České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Petr K ö r n e r

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Evoluční algoritmy s odhadem úspěšnosti křížení

Pokyny pro vypracování:

1. Prostudujte existující metody pro odhad úspěšnosti křížení v evolučních algoritmech (EA), příp. navrhněte vlastní. Inspirujte se u metod globální optimalizace, jako jsou např. metody STEP a DIRECT.
2. Implementujte zvolené metody a otestujte je v praxi v rámci EA.
3. Porovnejte výsledky evolučního algoritmu vybaveného odhadem úspěšnosti s klasickým algoritmem na několika typově různých účelových funkcích a zhodnoťte rozdíly.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Petr Pošík, Ph.D.

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 3. 2. 2011

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Petr Körner

Study programme: Software Engineering and Management

Specialisation: Intelligent Systems

Title of Bachelor Project: Evolutionary Algorithms with Crossover Success Estimation

Guidelines:

1. Explore the existing methods of estimating the crossover operation success in evolutionary algorithms (EA), or propose a new ones. Methods of global optimization, like STEP or DIRECT, can serve as an inspiration.
2. Implement the chosen methods and incorporate them into EA.
3. Compare the results of EA equipped with crossover success estimation and ordinary EA using several diverse objective functions and assess the differences.

Bibliography/Sources: Will be provided by the supervisor.

Bachelor Project Supervisor: Ing. Petr Pošík, Ph.D.

Valid until: the end of the winter semester of academic year 2011/2012

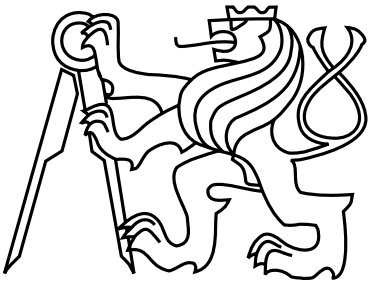
prof. Ing. Vladimír Mařík, DrSc.
Head of Department

L.S.

prof. Ing. Boris Šimák, CSc.
Dean

Prague, February 3, 2011

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor Project

Evolutionary Algorithms with Crossover Success Estimation

Petr Körner

Supervisor: Ing. Petr Pošík, Ph.D.

Study programme: Software Engineering and Management

Specialisation: Intelligent Systems

May 27, 2011

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, Ing. Petru Pošíkovi, Ph.D., za cenné rady a přípomínky a za jeho nadšený přístup a příjemnou spolupráci.

Velké poděkování taktéž náleží celé mojí rodině, která je mi při mých studiích nesmírnou oporou, a které si velmi vážím.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu literatury.

V Praze dne 27. 5. 2011 .....

Abstract

This bachelor project aims on improvement of fitness evaluation process in evolutionary algorithms. In its progress, existing methods of fitness efficiency enhancement were explored and two new methods of crossover success estimation were proposed and then tested. The proposed methods served to create new types of selection of individuals for crossover process and as such were incorporated into an evolutionary algorithm. Performance of the designed algorithms, along with ordinary evolutionary algorithms, was then tested on optimisation of common testing objective functions with modified domains. Extensive tests proved that the methods of crossover success estimation can be efficient in optimisation of certain functions, but are not successful on all of the tested types of problems. Nevertheless, it was determined that the crossover success estimation is effective at least at the beginning of the evolution.

Abstrakt

Tato bakalářská práce se zaměřuje na zlepšení procesu ohodnocení fitness v evolučních algoritmech. V jejím průběhu byly prozkoumány existující metody pro zvýšení efektivity ohodnocení fitness a byly navrženy a otestovány dvě nové metody odhadu úspěšnosti křížení. Předložené metody posloužily k vytvoření nových druhů selekce jedinců ke křížení a jako takové byly začleněny do evolučního algoritmu. Úspěšnost takto navržených algoritmů byla poté testována v porovnání s běžnými evolučními algoritmy na optimalizaci obvyklých testovacích cílových funkcí s upravenými definičními obory. Rozsáhlé testy ukázaly, že navržené metody odhadu úspěšnosti křížení mohou být efektivní při optimalizaci některých funkcí, ale nejsou úspěšné na všech testovaných problémech. Nicméně bylo zjištěno, že odhad úspěšnosti křížení je vždy přínosný alespoň v počátku evoluce.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimisation, as an extensive and important discipline of computer science, becomes greatly useful in application to various real-life problems in order to solve them or improve previously known solutions to these problems.

In typical situation, optimisation means solving problems by searching for a solution to minimise or maximise objective function that represents the solved problem. Generally, this is performed by searching for optimal or suboptimal values of the objective function within its given domain. Various types of objective functions with different types of domains can be optimised.

In order to solve an optimisation problem, an optimisation algorithm or method may be used. While many different computational optimisation techniques exist in computer science, this bachelor project focuses on one of the optimisation methods of artificial intelligence, that is an *evolutionary algorithm*.

1.1 Evolutionary algorithms

Evolutionary algorithms (EAs), sometimes, under certain circumstances, also called genetic algorithms (GAs), heuristically search for exact or approximate solutions to optimisation problems. Evolutionary algorithms were developed and described by J. Holland [1] and later by D. Goldberg [2].

An evolutionary algorithm works with a set of k states, called the population. Each state, or individual, represents a solution to the problem that is being solved or optimised. Each individual is represented as a string of characters—most frequently, a string of binary digits or a string of real numbers. Sometimes, evolutionary algorithms that use binary representation of an individual are called *genetic algorithms*. Since this work concentrates on optimisation of problems in continuous spaces where an individual is represented as a string of real number values, we will stick exclusively to the use of term *evolutionary algorithms* or its abbreviation *EAs*, in further text.

Each individual in the population can be rated by evaluation, or objective, function which is, in EA terminology, called the fitness function. The fitness function evaluation determines quality of an individual within the solved problem. To meet this requirement, the fitness

function should return higher values for better states if the objective of optimisation is maximisation or lower values if the objective is minimisation.

Standard EA scheme begins with initialisation of population followed by start of repeated generational cycle, where each cycle consists of operations of *selection*, *crossover*, *mutation*, and *replacement*, which are used by the algorithm to sample new points in the search space. The generational cycle is stopped as soon as the algorithm reaches its terminal condition. For further details, see Chapter 2.

1.2 Motivation

Although evolutionary algorithms have proved their great functionality in solving various optimisation problems, there are several disadvantages that arise with the use of EAs for complex real-life problems. The most significant disadvantage, in such cases, is repeated fitness function evaluation, the most limiting part of evolutionary algorithms. Searching for the optimal solution to complex high dimensional and multimodal problems generally requires number of expensive fitness function evaluations.

In order to reduce this handicap, various approaches have been proposed, that would either reduce the amount of fitness function evaluations, as in *fitness inheritance* method [3], or simplify the computational process of fitness function. In order to achieve the latter, an approximated fitness function model may be used, as in *fitness modelling* [4], instead of an exact fitness evaluation.

1.3 Objectives

As opposed to the two above mentioned methods, the approach proposed in this bachelor thesis may reduce amount of fitness function evaluations by crossing over only the parents that might reproduce a successful offspring.

In a basic EA, after a whole new population of offspring is created, every single offspring is evaluated with fitness function and then, only the better successors are selected to form a following generation. This implies that a number of successors (those with worse fitness values) was reproduced and evaluated unnecessarily.

To reduce this inconvenience, this bachelor project shall provide a method that would estimate successfulness of crossing over any two parents in population, i.e. be able to rate the possibility of any pair of individuals to reproduce a successful offspring. The breeding process would then be driven by these estimations, and only pairs with high possibility of successful reproduction would be crossed over.

Such approach, if implemented, might produce better results and bring following benefits to an evolutionary algorithm:

- smaller amount of fitness function evaluations required in total,
- smaller amount of crossover operations required in total,
- faster convergence of fitness value of the best individual,
- more effective preservation of population diversity in multi-modal functions.

1.4 Navigation

This bachelor thesis is divided into six chapters. Chapter 1 provides general introduction to evolutionary algorithms, defines their weaknesses, and specifies the objectives of the bachelor project. Chapter 2 describes evolutionary algorithms in detail along with further description of existing methods improving their performance. In Chapter 3, new methods for improvement of EAs, the crossover success estimation methods, are proposed and correlation tests are performed to signify their possible contribution to EA improvement. Chapter 4 introduces possible variants of selection methods based on the crossover success estimation, together with other components that will be later used in testing EAs. The ECDF tests are performed to estimate the contribution of the selection methods based on the crossover success estimation. Chapter 5 describes the tested variants of evolutionary algorithms and presents the results of optimisation tests performed to finally determine successfulness of crossover success estimation methods applied to evolutionary algorithms. Lastly, Chapter 6 concludes the bachelor project, summarises its results, and discusses possible development of the crossover success estimation.

Chapter 2

Evolutionary algorithms

The basic information on evolutionary algorithms was provided in previous chapter. The aim of this chapter is to characterise evolutionary algorithms in more detail. The most common scheme of the run of an EA is shown in Figure 2.1.

Classic evolutionary algorithm scheme

1. Initialise and evaluate a population of individuals.
2. Repeat to create a new population:
   (a) Select a pair of parents.
   (b) Create an offspring by crossover.
   (c) Mutate the offspring.
   (d) Evaluate the offspring.
   (e) Add the offspring to new population.
3. Replace old population of parents with the new one.
4. If terminal condition is not reached, go to step 2.

Figure 2.1: Steps in a standard evolutionary algorithm

2.1 Components of EA

The following text describes every important component of an EA in the same order as it is used during the run of the algorithm.

2.1.1 Initialisation

Usually, evolutionary algorithms begin with a set of k randomly generated individuals that make the initial population, also determined as the first generation. The initial population may also be created non-randomly, for example using individuals created based on certain knowledge of the problem. The whole population is then evaluated by the fitness function.

2.1.2 Selection

After every individual in the population is evaluated, various pairs of individuals are being sequentially selected for reproduction, or crossover process. This step may be done in various ways depending on chosen selection method.

Selection is one of the most important parts of an EA, having a significant influence on behaviour of the whole algorithm and its convergence to optimum value. Therefore, the selection method requires great attention during the design of an evolutionary algorithm.

Most selection methods choose individuals for crossover in accordance with their fitness values, where the best individuals are selected with the highest probability, while the worst ones with the lowest probability. This causes that some individuals in a generation may be selected for crossover process more than once while some other may not be selected at all.

2.1.3 Crossover

Crossover is a process in which a selected pair of individuals is used to produce new offspring. The representation of an individual, called *chromosome*, has an essential importance in this process, since the new offspring is usually created entirely from its parents' chromosomes.

Often, the offspring are created by crossing over the parent strings at crossover point or points, where the crossover point is randomly chosen from the positions in the parent chromosome string. If a single crossover point is generated, each parent chromosome is divided into two parts by this crossover point. The first offspring can then be created by merging the first part of the first parent chromosome and the second part of the second parent chromosome. Likewise, the second offspring would then be created by merging the first part of the second parent chromosome and the second part of the first parent chromosome.

The product of crossover process does not necessarily need to be a pair of offspring, since only a single offspring may be reproduced, depending on the crossover method used.

