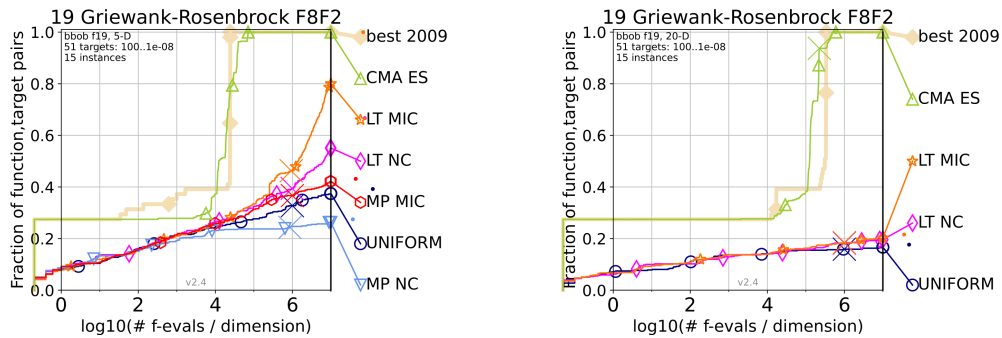


Figure 6.6. Graphs of the empirical cumulative distribution functions of the introduced algorithms on the Ellipsoid separable function for dimensions five and twenty.

Chapter 7

Conclusion

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

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

In chapter 4, two approaches to find pairwise dependencies between variables were introduced. Firstly, the fitness-based approach called non-linearity check (NC) determines the possible dependence between a pair of variables according to the fitness values. Secondly, an approach called maximal information coefficient (MIC) identifies 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 for four regular variants using the above-mentioned methods DE_LT_NC, DE_MP_NC, DE_LT_MIC, DE_MP_MIC, two artificial variants with full prior knowledge of dependency structure were also proposed (DE_LT+, DE_MP+).

According to the results presented in section 6.1, newly proposed methods show to achieve 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 the FMIN algorithm, and for some problems, even better scalability than state-of-the-art evolutionary algorithm CMA-ES. The usage of any of four proposed regular variants of DE instead of the original one would bring better performance.

According to the results from the previous chapter, it may be concluded that linkage tree representation, thanks to its robustness, seems to be a 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 easily recognize the separability of two variables, the same task is challenging for MIC ones because the DE selection operator aligns 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 takes advantage of relatively 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 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 reduced by further adjustments.

For instance, *incremental dependency updating* described in [16] would probably 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 unanswered 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. Moreover, the optimal linkage structure of these problems is unknown. So to find the optimal structure for these problems would also be 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-maes>. 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, İlhan 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: numbbbo/COCO on Github*. <https://github.com/numbbbo/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

Abbreviations

BBOB	Black box optimization benchmarking
CMA-ES	Covariance matrix adaptation evolution strategy
COCO	Comparing continuous optimizers
DE	Differential evolution
DE_LT	Differential evolution which uses the linkage tree to represent the structure of a problem
DE_LT+	Differential evolution which uses the linkage tree to represent the structure of a problem and has full prior knowledge of pairwise dependencies
DE_LT_MIC	Differential evolution which uses the linkage tree to represent the structure of a problem and finds dependencies by maximal information coefficient
DE_LT_NC	Differential evolution which uses the linkage tree to represent the structure of a problem and finds dependencies by non-linearity check
DE_MP	Differential evolution which uses the marginal product to represent the structure of a problem
DE_MP+	Differential evolution which uses the marginal product to represent the structure of a problem and has full prior knowledge of pairwise dependencies
DE_MP_MIC	Differential evolution which uses the marginal product to represent the structure of a problem and finds dependencies maximal information coefficient
DE_MP_NC	Differential evolution which uses the marginal product to represent the structure of a problem and finds dependencies by non-linearity check
DE_UNIFORM	The original version of differential evolution
EA	Evolutionary algorithm
ECDFs	Empirical cumulative distribution functions
FOS	Family of subsets
LT	Linkage tree
MI	Mutual information
MIC	Maximal information coefficient
MP	Marginal product
NC	Non-linearity check
OSoREB	Overlapping Sum of Rotated Ellipsoid Blocks function
SoREB	Sum of Rotated Ellipsoid Blocks function



Appendix B

Complete BBOB results

Appendix C

Implementation and contents of CD

Here a brief description of a program used to generate results 6 is provided. All source codes together with README.txt file are on the enclosed CD. The program offers two functionalities for users.

Firstly, the procedure of finding the population size of chosen algorithms for chosen problems and dimensions as was described in section 5.4.1.

Secondly, finding the median over successful runs of the selected algorithm for selected problem and dimension.

The contents of CD:

- BcThesis.pdf
- comparison.c
- comparison.h
- fitnessFunctions.c
- fitnessFunctions.h
- main.c
- Makefile
- mine.c [23]
- mine.h [23]
- parameters.c
- parameters.h
- README.txt
- search.c
- search.h
- utils.c
- utils.h