2.1.4 Mutation

A very important part of an evolutionary algorithm is the process of mutation, which slightly modifies the offspring. Since every single offspring's chromosome is produced by combining sections of chromosomes from each parent, some chromosomes with good fitness values may exist, that the algorithm might not be able to reach only by crossing over.

To reduce this problem, each offspring created in the crossover process has a chance to be mutated, given by certain probability. The mutation of an individual is then performed by slight modification of each piece of its chromosome.

Besides the fact that mutation is very important for an evolutionary algorithm to find better solutions more often, it also helps to maintain diversity of the population throughout the entire run of the algorithm.

2.1.5 Replacement

The whole sequence of selection, crossover and mutation is being performed repeatedly until a new population of successors is created, with its size being the same as the size of the

initial population. These two populations are then processed, using a replacement strategy, to form the next generation. Often, the old population of parents is completely replaced with the new population of successors. This method is called the *generational replacement*.

A whole class of other replacement methods, the *steady-state strategies*, is also frequently used. In steady-state replacement strategy, individuals from both, the old and the new population, are selected to form the following generation, based on fitness value of each individual. This approach, among others, preserves the population diversity.

2.1.6 Termination

Generally, average fitness value of individuals in any generation throughout the run of an EA is better than average fitness value of individuals within its preceding generation, which is how the algorithm optimises the objective function gradually and finds better and better solutions to the problem. Usually, an evolutionary algorithm itself is terminated when the final solution to the problem is found, if such case is distinguishable, or when the designated number of generations is reached.

2.2 Fitness efficiency enhancement

As it was explained before, fitness evaluation in real-world applications often causes a lot of computational overhead. The previously mentioned methods of *fitness inheritance* and *fitness modelling* were proposed for tackling this problem.

2.2.1 Fitness inheritance

An approach called fitness inheritance, which reduces the amount of fitness evaluations, was originally proposed by R. Smith in paper [3]. This paper examines a genetic algorithm that overcomes the difficulty of computationally expensive fitness evaluation of each individual by evaluating only a portion of the population. The remainder of the population has its fitness value assigned by inheritance, which means that fitness of each of such individuals is evaluated indirectly by interpolating the fitness value of its parents.

Results of tests on simple functions and problems were quite promising and indicated that inheritance may allow less expensive population evaluation. However, the authors of paper [5] raised an objection that the problems on which the inheritance had n tested within the original research are very simple and thus, they tested the performance of fitness inheritance on a well known test set of multiple objective optimisation problems in order to determine, whether fitness inheritance is really useful for real-world applications. Results showed that fitness inheritance can only be applied to convex and continuous problems.

2.2.2 Fitness modelling

Fitness modelling was examined for example by A. Ratle [4]. The paper investigates Kriging interpolation and estimation as a fitness function approximation for the optimisation of computationally complex functions. A model of the fitness function is built from a small

number of samples of this function. The model, initially, is a global approximation of the entire domain, and successive updates during optimisation process transform it into a more precise local approximation.

Results obtained from theoretical problems showed that significant gains can be obtained compared to basic evolutionary algorithms. However, the translation of these results to a real-world problems of unknown structure have appeared to be very difficult.

This bachelor project shall provide another different techniques for fitness efficiency enhancement. The proposed methods are based on *crossover success estimation* of two individuals and will be described in the following chapter.

Chapter 3

Crossover success estimation

To accomplish objectives given in assignment of this bachelor project, two methods of crossover success estimation were proposed. The design of these methods was dependent on few preconditions. First, to reduce computational overhead caused by fitness evaluation, any crossover success estimation method proposed has to be relatively simple, the crossover success estimation for any pair of individuals must not be computationally complicated. Furthermore, a crossover success estimation method should be closely related to the used crossover method.

Benefit of the use of crossover success estimation method will depend on the optimised objective function and, eventually, on current evolution phase of the EA.

The crossover success estimation for a pair of individuals in a population will be expressed by *crossover complexity* of this pair. The crossover complexity of a pair of parents estimates how profitable would crossover of this pair be, in the sense of produced offspring's quality.

Both proposed methods—*linear complexity* and *parabolic complexity*—are different, yet similar in some characteristics. Besides, both use elements derived from existing deterministic global optimisation algorithms.

3.1 Linear complexity method

The modified *Lipschitzian optimization* method was first introduced by D. Jones [6] and used in *DIRECT* global optimisation algorithm described, among others, by D. Finkel [7]. The Lipschitz optimisation can be applied to a closed interval of continuous function f of an unknown shape in two-dimensional space.

As Figure 3.1 shows, the very first estimation of minimum value of the function f is calculated as a point of intersection of two lines, while each of them cross one endpoint of the function interval and form equivalent, yet inverse, angle $\pm\alpha$ with an imaginary vertical line crossing the intersection of these two lines.

Subsequently, the whole interval is divided into two regions by the point of intersection and the algorithm continues by performing the same operation on the smaller regions to obtain more accurate estimation of the function minimum. Figure 3.2 shows the second iteration of the algorithm.

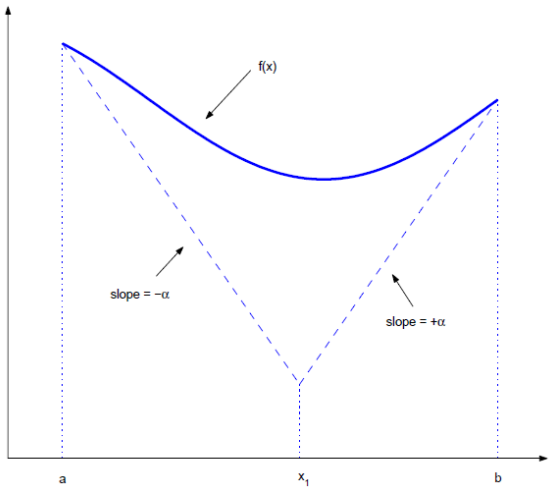


Figure 3.1: An initial estimate of the minimum value of f in the Lipschitz optimisation (Adopted from Finkel: *DIRECT Optimization Algorithm User Guide*)

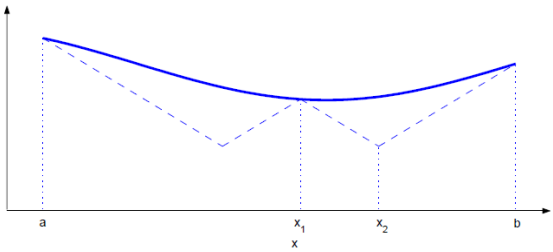


Figure 3.2: A second estimate of the minimum value of f in the Lipschitz optimisation (Adopted from Finkel: *DIRECT Optimization Algorithm User Guide*)

The linear complexity method proposed in this bachelor project is significantly inspired by the Lipschitz optimization and its application in DIRECT algorithm, although it differs in some aspects.

Linear complexity can be calculated as an estimation of successful crossover of two different individuals in population of an EA. By the successful crossover, a crossover of parents is meant, that would reproduce an individual with better fitness value than fitness value of the current best individual.

Let $\epsilon > 0$ be a positive constant and let f_{min} be the current best fitness value. In order to estimate crossover success of two individuals, a value f_{min} has to be modified so that it would express the value we need to reach by crossing over a pair of individuals. This is done by subtracting a small value ϵ from f_{min} and thus updating the f_{min} value. A reasonable value for ϵ appeared to be 1×10^{-8}.

Since the idea of intersection of lines doesn't translate well into higher dimensions, to compute the linear complexity of two individuals in n-dimensional space, coordinations of

these individuals represented as their chromosomes need to be transformed into the space of single dimension. This is done by moving the first individual to the origin of coordinate system and by transforming coordinates of the second one to 1-dimensional space by calculating the Euclidean distance between these two individuals. The knowledge of fitness values of both individuals is essential.

When the updated f_{min} value is acquired and coordinates of the pair of individuals are transformed, a linear complexity of the pair can be computed. Both individuals, now labelled as X_1 and X_2, are now represented as points in a 2-dimensional space, where the x-coordinates are defined by the transformed coordinates of both individuals, and the y-coordinates are their fitness values.

As it is shown in Figure 3.3, two lines intersected in the value f_{min} need to be constructed, thus that each of them cross a parent point X_1 or X_2 and form an equivalent acute angle with the line defined the by value f_{min}.

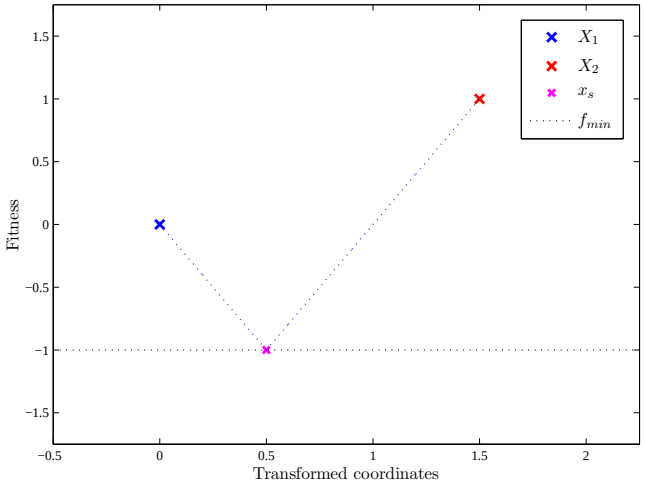


Figure 3.3: Demonstration of linear complexity method on two parent points

The problem of construction of such lines can be formulated using a slope-intercept equation form for a line:

$$y = kx + q, \tag{3.1}$$

where k is a slope, or gradient, of the line and q is the y-intercept—a point at which the line crosses the y-axis. By substituting into this equation, we can describe two lines, each crossing one individual X_1 or X_2, as

$$f_1 = -kx_1 + q_1, \tag{3.2}$$

$$f_2 = kx_2 + q_2, \tag{3.3}$$

where f_1 and f_2 are fitness values of parents X_1 and X_1, while x_1, x_2 are transformed coordinations of each of the parents.

Let x_s be an x-coordinate of the intersection point of the two designated lines. Then the following two equations describe a situation when this intersection lies in value f_{min}:

$$f_{min} = -kx_s + q_1, \tag{3.4}$$

$$f_{min} = kx_s + q_2. \tag{3.5}$$

By solving the set of previous four equations, we get a solution to the problem, that is a single equation for value

$$k = \frac{f_2 - f_{min} + f_1}{x_2}, \tag{3.6}$$

where k is the slope of both lines and will be used as a crossover success estimation value of the designated pair of individuals.

The closer a pair of individuals is to each other, the more acute is the angle between two lines, each crossing one parent point, with intersection in x_s, and thus, the higher is the slope k. The angle between these two lines forming a V-shape also gets more acute with increasing distance from the parent points to the updated f_{min} value. Therefore, the value k depends on distance between pair of individuals X_1 and X_2, and on distance between the pair and the f_{min} value.

As a result, in the sense of linear crossover complexity of two individuals expressed through the value k, the worst possible pair to cross over is a pair of individuals that is very close to each other and far away from f_{min} at the same time. In such case, the k value is high. On the contrary, the best pair to cross over is a pair of individuals that is far away from each other and very close to f_{min} at the same time, in which case of, the k is minimal.

If the linear crossover complexity method is implemented and used in selection process, only those pairs of individuals are crossed over, for which the complexity method returns an adequately small value.

3.2 Parabolic complexity method

The *S.T.E.P.* algorithm for global optimisation was developed by S. Swarzberg [8]. The algorithm determines the next point where to evaluate an objective function in, by analysing the usefulness of the function evaluation at a certain position. The essential idea is to evaluate the objective function at a point for which there is the greatest chance of exceeding the best value found until then.

To achieve this, the search space is divided into partitions delimited by already evaluated points, and the partition with the greatest possibility of including a point, that would exceed the best point, would be chosen. To determine such partition, every partition is evaluated with its difficulty, where partition with the minimum difficulty is the best.

The difficulty of each partition is represented as a second derivative of a parabola passing through the two boundaries of the partition, and with a certain value at its optimum defined in dependence on value of the current best point.

Proposed parabolic complexity method is strongly inspired by the S.T.E.P. algorithm and its idea of the difficulty expressed through a second derivative of a parabola. Same as in

the linear complexity method, the complexity of crossover can be computed for any pair of distinct individuals in the population of an EA.

Let X_1 and X_2 be points representing the designated pair in 2-dimensional space, where the x-coordinates are transformed original coordinates and y-coordinates are individuals' fitness values, same as in the linear complexity method. Let again f_{min} be the value of the best current individual decreased by a small constant ϵ. Then, a parabola crossing both parent points X_1 and X_2, with its vertex—the lowest point—lying in f_{min} as shown in Figure 3.4, can be constructed.

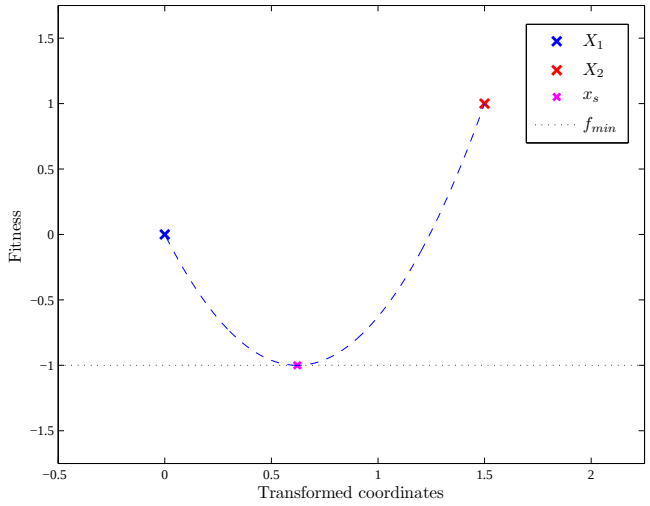


Figure 3.4: Demonstration of parabolic complexity method on two parent points

The parabola is expressed in standard form equation as

$$y = ax^2 + bx + c, \tag{3.7}$$

where a, b and c are parabola's parameters. Because the x-coordinate of the parent point X_1 is always zero, a parabola crossing this parent can be expressed in standard form equation

$$f_1 = c, \tag{3.8}$$

where f_1 is fitness of the first individual. Using this knowledge, condition of the parabola crossing the parent point X_2 can be expressed as

$$f_2 = ax_2^2 + bx_2 + f_1, \tag{3.9}$$

where f_2 is fitness value of the second individual, and x_2 is x-coordinate of the parent point X_2. Finally, the position x_s of parabola's vertex lying in value f_{min} is defined by equation

$$f_{min} = ax_s^2 + bx_s + f_1. \tag{3.10}$$

Let the desired parabola be denoted as p. Then its second derivative, describing the curvature of the parabola in its vertex, can be calculated from the parameter a as

$$p'' = 2a. \tag{3.11}$$

Therefore, by solving a set of three equations 3.8 to 3.10, we can get the parameter a and use it to calculate p'', which will be used as a crossover success estimation value for the designated pair of individuals.

The closer a pair of individuals is to each other, the more closed the parabola p gets, the greater the curvature in its vertex is and thus, the greater also its second derivative p'' is. The curvature in the parabola's vertex, together with the second derivative p'', gets smaller with decreasing distance between the pair of individuals and the f_{min} value.

Similarly as in the linear complexity method, the value of p'' that serves as a crossover success estimation value is dependent on distance between individuals X_1 and X_2 from one to the other, and also on distance from the pair to the f_{min} value.

As a result, if the parabolic complexity method is used for crossover success estimation for pairs of individuals from the population of an EA, only the pairs with small p'' value would be crossed over.

3.3 Correlation tests

To reduce excessive fitness evaluation overhead in evolutionary algorithms, the proposed crossover success estimation methods may be used to estimate usefulness of crossing over certain pairs in the population, and only the pairs that would, by estimation, reproduce a successful offspring, will be crossed over.

To prove that such approach is acceptable, correlation tests between the estimation and the reality need to be performed. In other words, we have to determine whether there is a relation between complexity of a pair of individuals and fitness value of actual offspring reproduced by this pair. For such purposes, *Pearson product-moment correlation coefficient* and *Spearman's rank correlation coefficient* should serve well.

The Pearson correlation coefficient is used to measure the strength of linear dependence between two variables X and Y. For two vectors of variable X and Y values, it returns a coefficient value from interval $[-1, 1]$. A correlation value of 1 implies that Y is perfectly dependent on X and contrary-wise. A value of -1 means that Y is perfectly dependent on X, but reversely, and a value of 0 implies that variables X and Y are completely linearly independent from each other.

The Spearman correlation coefficient expresses how well can the relationship between two variables X and Y be described using a monotonic function—a function that preserves the given order. Same as in the Pearson correlation, it returns value from interval $[-1, 1]$. The correlation coefficient is calculated as the Pearson correlation, but between the ranked variables instead of the actual values.

Throughout the testing, correlations between the crossover complexity of a pair of individuals and actual fitness of the reproduced offspring were measured using both the Pearson

and Spearman correlation coefficients. The tests were performed by sampling points in the search space of different objective functions with various dimension sizes. The used objective functions were *linear*, *sphere*, *ellipsoid*, *Griewank* (two variants with different domains) and *Rosenbrock* function. For more details about the testing functions, see Appendix A.

First of all, tests on an initial population of EA were performed. In every single test, a population of size $k = 50$ was randomly generated within the domain of the tested objective function. Then, different pairs of individuals were being randomly selected from the population. The linear and parabolic complexity was calculated and recorded for each selected pair and then, the pair was crossed over and the reproduced offspring's fitness value was recorded as well.

Afterwards, both the Pearson and Spearman correlation coefficients were calculated for the linear and parabolic complexity method and corresponding correlation graphs were plotted. The graphs and the correlation coefficients for linear function in 5-dimensional space can be seen in Figure 3.5.

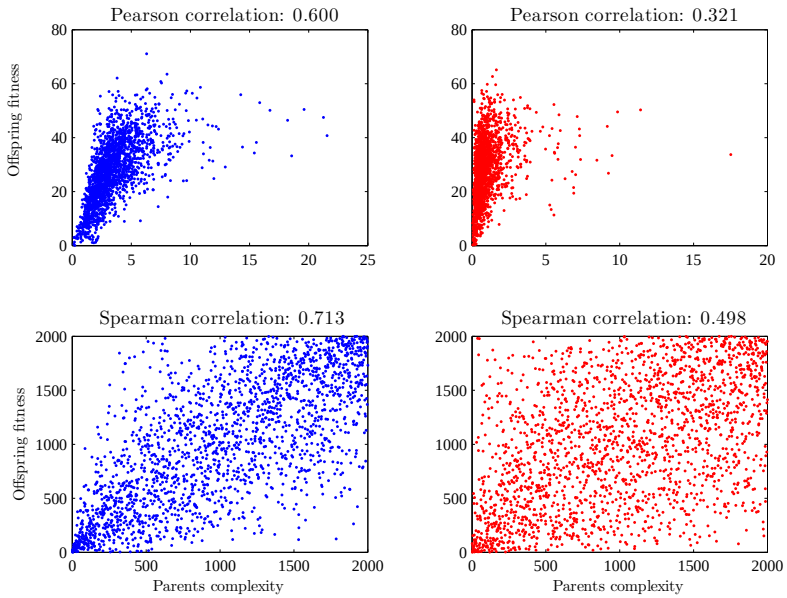


Figure 3.5: Correlation graphs of the linear complexity method (blue) and parabolic complexity method (red) for initial population of the linear function in 5D

As one can see from the graphs, the use of the linear complexity method in this case ensures much higher correlation coefficients than the use of the parabolic complexity method. Besides, the Spearman correlation coefficient is higher in both cases.

The results of tests for the remaining objective functions, tested in the space of the same dimension, are recorded in Table 3.1.

Objective function	Linear complexity		Parabolic complexity	
	Pearson	Spearman	Pearson	Spearman
Linear	0.600	0.713	0.321	0.498
Sphere	0.665	0.791	0.108	0.639
Ellipsoid	0.750	0.788	0.451	0.752
Griewank1	0.700	0.799	0.373	0.638
Griewank2	0.449	0.548	0.191	0.399
Rosenbrock	0.731	0.817	0.536	0.730

Table 3.1: Correlations of crossover complexity methods for initial population in 5D

From the values in the table, it appears that the linear complexity method is more useful in estimation of crossover success than the parabolic method, as it has higher correlation in the use with all of the objective functions. Other than that, both of its correlation coefficients are more or less stable.

The same tests as performed in the space of dimension 5 were also performed in the space of dimension 10 and 20. For demonstration, correlation graphs for Rosenbrock function in the space of dimension 20 are shown in Figure 3.6.

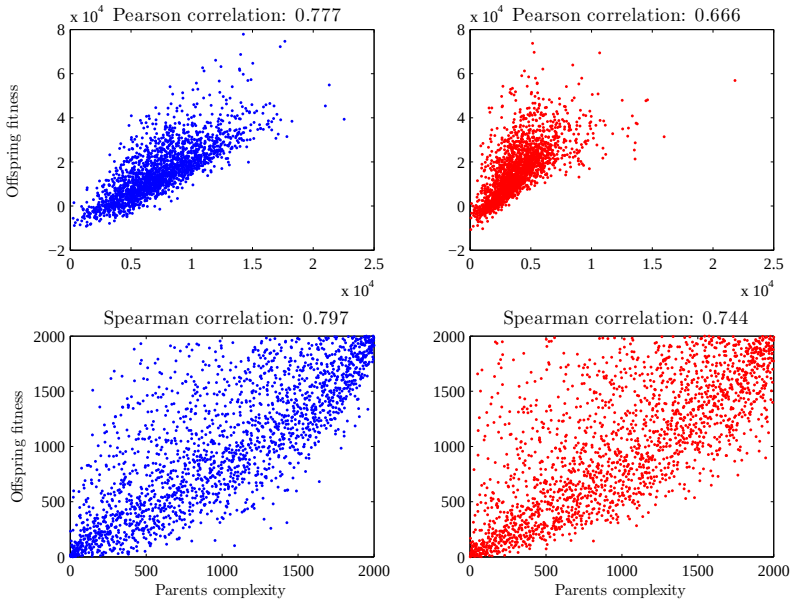


Figure 3.6: Correlation graphs of the linear complexity method (blue) and parabolic complexity method (red) for initial population of Rosenbrock function in 20D

In comparison to the results of tests in 5-dimensional space, the correlation graphs on Rosenbrock function show better correlation between the complexity of parents and the actual fitness of the reproduced offspring in both, the linear and especially the parabolic

complexity method. Correlations for both methods are also more balanced as opposed to the results from previous tests. The correlation coefficient values for other objective functions in 20-dimensional space can be seen in Table 3.2.

Objective function	**Linear complexity**		**Parabolic complexity**	
	Pearson	**Spearman**	**Pearson**	**Spearman**
Linear	0.825	0.828	0.664	0.700
Sphere	0.803	0.813	0.701	0.741
Ellipsoid	0.809	0.798	0.743	0.782
Griewank1	0.791	0.805	0.691	0.729
Griewank2	0.777	0.787	0.690	0.742
Rosenbrock	0.779	0.797	0.666	0.744

Table 3.2: Correlations of crossover complexity methods for initial population in 20D

The results of tests performed on the initial population of an EA suggest that the methods of crossover success estimation might cause significant improvement in convergence at the beginning of the evolution in an EA. Nevertheless, it is still necessary to determine how dependable the estimation methods would be, if the evolutionary algorithm is partially converged. To estimate this, we need to measure the correlations in partially converged state of an EA.

In order to measure correlations in converged state without the need of running the complete evolutionary algorithm, the required number of generations was only simulated. The simulation of n-th generation is done by randomly generating a population of size $k \times n$, where k would be the size of initial population, and choosing only the best k individuals.

When a desired generation was simulated, the same correlation tests were performed as before. The results of tests performed after 30 simulated generations, recorded in Table 3.3, show, that the correlations measured after the given number of generations are lower than those measured right after the initialisation. The degradation is slightly more noticeable in the case of the parabolic complexity method, though.

Objective function	**Linear complexity**		**Parabolic complexity**	
	Pearson	**Spearman**	**Pearson**	**Spearman**
Linear	0.720	0.727	0.510	0.528
Sphere	0.718	0.723	0.559	0.614
Ellipsoid	0.800	0.796	0.701	0.755
Griewank1	0.740	0.755	0.561	0.623
Griewank2	0.705	0.724	0.537	0.594
Rosenbrock	0.697	0.718	0.612	0.659

Table 3.3: Correlations of crossover complexity methods after 30 generations in 20D

Another results, this time measured after the simulation of 80 generations, can be seen in Table 3.4. The correlations for all objective functions decreased again, but this time only slightly in comparison to the previous table.

Objective function	**Linear complexity**		**Parabolic complexity**	
	Pearson	**Spearman**	**Pearson**	**Spearman**
Linear	0.700	0.707	0.497	0.541
Sphere	0.696	0.718	0.563	0.617
Ellipsoid	0.786	0.786	0.683	0.734
Griewank1	0.717	0.745	0.541	0.584
Griewank2	0.718	0.728	0.575	0.632
Rosenbrock	0.698	0.723	0.593	0.640

Table 3.4: Correlations of crossover complexity methods after 80 generations in 20D

According to the correlation tests performed on the objective functions of dimensions 5, 10 and 20, and also in different phases of evolution, the approach of the crossover success estimation appears to be quite promising.

The tests verified that there really is a connection between the estimated crossover success and the fitness of the produced offspring. Nevertheless, this connection does not predicate how successfully may an EA, with selection based on the crossover success estimation, work, because the successfulness of the algorithm consists in improving the best solution, which was not measured in the correlation tests.

To determine whether the crossover success estimation methods are profitable for an EA or not, more different testing has to be done. As the linear crossover complexity method appeared to have better results throughout all the correlation tests, we will concentrate on this method in the next progress. The following chapters shall offer few different selection methods based on the crossover success estimation provided by the linear complexity method, and test them in the use with a complete evolutionary algorithm.

Chapter 4

Specification of tested EAs

The linear crossover complexity method was chosen from the two crossover success estimation methods and will be used for selection in an evolutionary algorithm. For comparison of the EA using crossover success estimation and the classic EA, four variants of an EA were implemented and tested in this bachelor project.

4.1 Common components

All four variants share most of the EA components, such as *initialisation*, *crossover*, *mutation* and *replacement*, but they differ in the used *selection* method. This section describes in detail the common components that are used by all of the variants of an EA.

4.1.1 Initialisation

The initialisation of population in each algorithm is *random*. As the domain of optimised objective function is known beforehand, the initialisation is done by randomly generating k sample points with chromosome length n within the function's domain, where k is the population size and n is dimension of the problem.

Duplicity in the initial population is forbidden and thus, any duplicate sample points, or individuals, are replaced with newly generated ones. In the end of the initialisation, all the population's individuals are distinct from each other.

4.1.2 Crossover

It was stated at the beginning of the previous chapter that the crossover success estimation method and the used crossover method have to be closely related to each other. This requirement is accomplished, because same as the crossover success estimation method expresses the complexity of crossover of a pair by interlacing the parent points with lines crossing each other in the f_{min} value inside of the interval restricted by both parents, the used crossover method reproduces a new offspring likewise, that is, inside of the interval defined by the parents.

In this sense, the reproduction of a new offspring is simply done by choosing a random point that lies on an imaginary line between the first and the second selected parent.

4.1.3 Mutation

As every individual in the population is represented as a string of real value numbers, the real valued mutation is used after the crossover process in all of the tested evolutionary algorithms. Each offspring is mutated with probability $p_{mut} = 0.25$ using the *Gaussian mutation*. The Gaussian mutation consists in modification of each element in the offspring's chromosome by adding or subtracting a random number from the Gaussian probability distribution to or from the chromosome's element.

The amount of value by which each element is modified, depends on the *mutation range* r_{mut} that multiplies the randomly generated number. In our case, the mutation range was defined as

$$r_{mut} = 0.015 \times f_{range}, \tag{4.1}$$

where f_{range} is size of the optimised objective function's domain. It is important for an offspring to be mutated in accordance to the size of the function domain, because if the mutation rate was constant, undesirable situations may occur. For example, an individual in a function of extensive domain would be mutated insignificantly, while an individual in a function of small domain would be mutated extremely, exceeding thus the function domain. Both of these extreme cases would have a negative impact on convergence of the EA.

4.1.4 Replacement

The used crossover method, reproducing an offspring as a point lying on the line segment between its parents, tends to generate offspring closer and closer to center of the search space as the algorithm converges. Along with use of the *generational replacement*, this causes that a convex envelope of the population gets smaller and the population becomes located at a certain small area of the search space. As a result, the convex envelope of the population points might completely abandon the part of the search space where the global minimum is located, and the evolutionary algorithm would then converge to a local minimum.

To prevent this inconvenience, a *Steady-State replacement* strategy shall be used. The Steady-State replacement strategies generate selection pressure and ensure that the best individual found so far, is preserved and used in the following generation.

In this bachelor project, one of the Steady-State replacement strategies, called the *Restricted Tournament Replacement*, is used. The Restricted Tournament Replacement, or RTR, was examined, for example, by C. Lima [9]. The RTR method operates with the population of parents and the population of offspring, following steps from Figure 4.1.

For every individual X from the offspring population:

1. Select a random subset W of individuals from the parent population.
2. Choose an individual Y from W that is most similar to the offspring X, in terms of distance of their chromosomes.
3. Replace Y in the parent population with offspring X, if X is better, otherwise reject X.

Figure 4.1: Steps in the Restricted Tournament Replacement method

Besides the preservation of the best existing individual, the Restricted Tournament Replacement method also reduces population size requirements and helps to maintain the population diversity.

4.2 Selection methods

As it was explained, the tested evolutionary algorithms differ in the used selection method. Output of each of the implemented selection methods is a pair of distinct individuals. This section describes the four used selection methods.

4.2.1 Random selection

The random selection method uses no rules when choosing individuals from a population. It randomly selects two individuals from the population, and return them as an output. The only condition is that the selected individuals have to be distinct from each other.

4.2.2 Tournament selection

In general, the tournament selection is one of the most frequently used selection methods in EAs. The version implemented in this project consists of two separated tournaments, where each of them return one selected individual. In each tournament, a small subset M of m individuals from the parent population is randomly selected. Then, the individual from M, with the best fitness value, is returned as a parent. The selection also ensures that both parents selected for crossover are distinct form each other.

4.2.3 Complexity selection

The complexity selection uses crossover complexity values provided by the crossover success estimation method to select pairs of individuals for crossover. This method operates with a large set P of individual pairs, called the *pool*. The pool consists of p pairs, where each pair is selected from the parent population using the random selection.

The crossover success estimation method is then used to calculate crossover complexity of each pair from the pool P. After every pair in the pool is evaluated, m pairs with the smallest crossover complexity are separated to create a set M of the best pairs. Finally, a single pair from M is randomly selected and returned for crossover process as an output.

Throughout the testing, the complexity selection method was used with configuration of parameters set as $p = 100$ and $m = 10$.

4.2.4 Combined selection

The combined selection method is a mutation of the tournament selection and the complexity selection. Same as the other methods, the combined selection returns a pair of individuals. The first individual X is selected using the tournament selection. Then, a pool P similar to the one used in complexity selection method is created and filled up with pairs of individuals.

In this case, each pair consists of the first individual X, previously selected in the tournament, and a randomly selected second individual.

Then again, the crossover complexity of each pair from the pool P is calculated, and the best pair with the smallest crossover complexity value is selected and returned for crossover process. This is different as opposed to the complexity method, which selects the final pair of parents from m best pairs randomly, whereas this method chooses the parent pair as the best pair from the pool P directly.

4.3 ECDF tests

Before the main tests with complete evolutionary algorithms were performed, we decided to measure how the population changes after an execution of one generational step using each of the implemented selection methods. By the generational step, a process is meant, where an amount of pairs selected from the population using a certain selection method, is crossed over to create a new offspring population. Without use of the mutation, the replacement method is then applied to produce a next generation from the parent and the offspring populations.

To measure the distribution of fitness values in a population, the *Empirical Cumulative Distribution Function*, or *ECDF*, can be computed for a vector of fitness values. The ECDF provides estimation $\hat{F}_n$ of the cumulative distribution function for given sample, or a vector of observation values. It is a step function that jumps for i/n at observation values, where i is the number of tied observations at the current value, and n is a length of the sample. In other words, the ECDF output value increases faster in the interval where the input sample contains a large amount of identical (or similar) values, and increases slower in the interval where the sample contains small amount of identical (or similar) values.

In the first place, an initial population was generated, and one generational step was executed using each of the selection methods. The ECDF for fitness values in the initial population and also fitness values in each of the populations newly created using the different selection methods, was then computed and plotted into a graph. The resulting graph for the linear function in 10-dimensional space is shown in Figure 4.2.

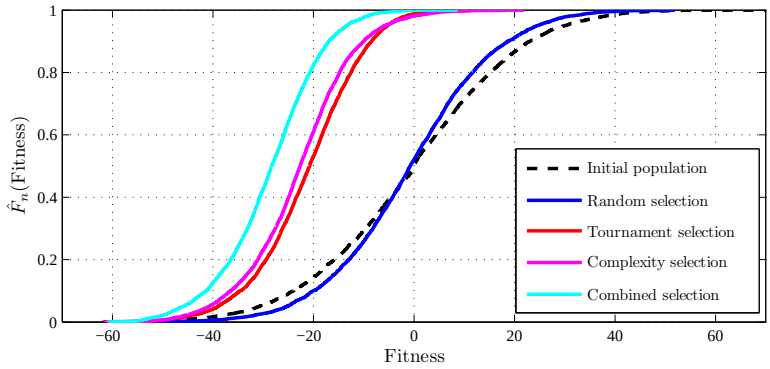


Figure 4.2: ECDF for fitness values in initial population of the linear function in 10D

All of the fitness values were translated with respect to the median of values in the initial population, causing thus that the ECDF graph for the initial population reaches 50% of its range in the fitness value of 0. Besides this, various facts can be observed in the ECDF graph. Firstly, the curve of random selection crosses the curve of the initial population right in the middle in the value 0, yet it is steeper. This shows, that the use of random selection didn't produce better results, in general, it only moved the new population closer to the center of the search space. The ECDF curves for the remaining selection methods show, that these methods produced much better individuals than the random selection. The curves are also steeper, which means that the population is concentrated at the middle of its fitness range.

The Figure 4.3 show the result of ECDF test for the ellipsoid function in 10-dimensional space. As it can be seen in the graph, the population was initially non-uniform, in the means of fitness, and the tournament, complexity and combined selection made the population with more uniform distributed fitness, yet they did not improve the best solution that much.

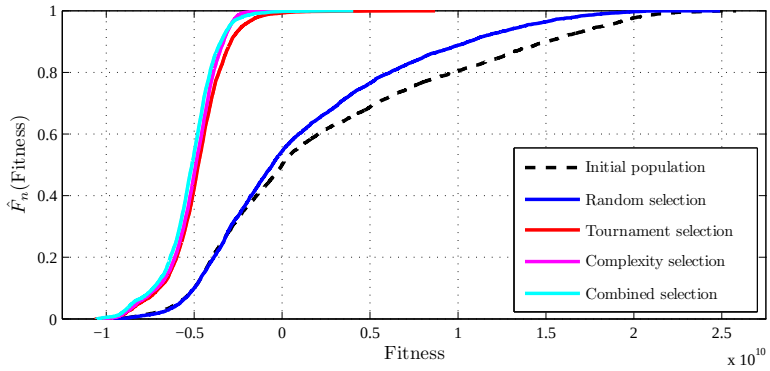


Figure 4.3: ECDF for fitness values in initial population of the ellipsoid function in 10D

The ECDF tests were also performed in a partially converged state of evolution, simulated identically as in the correlation tests in Chapter 3. An ECDF graph for the linear function tested after 10 simulated generations is shown in Figure 4.4.

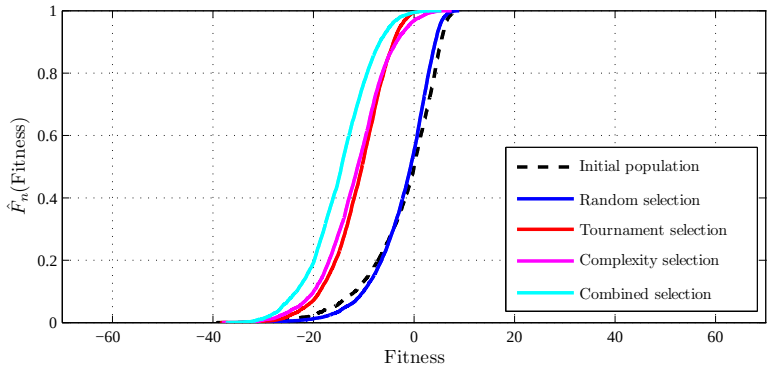


Figure 4.4: ECDF for fitness values after 10 generations of the linear function in 10D

In comparison the the results of the previous test on the linear function, all of the curves are much steeper, which is a result of the partial convergence of the population. The amount of the fitness improvement caused by the selection methods is not as substantial as in tests performed on the initial population.

The last ECDF graph in Figure 4.5 shows results for the linear function tested after 50 simulated generations. The curves are again little steeper, but the difference between the results of tests after 10 and 50 simulated generations, is not as substantial as in the comparison to the test on the initial population.

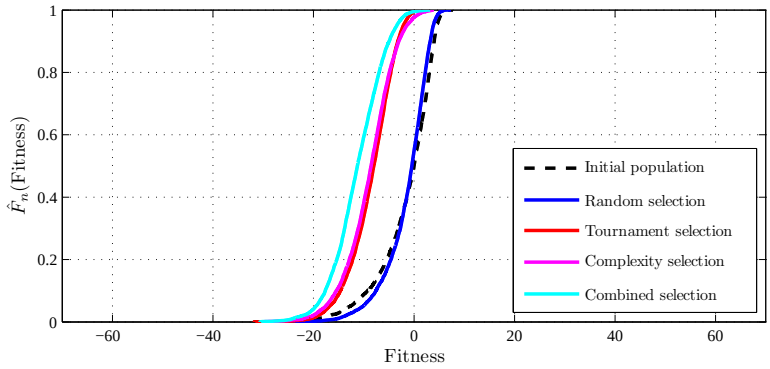


Figure 4.5: ECDF for fitness values after 50 generations of the linear function in 10D

The results of all the ECDF tests showed, that the combined selection method improves general fitness of a population significantly. Nevertheless, its capability of improving the best solution can not be deduced from these results and still remains unknown. The following chapter shall provide tests with complete evolutionary algorithms, that would expose the actual efficiency of use of the crossover success estimation in selection methods.

Chapter 5

Testing of EAs

To examine performance of evolutionary algorithms incorporating a selection based on the crossover success estimation, various evolutionary algorithms, each using a different selection method, were implemented. As it was explained in the previous chapter, the only component in which the algorithms differ is a selection method. The following text introduces the implemented EAs, and describes the sequence of performed tests and their results.

5.1 No-Selection EA

The first implemented EA uses only the random selection for selecting parents from a population and thus, it is labeled as an EA with no selection. The selection pressure in this algorithm is generated only by the RTR replacement method. Due to the poor selection method used in this EA, no prominent results are being expected from this implementation.

5.2 Classic EA

The second algorithm implemented for comparison of its successfulness with the crossover success estimation based EAs, is a classic, most frequently used version of an evolutionary algorithm. This version uses the tournament selection method to select pairs of parents for crossover process.

5.3 Complexity EA

The first EA that incorporates the crossover success estimation, uses the complexity selection, and thus will be labelled as the complexity EA in further text.

The first performed tests compared the three above described EAs. An initial population was generated, that served as an input for all three evolutionary algorithms. Throughout the run of each algorithm, the fitness value of the best found individual in each generation, was being continuously recorded. The algorithms were stopped after the 100^{th} generation was created and evaluated. The tests were performed on optimisation of all six objective functions in dimensions 5, 10 and 20.

The resulting graph for optimisation of the Griewank1 function in 5-dimensional space is shown in Figure 5.1. The graph shows that the convergence of the best fitness value is the fastest in the complexity EA and this algorithm, out of the three, finds the best solution.

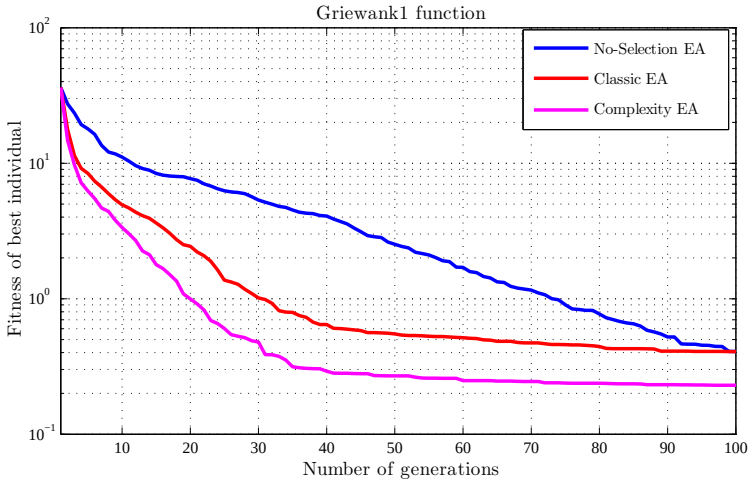


Figure 5.1: Optimisation of the Griewank1 function in 5D

The second graph, shown in Figure 5.2, denotes the performance of all three algorithms on the same function as in the previous test, but this time in the space of dimension 20. This time, the classic EA reaches the best solution, while the complexity EA has much slower convergence. The result of the EA with no selection is very poor in this case.

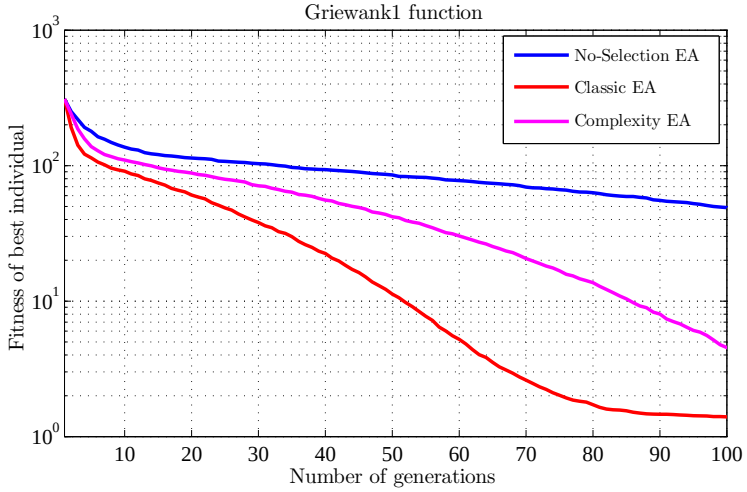


Figure 5.2: Optimisation of the Griewank1 function in 20D

Results of test on the optimisation of the Griewank2 function in the space of dimension 10, are portrayed in the graph in Figure 5.3. From all the performed tests, this brought one

of the complexity EA's worst results. The Griewank2 is a function of small domain with many elevations and the crossover success method doesn't appear to be profitable under such conditions.

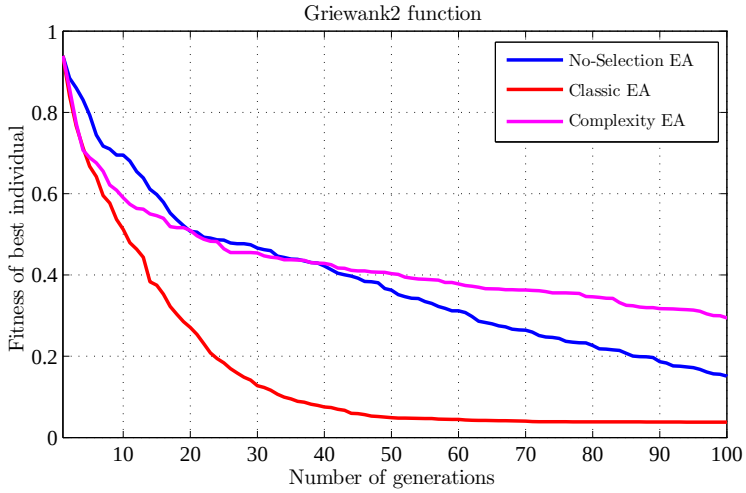


Figure 5.3: Optimisation of the Griewank2 function in 10D

The last graph, in Figure 5.4, shows results of optimisation of the Rosenbrock function in dimension 5. The runs of both, the classic and the complexity EA, are very similar and the amount by which the complexity EA wins, in the means of fitness of the best found solution, is minimal considering that the y-axis of the graph is logarithmic.

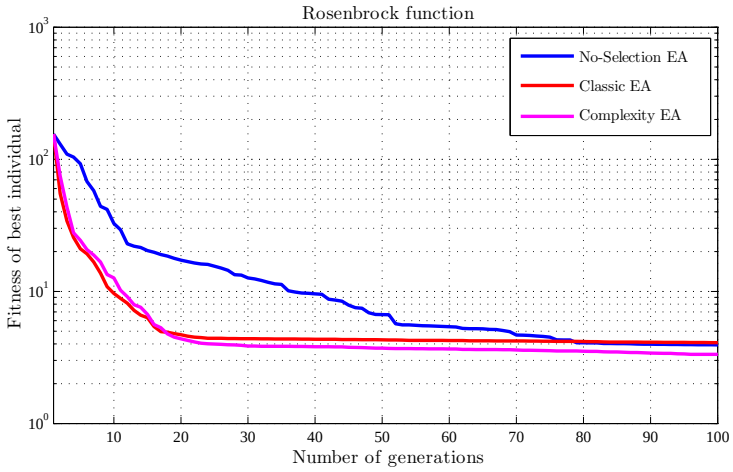


Figure 5.4: Optimisation of the Rosenbrock function in 5D

The testing brought satisfactory results of the complexity EA in optimisation of objective function in a smaller dimension. Nevertheless, in case of the optimisation in dimensions 10

and 20, the results for this variant of an EA, were mostly unsatisfactory and the algorithm's performance was easily exceeded by the classic EA using the tournament selection.

5.4 Combined EA

The lastly implemented algorithm, the combined EA, was inspired by insufficient results of the complexity EA. This variant of an EA uses the combined selection method for selecting parents to crossover process.

In the next part of testing, all of the objective functions were being optimised by the four variants of an EA. Results of optimisation of the Griewank1 function in dimension 10 can be seen in Figure 5.5. Convergence of the best fitness value in the combined EA is quite fast in the first place, but the algorithm gets stuck in a local minimum at around the 30^{th} generation, and later, it is overcame by the other EAs.

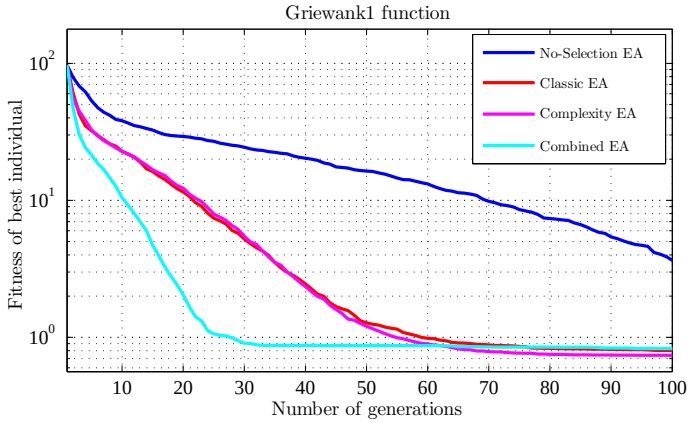


Figure 5.5: Optimisation of the Griewank1 function in 10D

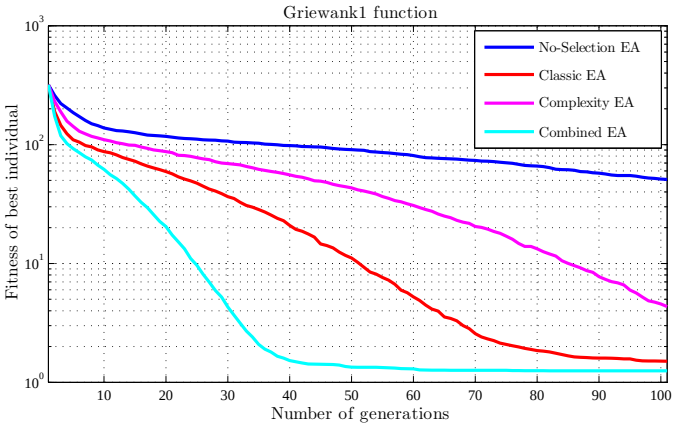


Figure 5.6: Optimisation of the Griewank1 function in 20D

When optimising the same objective function in the space of higher dimension, the combined algorithm gets the best results out of the all four algorithms, as the graph in Figure 5.6 shows. This is contrary to behaviour of the complexity EA, which appeared to overcome the other EAs rather in a space of small dimension.

Performance of the four EAs in optimisation of the linear function in 5-dimensional space is shown in Figure 5.7. The convergence of the best individual's fitness in the combined EA is by far the fastest out of the four. Furthermore, the combined EA proved to be very efficient in minimasation of the linear function, independently of the problem dimension.

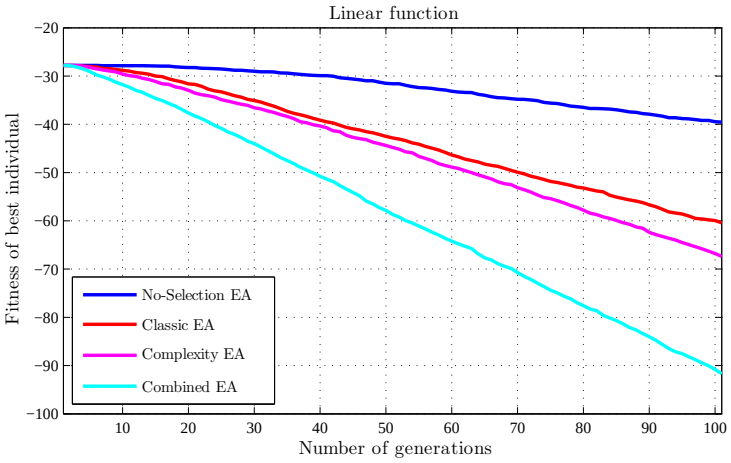


Figure 5.7: Optimisation of the linear function in 5D

The results for a different objective function, the ellipsoid in dimension 20, are portrayed in Figure 5.8. Convergence of the combined EA is the fastest and it finds the best solution out of the four EAs, yet the result is not very distinct from the result of the classic EA.

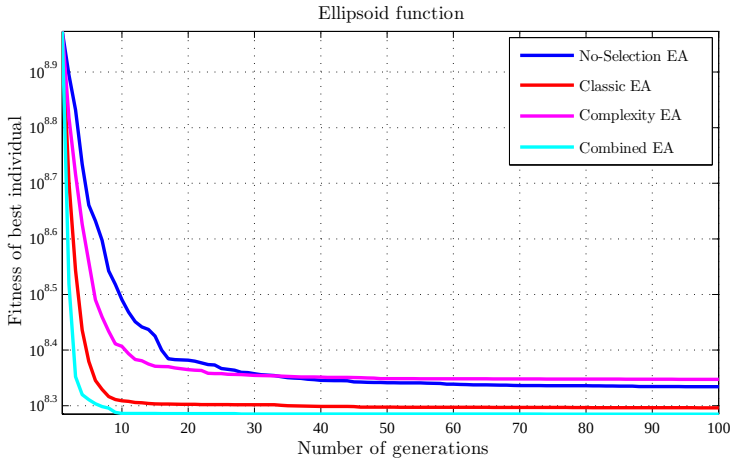


Figure 5.8: Optimisation of the ellipsoid function in 20D

Moreover, the combined EA's performance on the same function in lower dimension is unsatisfactory, as the graph in Figure 5.9 shows. In the lower dimension of this problem, the combined EA provides the worst result, as well as in optimisation of some other objective functions (e.g. the Griewank2 function).

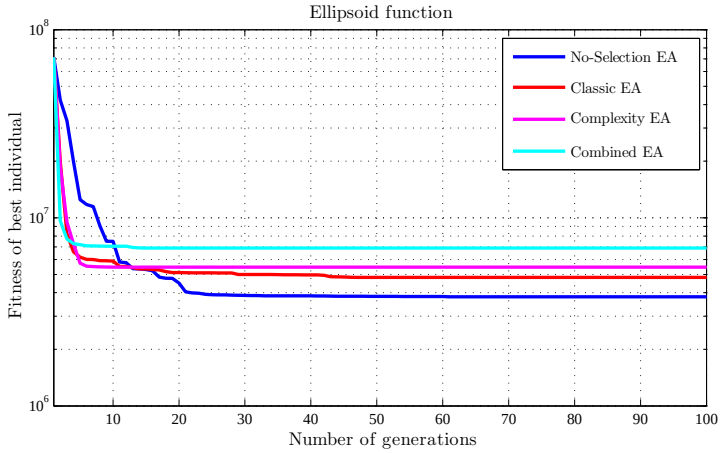


Figure 5.9: Optimisation of the ellipsoid function in 5D

In order to determine, where the problem resides in, we tried measuring the correlation between the complexity of parents and fitness of their offspring throughout the whole run of the evolutionary algorithm. Both correlation coefficients, the Pearson and the Spearman, were being computed and recorded. The development of the Pearson correlation coefficient during the run of all the algorithms in optimisation of the same problem as in the previous test, can be seen in Figure 5.10. As we focus on the combined EA, the correlation curve for this algorithm shall be examined. We can observe, that the correlation is really high at the beginning of the optimisation, but it gradually descends with the number of generations.

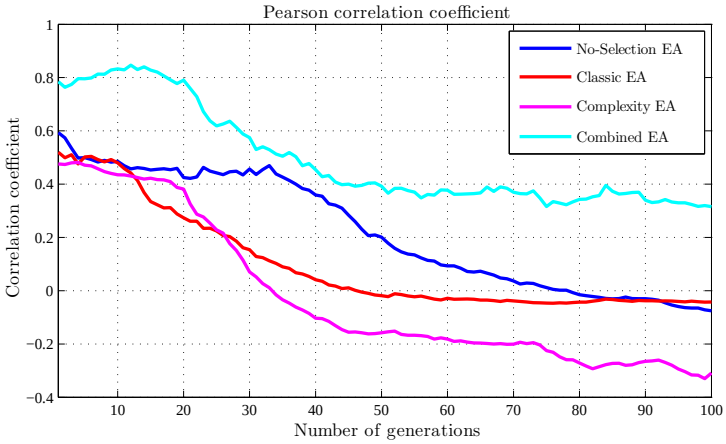


Figure 5.10: Pearson correlation in optimisation of the ellipsoid function in 5D

As the Pearson correlation coefficient was being measured as well, its development throughout the 100 generations is portrayed in Figure 5.11. Same as in the case of the Pearson correlation, the Spearman correlation coefficient is very high at first, but descends in time. By the middle of the run, the correlation coefficient even reaches 0.

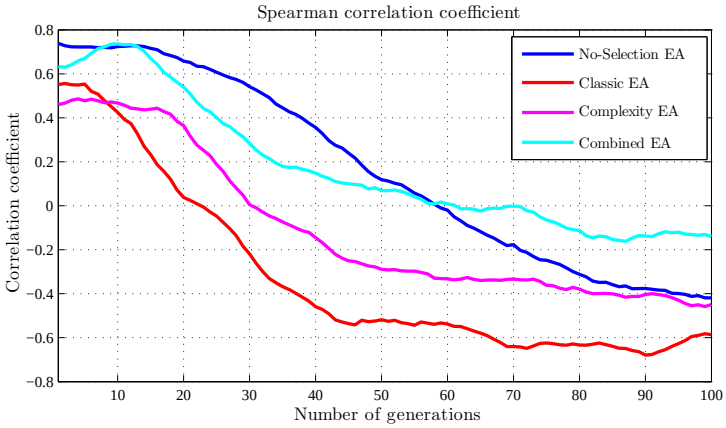


Figure 5.11: Spearman correlation in optimisation of the ellipsoid function in 5D

The behaviour of the complexity algorithm in optimisation of the Griewank2 function was comparable with its performance on the ellipsoid function. The result of optimisation tests of all the algorithms on this function is shown in Figure 5.12.

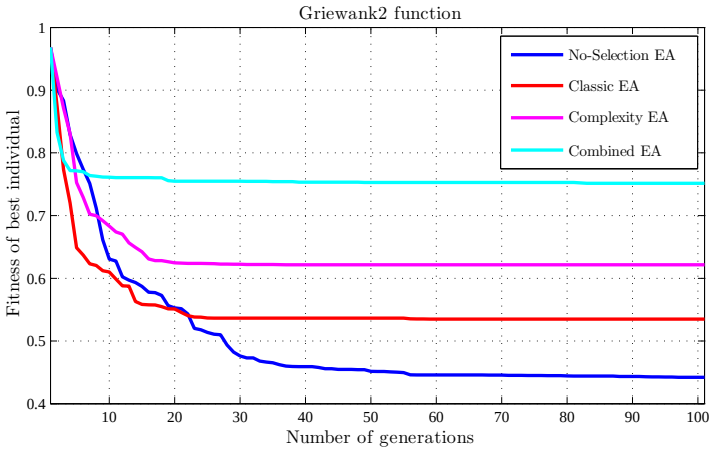


Figure 5.12: Optimisation of the Griewank2 function in 10D

Corresponding graphs of the development of the Pearson and Spearman correlation coefficients are shown in Figures 5.13 and 5.14. As it can be seen in the graphs, both correlation coefficients for the combined EA are very low from the beginning of the evolution up to its end. Primarily, the Spearman correlation coefficient oscillates around the zero value for the whole run of the algorithm.

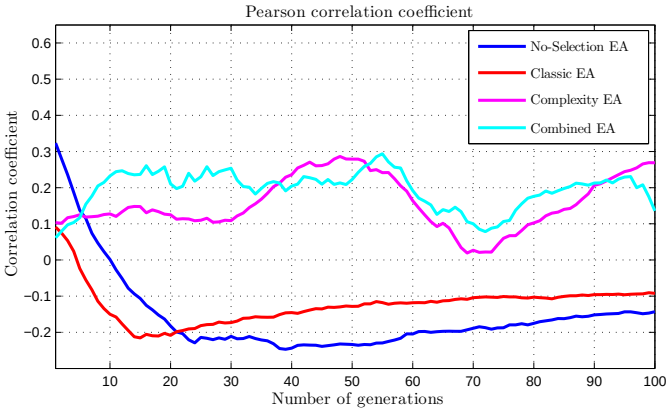


Figure 5.13: Pearson correlation in optimisation of the Griewank2 function in 10D

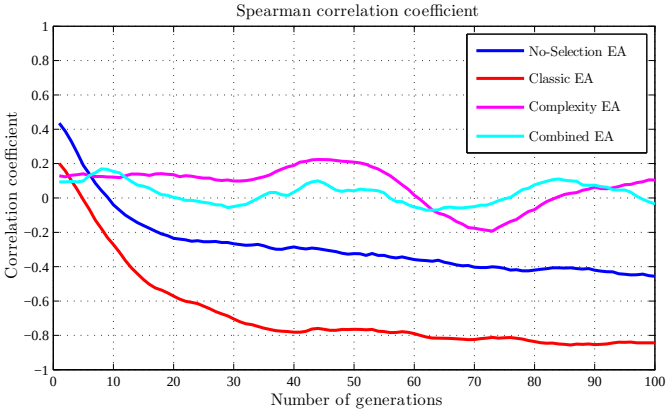


Figure 5.14: Spearman correlation in optimisation of the Griewank2 function in 10D

According to the results of performed tests, some of which are shown above, it appears that the change of correlation during the run of the combined algorithm might indicate how successful the algorithm is in searching for better fitness values. An idea came up, that an evolutionary algorithm using the combined selection method could be driven by the correlation of the complexity of crossed over parents and fitness value of the produced offspring. If the correlation coefficients decreases below a given value, this selection method would be turned off.

5.5 Switching EA

The last, additionally implemented, variant of an evolutionary algorithm uses the combined selection method for selection of parents for crossover process. During the whole run of the algorithm, the Spearman correlation of the crossed over parents' complexity and fitness values of the produced offspring is being measured. When the correlation falls bellow a

certain defined value, the switching EA turns off the combined selection method and switches to the tournament selection method.

The implementation of the switching algorithm used for testing, starts with the combined selection method and uses it for at least first 5 generations, because the previous observations showed that convergence of an algorithm using this selection method, is in the first 5 generations always the fastest among all tested EAs. The selection method is switched to the tournament, if the correlation coefficient in previous 5 generations in a row is below the value 0.5. Results of test performed on the Griewank2 function are shown in Figure 5.15. The algorithm switching from the combined to tournament selection has visibly the best performance, although the combined EA itself produces the worst result in this case.

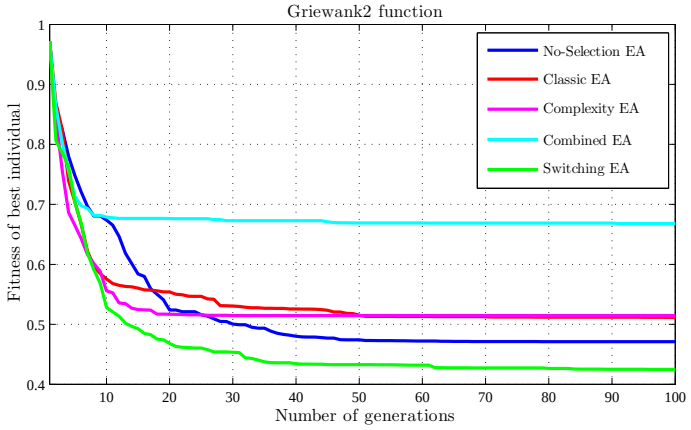


Figure 5.15: Optimisation of the Griewank2 function in 10D

The very last graph in Figure 5.16 demonstrates that the use of selection switching does not cause harm to performance in optimisation of objective functions, where the combined EA alone is successful.

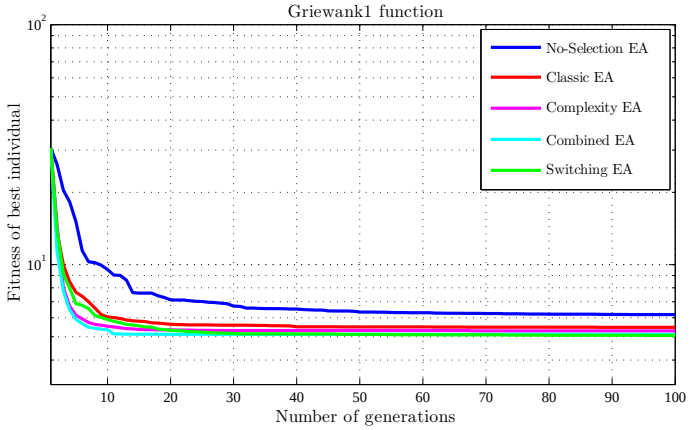


Figure 5.16: Optimisation of the Griewank1 function in 5D

Chapter 6

Conclusions

In this bachelor project, existing methods of global optimisation, particularly the *S.T.E.P.* and *DIRECT* algorithms, were examined and served as an inspiration for creating methods of estimating the crossover operation success. Two different crossover success estimation methods were proposed—the *linear crossover complexity method* and the *parabolic crossover complexity method*.

Both proposed methods were implemented and then extensively tested to determine successfulness of the provided estimation. Correlation tests comprising the computation of the *Pearson* and the *Spearman correlation coefficient*, were performed to analyse possible relation between the crossover complexity of a pair of parents, provided by the crossover success estimation methods, and the actual fitness value of the offspring reproduced by the designated pair.

The correlation tests revealed that there is a significant correlation between the complexity of pairs and fitness of the offspring. Based on the results of the correlation tests, the linear crossover success estimation was elected to participate in a selection method for selecting parents from a population to crossover process.

Two different selection methods established on the use of crossover success estimation method were proposed. First of these methods, the *complexity selection*, chooses pairs of individuals from population that have minimal crossover complexity, computed as the crossover success estimation. The other selection method, the *combined selection*, chooses a pair of individuals by combining a standard *tournament selection method* and the complexity selection method. The first of the parents is selected based on the tournament selection and the other parent is chosen with objective to minimise the complexity of the whole pair.

Both selection methods, along with the tournament selection and random selection, were implemented and their performance on a single generational step of the evolution in EA was tested through analysis of fitness distribution using the *Empirical Cumulative Distribution Function*. The results of tests were promising mostly for the combined selection method.

All created selection methods were incorporated into four variants of an evolutionary algorithm, where each variant varied from the others only in the used selection method. All four implemented EAs were widely tested on optimisation of the six included objective functions in dimensions of 5, 10 and 20. While the complexity EA using the complexity selection method performed well on the problems of lower dimension, the combined EA

using the combined selection, on the contrary, performed much better in the space of higher dimension, such as 10 or 20.

In optimisation of most of the objective function, the combined EA found the best solution out of the four used EAs and proved to be sufficiently successful. Nevertheless, the algorithm performed poorly on the optimisation of some of the functions, when searching in a space of small dimension. It also failed to optimise functions of small domain, such as the Rosenbrock function, and functions with many peaks, such as the Griewank2 function.

After the results of optimisation tests were examined, we tried to measure correlations between complexity of parents and fitness of the offspring again, but this time in each generation of the evolution during the whole run of each evolutionary algorithm. These tests indicated that performance of the combined EA may be affected by the correlation coefficients in each generation.

Thus, the final algorithm was proposed, that would start with the combined selection method and measure the correlations during its run, while the Spearman correlation is used. If the correlation coefficient of a population in five generations in a row is bellow a defined value, the algorithm turns off the combined selection method and switches to the tournament selection. The tournament selection is then used for the rest of the run of the algorithm.

The switching EA was implemented and hastily tested, providing promising results. The algorithm performed very well on the objective functions on which the results of the combined EA were poor, while it did not take away the beneficial attributes of the combined selection method demonstrated on the other functions. Nevertheless, the switching evolutionary algorithm would require additional testing and tuning.

The whole idea of the crossover success estimation is very interesting and would deserve further research. In this bachelor project, only one of the two proposed crossover success estimation methods was widely examined, while the other one was omitted half way to the end of the project. To examine further behaviour of both of the crossover complexity methods, additional tests could be performed.

That would include optimisation of other well-known objective functions, in more different dimensions, if possible. Additional modifications could also be done to the used parameters, such as the correlation limit for switching from the combined selection to the tournament selection method in the switching EA. An option of switching back to the combined selection, if the correlation increases, would also deserve consideration.

Bibliography

[1] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* The University of Michigan Press, 1975.

[2] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[3] Robert E. Smith, B. A. Dike. Fitness Inheritance in Genetic Algorithms. *SAC '95, Proceedings of the 1995 ACM Symposium on Applied Computing*, 1995.

[4] Alain Ratle. Optimal Sampling Strategies for Learning a Fitness Model. *CEC '99, Proceedings of the 1999 Congress on Evolutionary Computation.*

[5] Els Ducheyne, Bernard De Baets, Robert De Wulf. Is Fitness Inheritance Useful for Real-World Applications? *Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science*, 2003.

[6] D. R. Jones, C. D. Perttunen, B. E. Stuckman. Lipschitzian Optimization Without the Lipschitz Constant. *Journal of Optimization Theory and Application*, 1993.

[7] Daniel E. Finkel. *DIRECT Optimization Algorithm User Guide.* Center for Research in Scientific Computation, North Carolina State University, 2003.

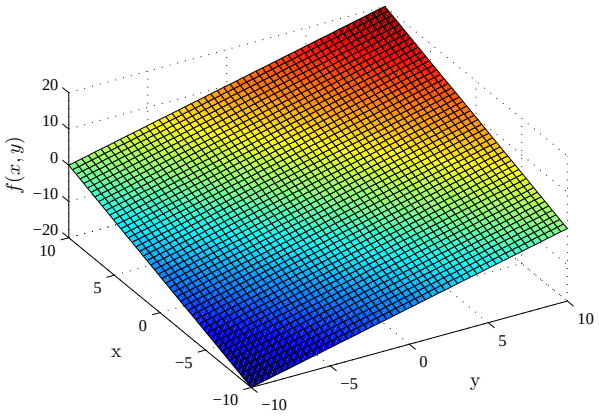
[8] S. Swarzberg, G. Seront, H. Bersini. S.T.E.P.: The Easiest Way To Optimize a Function. *WCCI '94, Proceedings of the IEEE World Congress on Computational Intelligence*, 1994.

[9] Claudio F. Lima, Carlos Fernandes, Fernando G. Lobo. Investigating Restricted Tournament Replacement in ECGA for Non-Stationary Environments. *GECCO '08, Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 2008.

Appendix A

Objective functions

A.1 Linear function

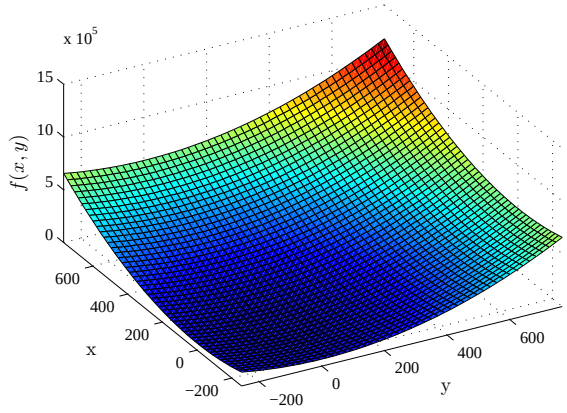


Function domain:
$[-10, 10]$

Position of global minimum:
$(x, y) = (-10, -10)$

Fitness at global minimum:
$f(x, y) = -20$

A.2 Sphere function

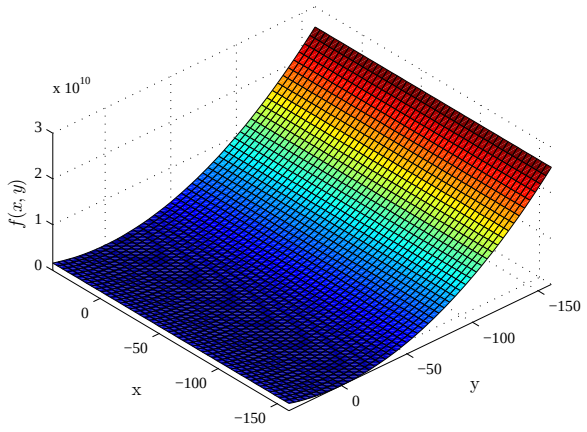


Function domain:
$[-256, 768]$

Position of global minimum:
$(x, y) = (0, 0)$

Fitness at global minimum:
$f(x, y) = 0$

A.3 Ellipsoid function

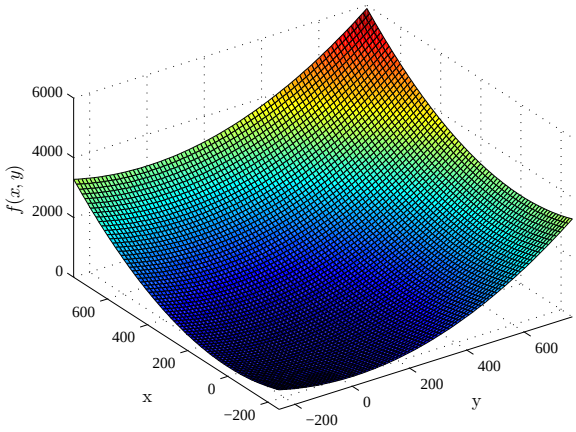


Function domain:
$[-160, 40]$

Position of global minimum:
$(x, y) = (x, 0)$

Fitness at global minimum:
$f(x, y) = 0$

A.4 Griewank1 function

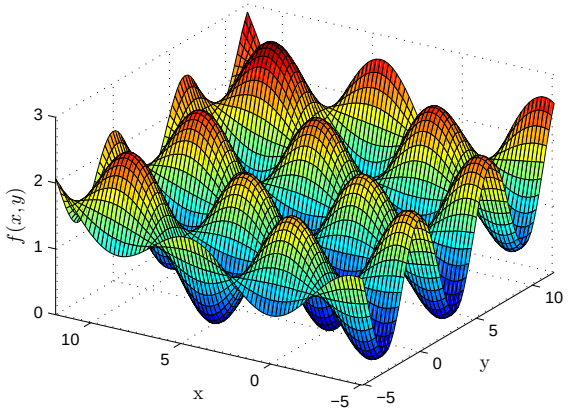


Function domain:
$[-256, 768]$

Position of global minimum:
$(x, y) = (0, 0)$

Fitness at global minimum:
$f(x, y) = 0$

A.5 Griewank2 function

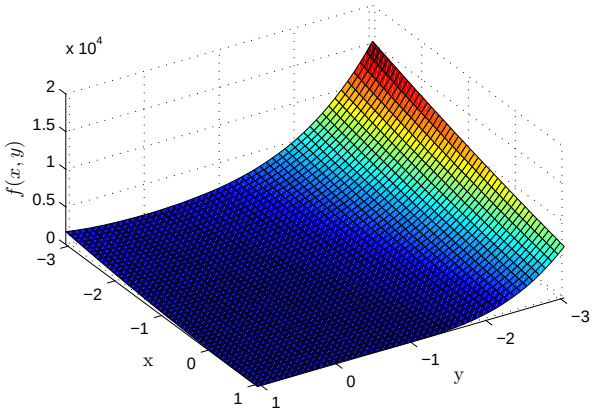


Function domain:
$[-5.12, 11.9467]$

Position of global minimum:
$(x, y) = (0, 0)$

Fitness at global minimum:
$f(x, y) = 0$

A.6 Rosenbrock function



Function domain:
$[-3.048, 1.048]$

Position of global minimum:
$(x, y) = (1, 1)$

Fitness at global minimum:
$f(x, y) = 0$

Appendix B

Programming environment

Since the character of this work is rather research-oriented than implementation-oriented, all programming was done in MATLAB, a computing environment and a programming language. The MATLAB environment was selected for its ability of easy manipulation with matrices, computing of mathematical functions, and plotting data and functions.

The programmed source codes, included on a CD enclosed with this bachelor thesis, consist of MATLAB scripts and functions in M-Files. The following two sections describe purpose of every file.

B.1 Scripts

correlation_tests.m Script for execution of correlation tests of complexity of a pair and fitness value of the produced offspring.

comparison_of_simulated_EAs.m Script for testing of selection methods on simulated partially converged state of evolution.

comparison_of_EAs.m Script used for final testing of performance of complete evolutionary algorithms on optimisation problems.

B.2 Functions

measureNextGeneration.m Function for execution of one generational step from the initial state of evolution.

measureLastGeneration.m Function for execution of one generational step from the simulated partially converged state of evolution.

drawCorrelGraph.m Function for plotting of correlation graphs.

drawCompareGraph.m Function for plotting of the best fitness value development in EA.

generatePopulation.m Function for generation of random initial population.

selectionRandom.m Function for random selection of two different individuals.

selectionTournament.m Function for tournament selection of two different individuals.

calculateDistance.m Function for calculation of Euclidean distance between two points.

getLineCrossComplexity.m Function for computation of linear crossover complexity.

getParabolicComplexity.m Function for computation of parabolic crossover complexity.

lineCrossBreed.m Function for crossover of two parents.

mutateGaussian.m Function for Gaussian mutation of an individual.

replacementRestrictedTournament.m Function for restricted tournament replacement.

stepClassicEA.m Function for execution of one generational step of classic EA.

stepComplexityEA.m Function for execution of one generational step of complexity EA.

stepCombinedEA.m Function for execution of one generational step of combined EA.

fullClassicEA.m Function for the complete run of classic EA.

fullComplexityEA.m Function for the complete run of complexity EA.

fullCombinedEA.m Function for the complete run of combined EA.

fullSwitchingEA.m Function for the complete run of switching EA.

getFitnessLinear.m Function for fitness evaluation of an individual in linear function.

getFitnessSphere.m Function for fitness evaluation of an individual in sphere function.

getFitnessEllipsoid.m Function for fitness evaluation of an individual in ellipsoid function.

getFitnessGriewank1.m Function for fitness evaluation in Griewank1 function.

getFitnessGriewank2.m Function for fitness evaluation in Griewank2 function.

getFitnessRosenbrock.m Function for fitness evaluation in Rosenbrock function.

Appendix C

List of abbreviations

5D Dimension 5

10D Dimension 10

20D Dimension 20

DIRECT DIviding RECTangles

EA Evolutionary Algorithm

ECDF Empirical Cumulative Distribution Function

GA Genetic Algorithm

MATLAB MATrix LABoratory

RTR Restricted Tournament Replacement

S.T.E.P. Select The Easiest Point